FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Provision of Service Level Agreements in Human-Enhanced Service-Oriented Computing Environments

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Roman Khazankin
Registration Number 0927683

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Prof. Dr. Schahram Dustdar

The dissertation has been reviewed by:

(Prof. Dr. Schahram Dustdar)

(Prof. Dr. Frank Leymann, University of Stuttgart)

Wien, 20.06.2012

(Roman Khazankin)

Acknowledgements

First of all, I am very grateful to Prof. Schahram Dustdar for the opportunity to be a part of the Distributed Systems Group (DSG) and to carry out my PhD studies under his supervision. He greatly supported me during all phases of my research, and I highly appreciate his assistance in determining my research goals, and his encouragement in pursuing them. I am convinced that this mentoring played a key role in realization of my potential. I would also like to thank Prof. Frank Leymann from the University of Stuttgart for being my examiner.

I am thankful to the Vienna Science and Technology Fund (WWTF) for the financial support during my studies (project ICT08-032).

Another source of motivation and inspiration were my colleagues, who helped me to keep an optimistic attitude throughout my time at DSG, to get rid of doubts regarding my work, and to substantially improve it. In particular, I would like to thank Christian Inzinger, Lukasz Juszczyk, Vitaliy Liptchinsky, Michael Maurer, Emmanuel Mulo, Harald Psaier, Benjamin Satzger, Daniel Schall, Florian Skopik, Martin Treiber, and Martin Vasko. I had a great time working with you all!

Back in school days, my life was strongly influenced by several people. Some of them might have been a reason my life became the way it is, and, subsequently, the reason this thesis exists. I specifically would like to acknowledge Albert Efimov, Vitaliy Kharkin, and Rustem Zhenodarov who taught me IT-related things and inspired me to continue developing myself in this area. Also, I would like to thank Marina Filtser, the best English teacher I ever knew.

Most importantly, I am very grateful to my family. I want to thank my parents Veniamin and Ludmila who have constantly supported me over my entire life and created many conditions for my success; I am infinitely thankful to my precious wife Albina for her patience, understanding, and love. It is you who reminds me every day why life is wonderful, and inspires me to achieve more.

Roman Khazankin
June, 2012
Vienna, Austria

Abstract

Deploying Service-Oriented Architecture (SOA) in enterprises has become mainstream, as it provides business agility benefits. Today's technologies allow to integrate human labour together with software services into service-oriented architectures therefore achieving smooth automation of business processes in such mixed systems. However, companies are continuously trying to optimize expenses associated with human labour. This results into scarcity of available resources and switching to more flexible paradigms of employment such as crowdsourcing. At the same time there is need to sustain the competitiveness by improving the quality of provided services and assuring Service Level Agreements (SLAs). However, the inherent uncertainty of crowdsourcing environments and the impact of human factors increase risks and complicate service level management. Although there is a solid amount of research on SLA-based optimization, and techniques for prevention of SLA violations in SOA, however, it does not take into account the specific features of human-provided and crowdsourced services.

This thesis addresses the problems by identifying specific control mechanisms and using adaptive techniques that can appropriately react to unexpected events and dynamically reconfigure certain processes characteristics or resources. It investigates SLA provision in crowdsourcing environments and adaptable techniques for SLA-aware execution of business process dependent on human-provided services. It presents, architectures, methods, and algorithms for SLA negotiation and fulfillment, and cost-based optimization in such environments.

Contents

CHAPTER 1

Introduction

The ever-growing information technologies industry intensifies the demand for system integration, and service-oriented approach for integration has become ubiquitous. Service compositions are again provided as services, thus enriching the consumer's choice and giving him/her alternatives regarding the trade-off between cost, quality, simplicity, and trust, while allowing the provider to benefit from the added value. Deploying Service-Oriented Architecture (SOA) in enterprises is considered mainstream, as it provides business agility benefits [30]. From business-to-business and business-to-consumer perspectives, offering services online has long ago become not a feature, but a business model.

Service providers can give certain guarantees for their services in form of Service Level Agreements (SLAs). From a business process management perspective, such guarantees can deliver significant additional value for services, as they can strengthen the certainty and predictability in process planning and design, and allow the consumers to use the provided services in more critical business processes [1, 2, 26]. To fulfill given guarantees, a service provider should negotiate an agreement which is feasible in the first place, and so that already established agreements are not endangered. Then, the provider needs to ensure the agreement during its period of validity. All that should take into account underlying resources and systems to make sure that there are enough resources to fulfill the agreement and the system is compliant with it. Moreover, the activated resources should be balanced in a way to minimize the expenses while fulfilling negotiated SLAs.

While enterprises seek to automate all sorts of their front and back office operations to cut costs and eliminate human factors, humans inevitably remain as key elements of many business processes. There is a plethora of activities that are hard to reproduce with technology. Work concerned with creativity, management, or social communication, performed by a machine in a real business process is still hardly imaginable. Even basic activities, such as image recognition or categorization, as well as those requiring specific skills, such as text translation or usability testing, are difficult to fulfill with software only.

Integration of human labour into service-oriented architectures was investigated both in research [76] and in industry [4, 24]. The developed frameworks however mainly provide capa-

bilities of interfacing with people in SOA. They allow to communicate with the workforce via standardized interfaces like Web Services and enable seamless integration of human-provided services into processes expressed with a standardized process execution language [43].

1.1 Problem Statement

While trying to optimize expenses associated with human labour, companies have to improve the quality of provided services in order to stay competitive. However, the resulting scarcity of resources increases the impact of human factors, flexible demand, and unexpected situations on the running processes, as they become harder to escalate. These issues become even more critical when the services dependent on human resources have SLAs bound:

- Unexpected situations, such as when service failures occur, when demand gets higher than anticipated, or when unplanned structural changes happen in a process, can obviously affect the fulfillment of SLAs. In context of human work it can result in temporary situations when there are simply not enough human resources to sustain the workload. From the SLA perspective, it means that response time and throughput guarantees for particular customers might be violated.
- Flexible demand poses a dilemma for management: either the sufficient workforce should be kept and thus periodically be idle, or it can be reduced, but then peak loads would result into performance bottlenecks. For certain types of labour, flexible demand can be met by using recently emerged paradigms of employment - internal and external crowdsourcing [18, 33]. Crowdsourcing allows to outsource tasks to an undefined network of people via a crowdsourcing platform maintained either within the enterprise (internal crowdsourcing) or by a third party (external crowdsourcing). However, current crowdsourcing solutions do not provide guarantees regarding task processing time. A customer can offer higher payments for tasks, which can of course shorten the processing time, but, nevertheless, no exact guarantees are given to him/her, and it is not clear which reward should be given to actually have the task done in time. Also, task processing quality in external crowdsourcing is not trivial to control, as *anyone* can register in the platform. Although some platforms provide the possibility to put qualification requirements on tasks, there are no mechanisms that are able to ensure that a task is going to be processed with this quality before the certain deadline.
- Human factors can never be eliminated completely due to the human nature, however, proper software tools can reduce the risk associated with them. Certain Service Level Objectives (SLOs), such as security and privacy concerns, can be directly affected by both underlying information systems and people that manage, configure, and work with these systems. Therefore, to ensure the level of the provided services, workers need to be equipped with special tools that allow to analyze processes flowing inside the company, align them with feasible SLAs, and prevent the misuse of data received from consumers.

On the one hand, many studies have been devoted to ensuring SLAs using various mechanisms, such as modification or re-composition of business process, replacement of services,

or re-configuration of service platforms and computational resources, i.e., load balancing [91]. While these methods are applicable for software services that are mainly constrained by hardware resources, they do not take into account the specific features of human-provided and crowdsourced services. On the other hand, a solid amount of research has been done in the field of workforce optimization which focuses on how to efficiently align the demand with the workforce size and scheduling [21, 28]. While the addressed problems are relevant, there is no clear way of applying the results of this research to SOAs, as the addressed problems are usually strict and devoted to very particular industry cases.

Currently available technologies and approaches are therefore not sufficient for creating solid human-enhanced service-oriented systems that allow to control the guaranteed level of service. This thesis focuses on the these shortcomings and aims to resolve the following generalized problems:

- *How can we influence business processes that comprise human-provided or crowdsourced services in a way that allows us to keep them compliant to established SLAs?*
- *From the service provider's point of view, how can we effectively estimate feasible Service Level Objectives for crowdsourced and human-provided services?*
- *How can we improve the quality and level of such services?*

The thesis considers these problems in different contexts where human work can be regarded as a key property. The next section describes those areas in detail.

1.2 Contributions

With the focus on service-oriented systems that comprise human-provided services, the main contributions of the thesis are to investigates fine-grain control mechanisms, architectures, and effective algorithms, that allow to negotiate feasible SLAs based on available resources and current processes states, perform dynamic SLA-based adaptations to ensure the fulfillment of agreements, improve overall Quality of Service (QoS), and reduce costs of process execution while adhering to SLAs. Specifically, the following areas are considered:

- **Adaptive request prioritization in process engines**

  Humans naturally work at lower speeds than software-based services. If such "low-throughput" services are part of concurrent process, the order in which process instances get the response from them can significantly impact process execution times. If the system gets temporarily overloaded or some processes are late due to unexpected delays, the SLAs might be violated due to the shortage of human resources. However, the control over the ordering of service requests can reduce penalties or prevent violations at all. This can be achieved by transferring the priority from less to more critical processes. In this context, the work comprises the following contributions:

  – A conceptual architecture that enables the support for dynamic re-prioritization of requests by business process engines.

  - A prioritizing algorithm that minimizes penalties based on established SLAs and execution states of running processes at runtime.
  - An evaluation of the performance and comparison with the conventional First-In-First-Out request processing.

- **SLA-driven provision of crowdsourcing services**

  Current crowdsourcing providers do not provide guarantees regarding task processing time therefore prohibiting the use of SLAs for crowdsourced services. This is mainly because their architecture follows a market-like operation chain where a platform announces received tasks at the web-portal and then workers choose among the assorted mass of task that they like to process. The selection is thus motivated by the personal preferences of workers, and is not influenced by any additional mechanisms. To address these issues, the platform architecture can be altered so that the workers are assigned to tasks *by the platform*. In this case workers register at the platform, provide information about their skills and availability, and expect the platform to allocate tasks to them accordingly. This approach provides fine-grain control over assignments, and makes it possible to establish SLAs. However, this requires to resolve such issues as uncertainties in availability of human resources, diversity of workers in terms of their skills, and prediction of platform capacity. This work investigates this approach and constitutes the following corresponding contributions:

  - A conceptual platform architecture, *scheduled crowdsourcing*, that support SLA-based task processing.
  - Algorithms for estimating feasible level objectives, SLA negotiation, and SLA-based task assignment in a scheduled crowdsourcing platform.
  - Analysis of the overall service level improvement in scheduling crowdsourcing by comparison the with different task assignment approaches.

- **Incentive-based approach for SLA-aware business processes execution on crowdsourcing platforms**

  Big companies have recognized the advantages of a flexible process management approach [33] where the tasks that need to be done are announced, e.g., in an internal crowdsourcing platform, and executors are selected among internal employees or external contractors via *competition*. People book tasks voluntarily in competition-based crowdsourcing, which means the only way to influence booking and execution times of single tasks is to either change incentives or modify other aspects of a task, e.g., define a later deadline. Given a process that comprises a number of such crowdsourced tasks, it is not clear how to cope with SLAs, i.e., how to adjust the aforementioned aspects of tasks so that the process is completed in concordance with the given guarantees. Moreover, a company is interested in reducing overall operational costs, therefore, the parameters should be chosen in a way to minimize the cost associated with crowdsourcing while adhering to SLAs. The thesis provides the following contributions to address these problems:

  - An approach for SLA-aware execution of business processes on top of an internal crowdsourcing platform.
  - Algorithms for statistical learning of crowd characteristics and for cost- and SLA-based process optimization.
  - An evaluation of efficiency and performance of the approach for diverse processes characteristics.

- **Monitoring of data usage to negotiate and ensure privacy-related objectives**

  Companies are responsible for the data obtained from services that are offered to customers or partners. However, the inappropriate use of data can happen unintentionally due to human factors in process and software design, and data integration activities, as the data comes to be maintained in various locations by different parties. This is especially important for privacy terms stated in an SLA. With that respect, big companies have a problem to track the usage of private data throughout their information systems and guarantee that it is used according to the negotiated agreements. Our contributions for this problem are the following:

  - An architecture for tracking of private data usage throughout the service-oriented enterprise system.
  - Mechanisms to assess the feasible guarantees that can be included into an SLA regarding the private data.
  - Mechanisms to prevent the privacy-related SLA violations.

For all of the proposed solutions, prototype implementations were performed and evaluated. To facilitate evaluations, the simulated environments that reproduce the problems addressed by the solutions (e.g., flexible demand, unexpected situations, and service failures) were designed.

1.3 Evaluation Approach

The goals of the thesis are to highlight the capabilities and prove the feasibility of the proposed novel methods and architectures, as well as to discover their limitations and applicability in the domain.

Due to the nature of the contributions, they cannot be tested on benchmarks or some statistical data. As the proposed solutions result in a direct impact on the behavior of software systems, the evaluation requires feedback which describes the effect of deploying the solutions, and which can only be retrieved from a real system. The evaluation in real business environments would however require certain efforts and up-front investments for developing quality software and re-designing the corresponding systems, as the contributions propose significant changes in architectures of large-scale and critical systems. Because of these reasons, we were not able to perform experiments on real data, and the evaluation has been done through the simulation of corresponding environments and system, i.e., service-oriented systems, business process execution engines, and crowdsourcing platforms.

Although we tried to prognosticate the meaningful parameters and to consider a wide range of setups for simulations, the effectiveness of algorithms depends on particular business settings and requirements which can still vary from the simulated setting to certain extent. This was the reason why we did not intend to achieve best performance, but rather to design base solutions which still deliver value, but can be perceived and reproduced without a tremendous effort. In practical use, these solutions can be extended to take into account the specifics of a particular business setup.

1.4 Published Papers

The results of this research have been published at acknowledged international conferences in the domain.

1. *Predicting QoS in Scheduled Crowdsourcing.*

   Khazankin R., Schall D., Dustdar S.

   The 24th International Conference on Advanced Information Systems Engineering (CAiSE'12), June 13-16, 2012, Gdansk, Poland.

2. *QoS-based Task Scheduling in Crowdsourcing Environments.*

   Khazankin R., Psaier H., Schall D., Dustdar S.

   9th International Conference on Service Oriented Computing (ICSOC'11), December 5-8, 2011, Paphos, Cyprus.

3. *Adaptive Request Prioritization in Dynamic Service-oriented Systems.*

   Khazankin R., Schall D., Dustdar S.

   The 8th International Conference on Services Computing (SCC'11), July 4-9, 2011, Washington DC, USA. **Best student paper award.**

4. *PROVIDENCE: A Framework for Private Data Propagation Control in Service-Oriented Systems.*

   Khazankin R., Dustdar S.

   ServiceWave 2010, December 13 - 15, 2010, Ghent, Belgium.

1.5 Structure of the Thesis

The structure of the thesis is strongly aligned with its contributions. The content is mainly split into three parts:

1. The introductory part which gives an overview of state of the art in areas that this thesis is concerned with. Chapter 2 explains how the research is positioned within the domains of service-oriented systems, SLAs, and crowdsourcing. Also, in this chapter, we define the focus of the dissertation, and review the related research done in this field.

2. The main contribution which describes the proposed approaches and their evaluation in detail. The part is split into four chapters:

    - Chapter 3 proposes the adaptive request prioritization approach that enhances a business process engine to support SLA-driven execution of processes that comprise "low-throughput" services.
    - Chapter 4 introduces *scheduled crowdsourcing* concept and investigates SLA negotiation and fulfillment techniques for crowdsourcing.
    - Chapter 5 presents an incentive-based SLA-aware method for executing business processes on top of an internal crowdsourcing platform.
    - Chapter 6 presents *Providence*, a framework for negotiating and ensuring data privacy SLA terms for large-scale service-oriented systems.

3. The conclusion part presented in Chapter 7 summarizes the achieved results, discusses the practical applications of the work, and provides an outlook for further possible research directions.

CHAPTER 2

State of the Art

This chapter provides an overall state of the art overview in areas related to the thesis. We show which parts and aspects of those areas are covered by this work and how is it positioned in the domain. Although the thesis addresses a narrow set of problems, it touches upon manifold specific fields. Therefore, more focused discussion of related research for each contribution is given in *Related Work* section of corresponding chapters.

2.1 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is a system design and integration approach that aims to improve flexibility, governance, and change management of software systems, and also to provide a better perspective on the implemented processes from the business point of view. SOA establishes a solid high-level outlook on enterprise information systems while hiding not relevant low-level details. It simplifies the implementation of high-level business logic and helps to unify process representation. This consequently makes it easier to map actual process implementation to business models, e.g., using Business Process Modelling Notation (BPMN) [58], and thus allows to lesser the gap in understanding between IT and management representatives. As a result, it helps to better understand the processes flowing in a company, and improve them more effectively.

The main principles of SOA are loose coupling, reusability, and composability [19], that mainly become enabled by the underlying technology. Web services are the state-of-art technology for SOA [61]. Created and supported by W3C standardization consortium [84], they provide the platform independency for SOA implementations. The main elements of web services' technological stack are Web Service Description Language (WSDL) [45] which enables to specify standardized service contracts, Simple Object Access Protocol (SOAP) [69] which standardizes the information exchange with web services, Business Process Execution Language (BPEL) [43] which allows to design and execute business processes on top of web services, and eXtendable Markup Language (XML) [44] which is used by all the other standards. Enterprise Service Bus

(ESB) delivers centralized control over message routing, mediation, adaptation, and transformation, as well as for governance and security of the service system. The complementary standards like WS-Coordination, WS-Security, WS-Policy, and others enhance the the basic specifications and provide the additional capabilities for transactions, security, and policy compliance.

Although SOA is being successfully deployed in various industries [30], there is a huge potential for improvement in the field. Most important research challenges in Service-Oriented Systems among others are dynamically reconfigurable runtime architectures, semantically enhanced service discovery, QoS-aware service compositions, dynamic and adaptive processes, service governance, Self-* properties (configuring, adapting, healing, optimizing, protecting), service versioning and adaptivity [62].

This thesis touches upon a number of those challenges. Adaptive request prioritization and incentive-based process execution deal with dynamic processes and use runtime QoS-aware adaptation and self-adapting management techniques in the context of human-provided services. Scheduled crowdsourcing enables crowdsourced services to be included into QoS-aware service compositions. The Providence framework enhances service governance in the context of handling of confidential and privacy-sensitive data.

2.2 Service Level Agreements

A Service Level Agreement provides means to formalize the guarantees regarding non-functional properties of a service for the consumer. An SLA normally has a form of a contract and is composed of a number of general terms and conditions, e.g., payment for the service, as well as Service Level Objectives (SLOs) that characterize the given guarantees. SLO is usually related to a specific metric and formally describes a way to measure it, a guaranteed value, and a penalty incurred in case of non-fulfillment. For example, an SLO for *availability* can be measured as number of successfully served requests divided by a total number of requests within a month, claim the guaranteed value of 0.999, and, if the eventually measured value turns out to be lower, waive the monthly payment for the service. Penalty functions can be applied not only to particular SLO, but to a group of SLOs or to the whole agreement. They are generally not restricted in their form, so they can include several metrics and implement complex logic.

2.2.1 Quality of Service

The relevant metrics for provision of services are usually referred to as Quality of Service (QoS) metrics, and describe performance, security, and transactional concerns. Table 2.1 lists the most featured QoS metrics in literature [23, 59, 71].

In context of the problems addressed by the thesis, we focus on a fraction of these metrics which is relevant to human-provided and crowdsourced services. In such a mixed system which relies on software components and infrastructure, as well as on human labour, some metrics are solely dependent on the former, while others can be strongly affected by the latter:

- Metrics like traceability, auditability, and scalability are suitable for auditing the service portfolio of a company. However, they are either not relevant for establishing SLAs, or should be represented by more specific metrics like *capacity*.

Table 2.1: Quality of Service metrics

Name	Description
Performance	
Scalability	The ability of a service to process more requests in a given period.
Robustness/flexibility	The degree to which a service handles correctly invalid, incomplete or conflicting inputs.
Availability	The probability of successful processing of a request by a service.
Reliability	The ability of a service to provide the claimed functionality under stated conditions for a specified period of time. More fine-grain metrics include Mean time between failure (MTBF), Mean Time to Failure (MTF), and Mean Time To Transition (MTTT).
Response time	How long does it take a service to process a request. Sometimes the term is used interchangeably with *latency*, however, SOA practitioners define latency as minimum time required to get any form of response, therefore referring to the time spent on the wire [25].
Capacity	The amount of requests that can be served by a service in parallel.
Throughput	The amount of requests that can be served over a specified period of time.
Accuracy	The error rate produced by the service.
Transactions	
Integrity	The ability of a service to conform to transactional ACID properties: Atomicity, Consistency, Isolation and Durability.
Security	
Authentication, authorization, and encryption	Mechanisms that are used to secure the information exchange and access to the service.
Confidentiality and accountability	The guarantees given regarding the usage of data submitted to the service.
Traceability and auditability	The ability of tracking the service requests and analyze its performance.

- Robustness depends on system design, i.e., how much effort was devoted for input data consistency checking, handling of incorrect input data, and so on. Integrity is achieved by aligning the underlying components with transaction control mechanisms offered by the service. Authentication, authorization, and encryption also represent mainly technical cross-cutting concerns.

- Availability and reliability depend on both underlying technical infrastructure and human resources. A failure can occur due to technical reasons, e.g., network, software, and hardware issues. Such problems are not dependent on the human force involved in provision of service functionality. Also, a failure can be caused by poor human resource management or unsatisfactory human work that result in poor quality and timeliness of the provided service. These factors are covered by more specific low-level metrics considered below.

- Response time, capacity, throughput, and accuracy are strongly affected by underlying human resources, as usually humans are diverse in their skills and work at slower speeds than software components. Also, human resource management and particularly workforce scheduling can significantly affect these metrics. Confidentiality and accountability can be facilitated by special technologies for access control and data provenance. However, in large enterprises, there are more prone to human factors because it gets easier to overlook the inappropriate use of data when it comes to be maintained in various locations by different parties.

This work focuses on the latter group for which human work has the strongest impact, and aims to facilitate service-oriented systems with more adaptive and automated approaches for negotiating and ensuring these properties.

2.3 Service Level Management

Service level management is continiously gaining attention from both business and research communities. From the business perspective, it is considered one of the key areas of IT service management and the best practices in this area are constantly evolving [53]. The research community is directed towards more automated SLA negotiation and ensurance, and investigates approaches and frameworks that that aim to make service level management more autonomous, efficient, and agile.

2.3.1 SLA negotiation

A most prominent approach for automated SLA negotiation is a policy-based approach where the process is driven by a negotiation policy which includes high-level goals, service level objectives, constraints (like best and worst acceptable values for service level objectives), options, and preferences (such as priorities and weights) [9, 13, 16, 94]. A decision support system is then used to calculate and negotiate optimal objective values based on the policy. These models and technologies allow to customize and perform fine control of the SLA negotiation process on different levels, whereas the approaches proposed in this thesis aim to estimate feasible values and offers based on the underlying resources that can serve as a basis for SLA management.

The technical aspects of SLA negotiation include agreeing on negotiation phases, steps, and logic. Also semantic problems can arise if consumer and provider do not share a common understanding of metrics and objectives used in SLA offers. These problems are resolved by using standardized negotiation protocols and languages that aim to establish an efficient dialog

between consumer and provider [32, 42]. This thesis does not focus on technical details of negotiation and approaches the SLA negotiation issues on the higher level.

2.3.2 SLA ensurance and violation prevention

In the first place, SLA fulfillment assumes that there are enough resources to handle the current and anticipated demand. The problems in this area are studied by capacity management and planning [89].

However, as it was discussed in Chapter 1, a mixed service system cannot be fully predictable due to human factors, flexible demand, or unexpected events. To overcome these issues, a system has to spot and escalate such issues in an adaptive way. Various adaptive mechanisms can be used to keep a Service-Oriented System compliant to SLAs, such as modification or re-composition of business process, replacement of services, or re-configuration of service platforms and computational resources (i.e., load balancing) [91]. Prevention of violations can be enhanced by collecting the knowledge about the impact of possible actions on the service composition, e.g., by using machine learning [46]. These approaches can only be applied when the appropriate mechanisms for adaptation are well-defined and integrated into the system's architecture. However, in the context of human-provided services, such adaptation mechanisms were not investigated, and the existing ones do not provide comprehensive means to address the problems described in Section 1.1. This thesis aims to bridge this gap and to identify such mechanisms, as well as propose and evaluate algorithms and architectures that can effectively use them.

2.4 Human-provided Services

Major industry players have been working towards standardized protocols and languages for interfacing with people in SOA. Specifications such as WS-HumanTask [24] and BPEL4People [4] address the lack of human interactions in service-oriented businesses [49].

These specifications allow to put additional management information into process description, including task stakeholders, employee and task groups, task states and priorities, time-outs and escalation rules, etc. For a business process engine, such a task looks like a regular web service and should be handled the same way. However, the tools based on these specifications allow advanced activities for the people assigned to tasks, such as forwarding, delegating, suspending and resuming, and also provide coordination mechanisms for managers.

Such standards are undoubtedly useful and provide solid foundations for creating more flexible service-oriented systems with human interactions. However, they only provide technical means but do not offer solutions for SLA provisioning in such systems, and therefore are complementary to the methods and algorithms discussed in this work.

2.5 Crowdsourcing

Crowdsourcing is *"an act of taking a task traditionally performed by a designated agent (such as an employee or a contractor) and outsourcing it by making an open call to an undefined but large*

group of people" [31]. The application area of crowdsourcing is very broad, and includes use and application of collective intelligence, mass creative works, filtering and organizing of vast information stores, use of the crowd's collective pocketbook (crowdfunding), and so on. Recent efforts demonstrate the successful adoption of crowdsourcing at an ever-increasing rate, and the amounts of both platforms and workers in such platforms are expected to grow rapidly [18].

Crowdsourcing can take different forms. Among others, it can be contest-based (e.g., when workers propose solutions for the problem, and only one of them is chosen and rewarded) or implement "crowd wisdom" approaches (when an undefined group of people collaborates to solve a problem). This thesis features a specific type of crowdsourcing which is used to process independent self-contained tasks with crowd workforce in exchange for rewards. Two types of platforms can be distinguished in this context:

- *External crowdsourcing* provides the ability to outsource human labour to a third party which acts as a broker and accumulates the workforce for performing the outsourced tasks. A crowdsourcing provider establishes contracts with workers and defines the internal policies and mechanisms for task assignment. For the consumer, the process of having a task done is therefore simplified to the form of software service invocation, thus releasing him/her from workforce contractual and maintenance issues. Moreover, such a scheme allows consumers to satisfy flexible demand as they can submit arbitrary amount of tasks to the platform. Such an approach is good for outsourcing tasks with minor effort that, however, require human capabilities (e.g., transcription, classification, or categorization tasks [35]). Examples of such systems are Amazon Mechanical Turk [82], Crowd-Flower [17] and ClowdCrowd [15].

- *Internal crowdsouring* has the similar architecture at its core, but the platform is deployed internally and is accessed only by company's employees or external contractors. It allows a company to utilize the skills of already employed people in a more flexible way by enabling workers to choose tasks they wish to perform, instead of assigning them directly. On the one hand, the participants can gain more bonuses by processing additional tasks, on the other hand, it helps the company to discover new expertise and skills of their employees. It also allows to establish straightforward rewarding mechanisms and therefore directly motivating people to produce more and better results. As the membership of the platform is limited by internal workforce, it reduces security issues and allows to crowdsource more complex, critical, and domain-specific tasks because the workers are motivated to guard their reputation and are experienced in the domain.

Although crowdsourcing provides indisputable benefits, there are several issues that prevent it from being extensively used in everyday business processes [38]. The flexibility offered by a platform makes it at the same time a complicated task to predict and control the timeliness and, for external crowdsourcing, also the quality of the received results. Currently, these platforms maintain a market-like architecture where workers register at choose the tasks they like to process, and there is no active influence upon assignments. As a result, the platform is unable to give any certain guarantees regarding the time of processing a task or the outcome quality a customer can expect. The work in this thesis features both these types of crowdsourcing, and

presents approaches to enhance such platforms with additional capabilities for negotiating and fulfilling the processing time and quality guarantees.

2.6 Workforce Optimization

Workforce optimization focuses on improving operational efficiency and managing the workforce effectively. A huge amount of research has been done in this field with focus on different industries [21, 28]. The addressed problems include demand modeling, satisfaction of the constraints arising from workplace regulations, line of work construction, shift scheduling, days off scheduling, consideration of social and psychological effects impact, and so on.

However, the considered models and algorithms require significant modification when they are to be transferred to a different application area [20], and there is no clear way of applying the results to SOA and crowdsourcing environments. Nevertheless, the research in this area can be used to improve and complement the methods we propose in this work, e.g., by refining proposed scheduling algorithms with consideration of employee satisfaction factors. Another example would be the usage of the prioritization approach presented in Chapter 3 on top of a staff scheduling framework, so that the staff scheduling algorithms are used to meet the overall demand and fulfill the regulations, while request prioritization is used to perform SLAs-based fine-tuning of the service-oriented system.

CHAPTER 3

Adaptive Prioritization of Requests in Orchestration Engines

The availability of scarce resources in a service-oriented system demands for context-aware selection policies that adapt based on service level agreements. One of the open issues is to prioritize service requests in dynamically changing environments where concurrent instances of processes may compete for resources. Here we propose a runtime monitoring approach to observe the actual state of the system. We argue that priorities should be assigned to requests based on potential violations of SLA objectives. While most existing work in the area of quality of service monitoring and SLA modeling focuses typically on purely technical systems, we consider service-oriented systems spanning both software-based services and human actors.

The approach presented in this chapter helps to cope with these challenges by prioritizing service requests that may cause violations of service level objectives such as *response time* and *capacity* that are associated with processes.

3.1 Overview

Service-oriented systems have become an important approach and technological framework to solve problems in distributed computing environments. Challenges in distributed service-oriented systems include the discovery of resources and monitoring of the system's runtime behavior. Capturing the current state of the system is essential in dynamic environments where services are discovered and invoked at runtime. Research in the area of quality of service (QoS) modeling and monitoring (e.g., see [56]) has provided an important building block to observe the runtime state of a service-oriented system. Keeping services compliant to SLAs is crucial in a service-oriented system. Usually, if the system is designed properly and acts as expected (e.g., response time and service availability), the SLA is satisfied. However, both internal and external factors can compromise the overall performance of the system. While the strategic actions should be taken to prevent the system from entering undesirable conditions (e.g., through

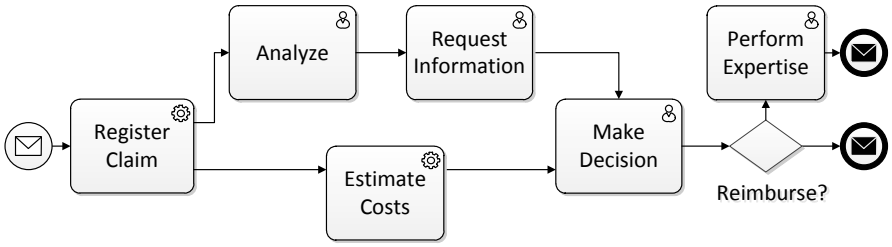


Figure 3.1a: Comprehensive insurance claim process

replicating the components, adding resources), the run-time adaptation can also be performed in attempt to minimize the penalties in given situations. This can be especially important when multiple processes need to access shared resources in a singleton manner. Assume a process consisting of multiple activities, some of them enacted by invoking software services and certain activities performed by human actors. In a service-oriented system, such a scenario could be realized by modeling and enacting compositions using the Business Process Execution Language (BPEL) [85], where human steps are modeled using BPEL4People and WS-HumanTask [49,74]. Service provided by human actors can be regarded as 'low throughput' services because humans naturally work at lower speeds than software-based services. If human-based low throughput services are part of a process, the order in which processes get the response from such services impacts the process execution times. If the system gets overloaded or some processes are late due to unexpected delays, the SLAs might be violated. However, the control over the ordering of service requests can reduce the penalties or prevent the violations at all. A specific example of such a situation is described in Section 3.2.

To address these challenges, we propose a dynamic adaptation approach and heuristic prioritizing algorithm that analyzes the current state of the service-oriented system at run-time and prioritizes service requests according to SLAs bound to processes in the system. We assume that the execution state of all the processes in a service-oriented system is accessible, and that the penalty functions of SLAs are provided. The main idea behind the algorithm is that the priority of a service execution is given to those processes that are expected to produce the highest penalties if this service is delayed for them. To illustrate our approach, we discuss insurance claim processes.

3.2 Scenario

To illustrate our approach, we discuss a motivating scenario where processes are designed and executed in the context of insurance claim handling. We look at different kinds of insurance processes: the first one dealing with a *comprehensive* (Figure 3.1a) insurance plan and the second with *liability* (Figure 3.1b) coverage.

The comprehensive plan ensures that damage (for example accidents or vandalism) is being paid by the insurance company. In certain European countries, liability is the minimum insurance coverage everyone must have due to government regulations. As an example, if A is responsible for the damage of B, then A's insurance company must pay for B's damage. Fig-

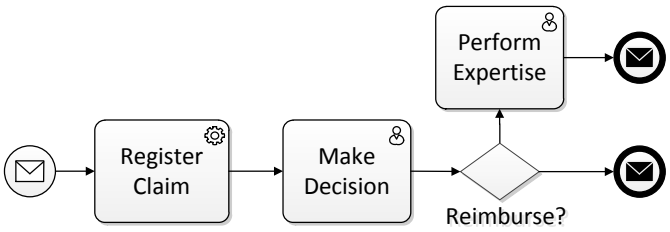


Figure 3.1b: Liability insurance claim process

ure 3.1a shows the process for the comprehensive insurance plan. People obtaining coverage through this plan may be regular or premium customers. For premium customers, the insurance company wants to provide better quality of service as for regular customers. For example, faster processing of the insurance claim. The process is initiated as soon as the customer issues an insurance claim. The registration of the claim is performed automatically by a software service. In the next step the process splits into two parallel branches. Based on the issued claim, a software service is invoked (step *Estimate Cost*) to perform an automatic calculation of the expected costs. A person from the insurance company analyzes the received claim and typically requests further information from the customer. After both branches have finished, a decision is made by a supervisor. The outcome may be to reimburse the customer or not. In the first case, an expert reviews the case by visiting the customer to obtain precise understanding of the damage upon which actual calculations are made. The alternate case terminates by sending a (auto-)generated notification to the customer.

The second process example is shown in Figure 3.1b. In contrast to the comprehensive insurance example in Figure 3.1a, we assume in this scenario that the person filing the claim is *not* a customer of the insurance company. The process is therefore simpler because the person filing the insurance claim only receives limited support (e.g., help desk) and also limited service-level guarantees are given. The process is initiated in the same manner as in the comprehensive insurance plan example. Afterwards, a decision is made based on received information. The next steps are again equivalent to the steps (*Perform Expertise* and notification) of the first process.

What these processes have in common is that they access shared resources. For example, by invoking a service in the step *Perform Expertise*. If a process invokes this service, other processes (instances) may need to wait until free resource capacities are available. However, this could cause violations in SLA objectives. Thus, careful scheduling of requests is needed to satisfy customer needs.

3.3 Adaptation Model

In this section we describe the conceptual architecture of our solution, the prioritization algorithm, and the remarks regarding the deployment of such a solution in a real SOA. We assume that all services in the system are atomic (not composite), as we can decompose all such services.

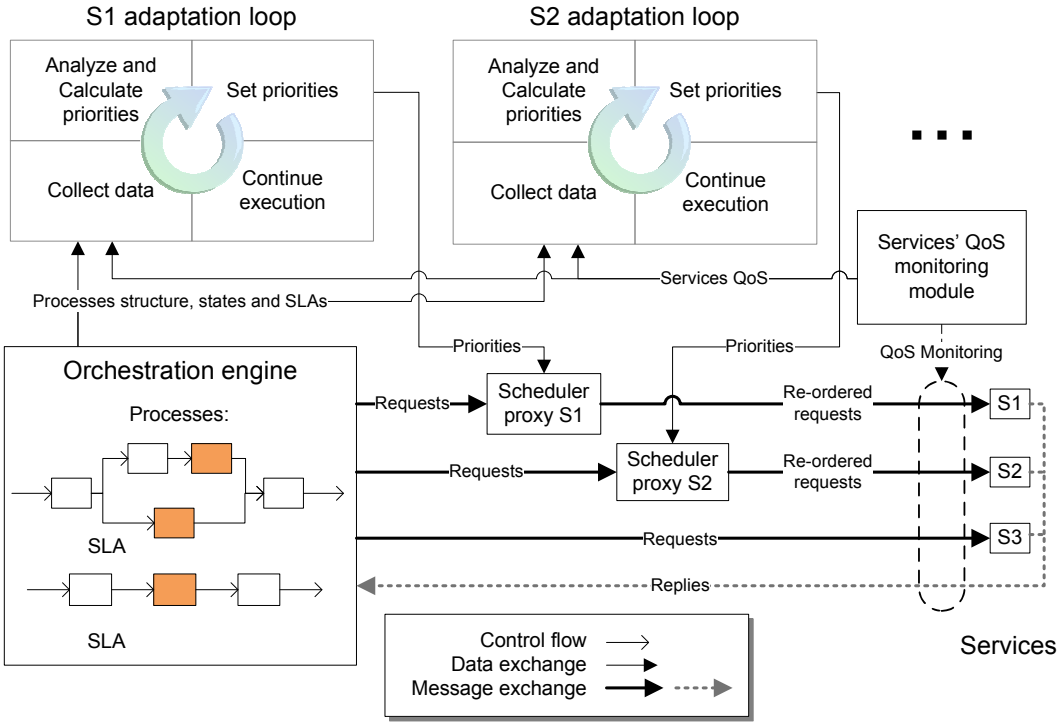


Figure 3.2: The overall architecture of the approach

3.3.1 Conceptual Architecture

The architecture of the approach is depicted by Figure 3.2. Normally, when a process invokes a service, a request message is sent to the service endpoint, so the order in which requests are processed by a service is determined by this service's implementation which is unaware of processes running, SLAs, or other context information. In our approach, a scheduler proxy is created and assigned to each service whose request priorities are being adapted. The scheduler proxy intercepts requests to the service and reorders them according to prior defined priorities. The scheduler is aware of the service's QoS (through the monitoring module), so it dispatches requests towards the service depending on the available free capacities. It ensures that the priorities are obeyed. The priorities are periodically updated by an adaptation component associated with the service. This component implements the common MAPE (Monitor, Analyze, Plan, Execute) loop logic. As the re-ordering is performed before the requests are sent to a service, the actual location of the service does not play a role, i.e., it can be both externally or internally provided service.

The adaptation loop consists of three phases:

1. *Collection of context and monitoring information.* The structure, the current execution state, and penalty functions (from SLAs) of all currently running processes as well as the QoS information are collected from the orchestration engine and from the monitoring module respectively. We use a deterministic QoS model, so the capacity and response

```
   Input  : Service S, its response time S_RT
   Input  : Set of processes P, for each process p penalty function L_p(t)
   Output: Ranked requests
1  for process p in P do
2      R_p = pending requests of S in p
3      R_e = requests of S predicted to be made during S_RT/2 period in p
4      R = R_p ∪ R_e
5      Assume that replies of all requests in R are received after S_RT, predict time t of
       finish for process p
6      l_0 = L_p(t) ; // Default penalty
7      for request r in R do
8          Assume that a reply of r is received after S_RT * 2 and replies of all other requests
           in R are received after S_RT, predict time t_r of finish for p
9          l_r = L_p(t_r) ; // Penalty for current request
10         d_r = l_r − l_0 ; // Difference between default penalty and the
           penalty for current request
11         Add the tuple of r, d_r, and request time k (either real or predicted) to list D
12     end
13 end
14 Sort D descending by d_r then ascending by k
15 Return D
```

Listing 3.1: Prioritization algorithm.

time are considered single values. We do not focus on particular approaches for QoS monitoring which represents an extensively studied distinct research field (see, e.g., [56, 92]).

2. *Calculation of request priorities.* The collected data is passed to the algorithm (See Listing 3.1) which calculates priorities for forthcoming and recently made requests to the service.

3. *Scheduler update.* The corresponding scheduler is updated with the calculated priorities, so the requests to the service can be ordered accordingly.

Iterations in the adaptation thread are performed with the period of the half of the service's response time. This value equals to prediction period (Listing 3.1, line 3). The value was selected empirically. As it was evaluated in experiments, if the period was greater, the algorithm performed poorer as sometimes service capacity was unused too long while waiting for predicted requests, however, the lesser period did not improve the performance of the algorithm.

3.3.2 Deployment in a Service-oriented System

Although not the main focus of this work, we give a short analysis of the mapping and deployment of our conceptual architecture in real SOA environments. We assume that there is single and accessible (in-house) orchestration engine. Our approach would also work with multiple

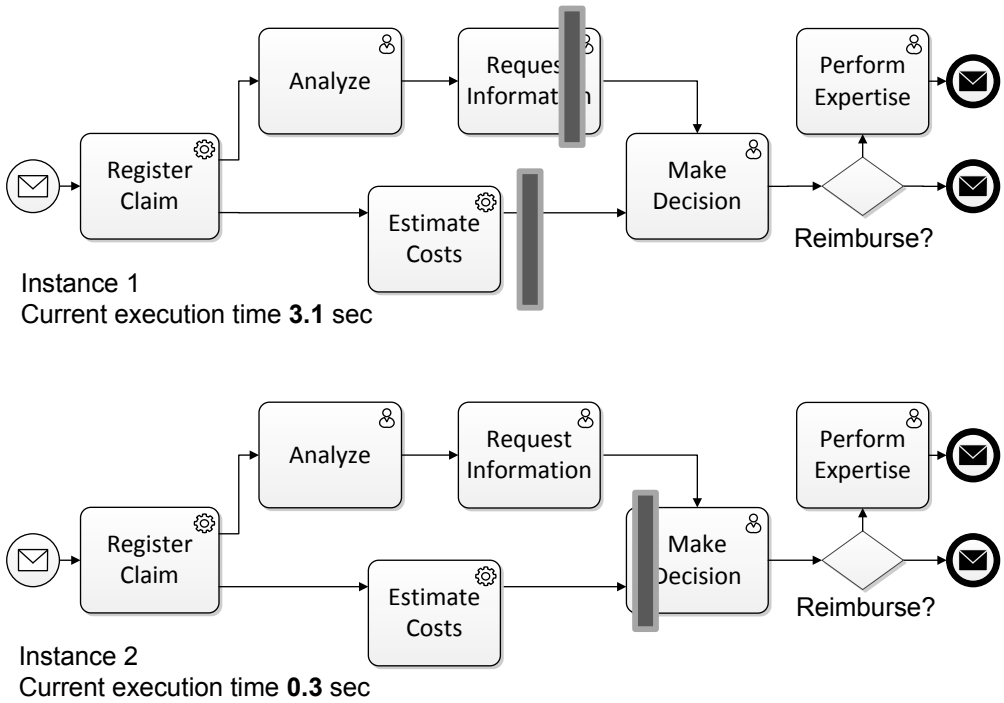


Figure 3.3: An example of process states

deployed orchestration engines in the environment. However, for simplicity of discussions, we assume only a single engine. In order to enable the deployment of the proposed architecture, the orchestration engine should be extended to supply the adaptation loops with process state information. Many SOAs have moved towards a bus-oriented messaging backbone. An enterprise service bus (ESB) should be configured to support scheduler proxies. We expect these extensions to be implemented as plugins for corresponding SOA components, however, such an implementation fully depends on the underlying technologies and software being used.

3.3.3 Prioritization Algorithm

The prioritization algorithm is outlined in Listing 3.1. The algorithm predicts forthcoming calls of the service and prioritizes the corresponding requests together with the pending requests according to the penalty difference which appears if the receiving of request's reply is delayed. The algorithm uses predictions which are performed straightforwardly, adding together response times of the services to be called according to the process structure. As for flows and conditions, the time of the most delayed branch is selected. The algorithm covers the main types of process constructs: *sequence*, *flow* and *condition*.

```
// Process instance 1:
```
1 Instance 1 has no pending calls of DMS, however, as the IRS is expected to respond in 0.05 sec (as its response time is 0.1 sec), then the call c_1 to DMS is predicted in 0.05 Section
2 Default process finish time is calculated: $3.1 + 0.05 + 0.2 + 0.5 = 3.85$ sec
3 Default penalty is calculated: $L_S(3.85, 3, 10) = 0$
4 c_1 is assumed to respond in $0.2 * 2 = 0.4$ sec, Process finish time is calculated: $3.1 + 0.05 + 0.4 + 0.5 = 4.05$ sec
5 The penalty for delayed c_1 is calculated: $L_S(4.05, 3, 10) = 10$
6 The penalty difference for c_1 is calculated: $d_{c_1} = 10 - 0 = 10$
7 $< c_1, 10, 0.05 >$ is added to D
```
//
// Process instance 2:
```
8 Instance 2 has a pending DMS call c_2. No other DMS calls are predicted.
9 Default process finish time is calculated: $0.3 + 0.05 + 0.2 + 0.5 = 1.05$ sec
10 Default penalty is calculated: $L_S(1.05, 3, 10) = 0$
11 c_2 is assumed to respond in $0.2 * 2 = 0.4$ sec, Process finish time is calculated: $0.3 + 0.05 + 0.4 + 0.5 = 1.25$ sec
12 The penalty for delayed c_2 is calculated: $L_S(1.25, 3, 10) = 0$
13 The penalty difference for c_2 is calculated: $d_{c_2} = 0 - 0 = 0$
14 $< c_2, 0, -0.01 >$ is added to D

Listing 3.2: Algorithm steps for the example.

3.3.4 Illustrative Example

To illustrate the work of the algorithm, the algorithm steps for two instances of comprehensive insurance claim scenario process are described. Let the services have the same QoS as in experimental setting (See Section 3.4) and let both process instances have SLA penalty functions L_S(t,3,10). Let the processes have the states as depicted in Figure 3.3. Instance 1 was delayed for some reason. The Information request service (IRS) was called 0.05 sec ago there, so the process is waiting for its response; the Cost estimation service has already returned the response. In instance 2 the Decision making service (DMS) was called 0.01 sec ago. Given that DMS's request priorities are being adapted, the analysis step of its adaptation loop's next iteration would perform as shown in Listing 3.2. Finally, when D is sorted, the priority of c_1 is considered higher than the priority of c_2.

3.4 Experiments and Discussion

We implemented an orchestration engine simulator which mimics the QoS characteristics of services and the execution of processes. It simulates the temporal behavior of the system and supports main basic process elements: *sequence*, *flow*, and *condition* (executes with given probability). To demonstrate the advantages of our approach, we simulated unexpected overloads

and delays in a service-oriented system under various circumstances. We scaled the realistic response times of the services for simulation from days to seconds. So the half of a simulated second corresponds to half of a day in real setting.

3.4.1 Setup

In our setup, several process types are repeatedly instantiated in the system according to the frequency $F(t)$, as shown in Figure 3.4.

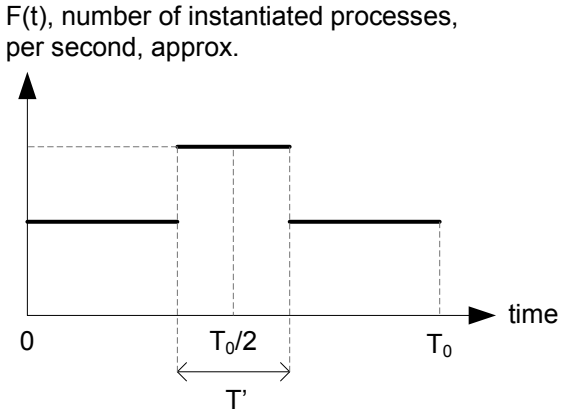


Figure 3.4: Experiment model

The type of instantiated process is chosen randomly (all types are considered equiprobable). The approximate number of instantiated processes per second is increased from F_0 to F' for a period T' in the middle of the overall process instantiation timespan T_0. The unexpected additional load is thus simulated. The inaccuracy of response time is simulated as well: the actual response time of a service is calculated as $RT + RT * k * R$ where RT - expected response time, k - inaccuracy factor, R - normally distributed random value.

We apply this system behavior for 6 series of experiments (E1-E6) based on the motivating scenario (Section 3.2). The experiments are described in Table 3.2. The processes types $T1$ and $T2$ correspond to comprehensive insurance claim and motor vehicle liability insurance claim processes. The QoS values used for services simulation are presented in Table 3.1 (the set of services maps to the steps of motivating scenario processes). All experiments use response time inaccuracy factor of *0.3*. The conditions in both processes are assumed to be true in 70% of cases. In E5 and E6, the analysis service happens to be delayed by 0.5 sec in 10% of cases.

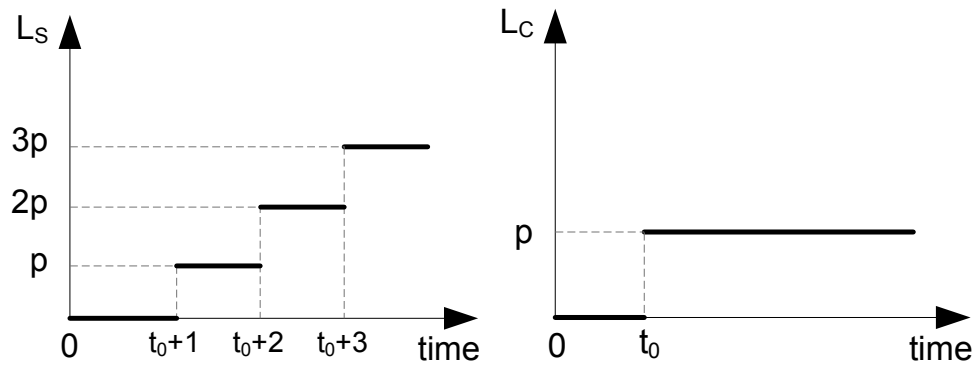
In our simulation, capacity indicates the number of simultaneous requests that can be served by a service. As penalty functions, we used staged L_S and constant L_C functions (see Figure 3.5).

Table 3.1: Values of service quality metrics in experiments

Name	**Response time** [sec]	**Capacity**
Analysis service	0.15	5
Expertise service	0.50	5
Decision making service	0.20	5
Information request service	0.10	10
Estimation service	0.10	100
Registration service	0.01	100

$$L_S(t, t_0, p) = \begin{cases} 0 & \text{if } t < t_0 \\ (trunc(t) - t_0) * p & \text{if } t >= t_0 \end{cases} \quad (3.1)$$

$$L_C(t, t_0, p) = \begin{cases} 0 & \text{if } t < t_0 \\ p & \text{if } t >= t_0 \end{cases} \quad (3.2)$$



(a) Staged. (b) Constant.

Figure 3.5: Penalty functions

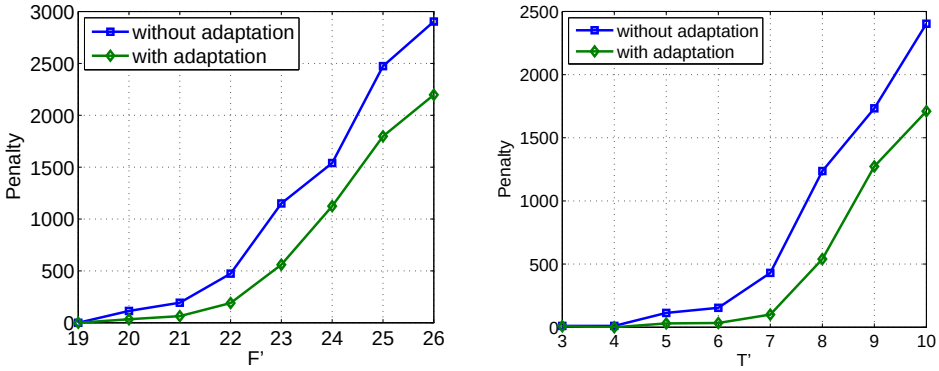
Each experiment was performed 2 times: first time with no adaptation with requests served in First-In-First-Out manner, and the second time with the adaptation enabled. Penalties were measured for each process. As the simulation involves various random factors (process instantiation, process type selection, error and unexpected delay injection, conditions), we made sure that such experiments get the same values returned by random generators. The results of experiments are depicted in figures 3.6-3.8.

3.4.2 Discussion

All experiments demonstrate a considerable reduction of penalties of 30-80%. In the following we show pairs of figures: the left figure showing SLA penalties by varying **F'** and the right figure by varying **T'**.

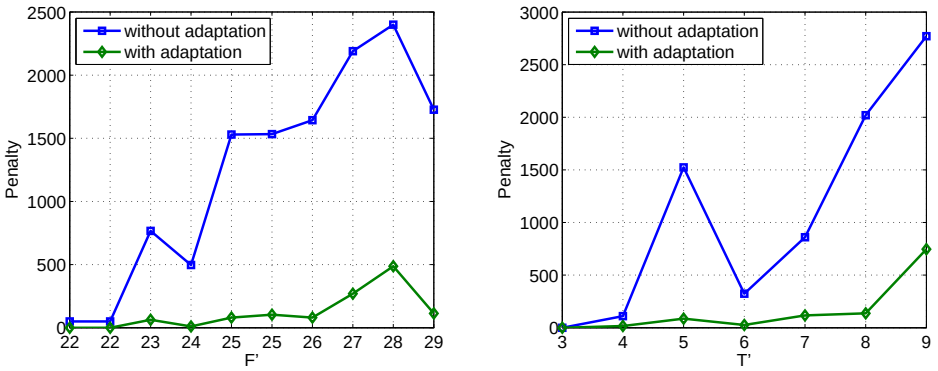
Table 3.2: Experiments performed

Name	Process types: penalty functions	T'	F'
E1	T1 : L_S(t,3,10) only	5	19 - 26
E2		3 - 10	20
E3	T1 : L_S(t,3,10), T2 : L_C(t,8,20)	5	22 - 29
E4		3 - 9	25
E5	T1 : L_S(t,3,10), T1 : L_S(t,3,15),	6	19 - 26
E6	T1 : L_S(t,3,20), T2 : L_C(t,8,20)	3 - 9	22



(a) E1 (b) E2

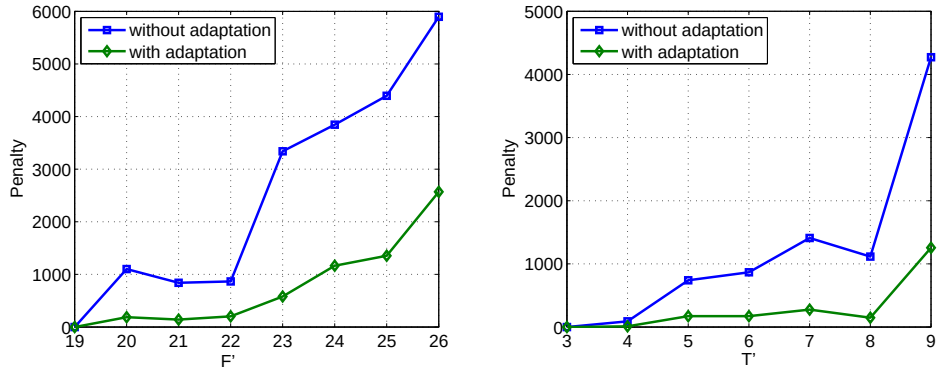
Figure 3.6: One process type, without delays



(a) E3 (b) E4

Figure 3.7: Two process types, without delays

In experiments E1 and E2 (see Figure 3.6) the absolute difference between penalties is relatively constant which is explained by the similarity of executed processes: only one process



(a) E5 (b) E6

Figure 3.8: Four process types, with delays

type is instantiated, no difference among instances in form of service delays, the only difference is the variety of response times resulted by the inaccuracy factor. Thus, these experiments give very limited freedom for re-prioritization. Still, the adaptation reduces penalty considerably.

In experiments E3-E6 (see Figure 3.7 and Figure 3.8) the reduction is greater than in E1-E2 because of the possibility to postpone the service calls in T2 processes at no expense ($t_0 = 8$ for L_C).

This is revealed mostly in E3 and E4 as approximately half of the processes were of type T2. In E5 and E6 the reduction is lesser than in E3 and E4, because only quarter of processes were of type T2. In contrast to E1-E2, the absolute difference between penalties in E3-E6 grows with the load increment, as the process pool contains various types of processes which causes the dissimilarity of re-prioritization impact, and, thus, increases the algorithm's efficiency. The non-monotonicity of penalty growth in E3-E6 is caused by the random factors in process generation and instantiation mechanism. To summarize these observations, the performed experiments clearly show the advantage of using adaptation for prioritizing requests in case of unexpected overload or response delays.

Of course, the are limits for applying the adaptation. These limits are reached when the time needed to perform an analysis iteration of the orchestration engine state becomes comparable with the response times of the services whose request priorities are being adapted. For example, in the experiment with large F' shown in Figure 3.9, the method becomes inefficient on $F' > 190$ (the maximal size of the process pool is about 600 processes). However, this limit would scale together with response times of the services, and will be hard to reach in a real setting.

3.5 Related Work

Our approach is aiming at minimizing SLA penalties via prioritizing the requests and assigning the available service capacity. Such an objective constitutes a scheduling problem. Among the variants of this problem, the resource-constrained multi-project scheduling problem (RCMPSP)

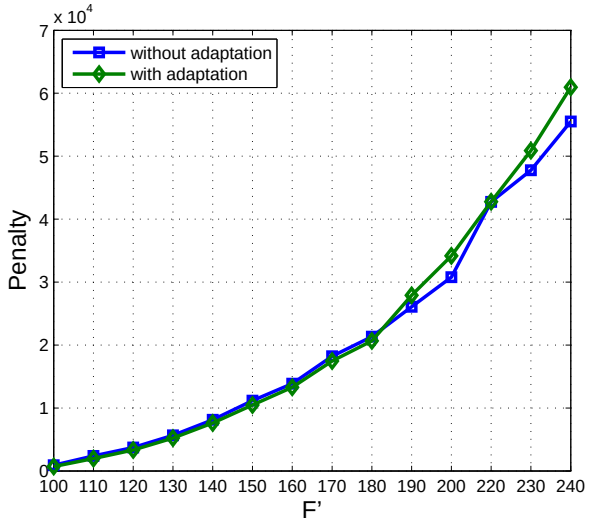


Figure 3.9: Large values

[29] is the most conformable to ours. However, those studies do not address the service-oriented architecture, and, thereby, related concepts such as SLA or QoS.

A priority scheduling method for process engines is proposed in [81]. It analyzes the execution status in the process engine and dynamically assigns the priorities to service requests alike to our approach. Instead of penalty functions, it considers utility functions. However, this work assumes that services support prioritized execution of requests with either *High* or *Low* priority. Such an assumption has two strong disadvantages: firstly, it severely reduces the scope of application, as services do not support prioritized execution in general, and, secondly, even if a service distinguishes between requests with high and low priorities, it would still not be able to distinguish between requests with the same priority which is crucial in case of multiple concurrent requests and low service throughput. Unlike it, our approach uses essentially different prioritization algorithm and request scheduling (via proxy schedulers), so it does not have these disadvantages.

SLA violation and prevention in service compositions through adaptation is addressed by various researchers. For example, [47] proposes a general adaptation framework for monitoring and preventing SLA violations by performing various actions upon the service composition, like changing the service bindings or composition structure. In contrast to it, our approach does not address the composition changes, but request prioritization among different compositions.

Various escalation mechanisms to avoid breaking the workflow deadlines are discussed in [83]. The prioritization of tasks or cases which is highly related by implication, is briefly discussed. However, the paper does not consider SLAs and penalties, and no rationale regarding the actual implementation of the method is given.

Trade-offs between costs and profits of various service composition adaptations are discussed in [48]. The adaptation proposed by us does not imply any costs besides the performance overhead used for the analysis.

The approaches like [22, 64] use dynamic binding to improve the QoS of process instances, whereas our approach does not assume the existence of several service endpoints.

3.6 Summary

The problem of reducing and preventing SLA penalties in the context of unexpected system overload or service response delays is considered in this chapter. The architecture for request scheduling in service-oriented systems and the request prioritization algorithm are proposed. A realistic motivating scenario was taken as a basis for evaluation. The proposed solution was evaluated for the scenario implemented in an orchestration engine simulator. The results of evaluation demonstrate the considerable (30-80%) penalty reduction, thus, showing the clear advantage of the approach.

Generally, this approach has no special requirements for SOA system, so it has no obstacles to be applied in practice. It can be extended to allow different services to share the resources, so, for example, if one human is assigned to perform different tasks represented by different services, the system will be aware that the call of one service would impact the QoS of another.

CHAPTER 4

SLA-aware Scheduled Crowdsourcing

External crowdsourcing has emerged as a new paradigm for outsourcing simple for humans yet hard to automate tasks to an undefined network of people, providing thus scaleability and flexibility for customers. However, crowdsourcing platforms do not provide guarantees for their services, such as expected quality of the result or the time of processing. Such guarantees are advantageous from the perspective of Business Process Management, as they can strengthen the predictability in process planning and design. In this chapter, we present an alternative crowdsourcing platform architecture, where the workers are assigned to tasks by the platform according to their availability and skills. We provide scheduling and prediction algorithms that allow to provide and fulfill SLAs for the consumers of a crowdsourcing platform. The proposed architecture and algorithms address specific to crowdsourcing problems, such as lack of full control of the workers and their membership, their limited predictable availability, and a tendency of workers to over-/underestimate their skills. We evaluate the approach in a simulated crowdsouring environment.

4.1 Overview

Enterprises seek to automate all sorts of their front and back office operations to cut costs and eliminate human factors. However, humans inevitably remain as key elements of many business processes. It is not only creative, management, and communication activities that are hard to reproduce with technology. Other tasks that require basic human skills, such as image recognition or categorization, as well as specific skills, such as text translation or usability testing, are also difficult to fulfill with software only.

Crowdsourcing allows companies to outsource such tasks to an undefined network of people using an on-line platform maintained by the other party. It offers great scaleability, as consumers do not need to dedicate any internal resources and can submit arbitrary amount of tasks to the platform, therefore providing human intelligence capabilities on-demand. Such platforms usually have a market-like operation chain where the tasks received from customers are announced

at the portal and the workers choose among the assorted mass of tasks those they like to process. Examples of such systems include Amazon Mechanical Turk [82], Crowd-Flower [17] and ClowdCrowd [15].

Although the aforementioned systems have achieved certain success, we argue that purely market-like architecture lacks some features that could realize more potential of crowdsourcing platforms. As in a market-like system the job assignments are initiated by the workers themselves, it is hardly possible for the system to have an active influence upon assignments. As a result, the platform is unable to give any certain guarantees for the consumers, neither about the time of processing a task nor about the outcome quality they can expect [38]. From a Business Process Management (BPM) perspective, such guarantees can provide an additional value for crowdsourcing services. First, it can strengthen the certainty and predictability in process planning and design. Second, if such guarantees are given in form of SLAs, it allows to use crowdsourcing in QoS-sensitive business processes [1, 2, 26].

To address these issues, the platform architecture can be altered so that the workers are assigned to tasks *by the platform*. In this case workers register at the platform, provide information about their skills and availability, and expect the platform to allocate tasks to them accordingly. The platform thus schedules submitted tasks based on these and other factors such as skill requirements provided by consumers, service level agreements with consumers, and monetary rewards. We refer to this model as *scheduled crowdsourcing*. While providing the same flexibility for consumers, scheduled crowdsourcing comprises a number of advantages:

- **Quality.** Skills of the workers are manifold. The tasks submitted to the platform are also diverse in their requirements. One can assume that the more the worker is suitable for a task, the better the expected outcome quality is. We refer to this indicator as *suitability*. Hence, by considering the worker-task suitability, it is possible to improve the overall results by assigning tasks to most suitable workers.

- **Deadlines.** In market-like platforms task completion times span from several to thousands of hours [35]. As in scheduled crowdsourcing the assignments are controlled by the platform, tasks can be scheduled according to specified deadlines.

- **Predictions and SLAs.** Considering the short-term information about workers' availability on the one hand and tasks in progress on the other hand, the platform can predict the available workforce and, thus, estimate what can be offered or guaranteed to a consumer who wants to submit a particular task.

4.2 Scheduled Crowdsourcing

This section describes scheduled crowdsourcing architecture. Alike to market-like approach, scheduled crowdsourcing implies that a crowdsourcing platform receives tasks from consumers and distributes these tasks for execution to the crowd. However, in contrast to market-like architecture, tasks are assigned to workers by the platform, and the assignment is based on negotiated SLAs. We assume that a task comprises manifold similar jobs that can be independently assigned to individual crowd workers. When a job is done, the result is returned to the consumer,

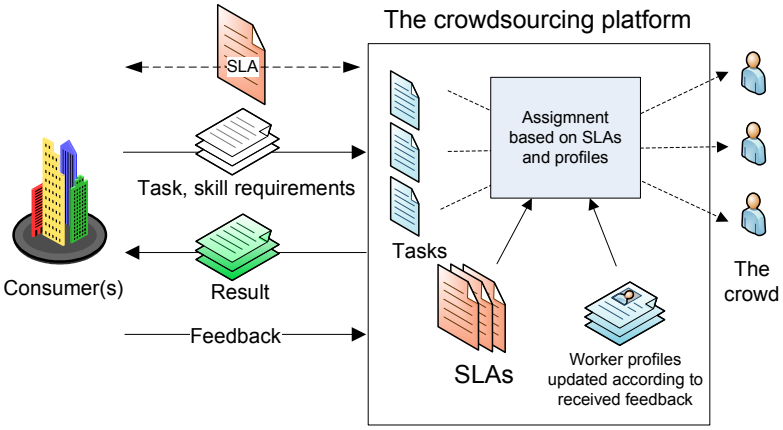


Figure 4.1: Scheduled crowdsourcing

which is invited to provide a quality feedback on this result. The overall architecture is shown in Figure 4.1.

Before a consumer submits a task, an SLA for this task is negotiated. It includes temporal, quality, and monetary objectives. The platform considers the availability of workers, and already submitted tasks to offer feasible guarantees. QoS metrics and negotiation procedure are presented in detail in Section 4.4.

Workers specify their minimal wage per time duration, and consumers specify payments for tasks. Jobs of a task are thus assigned only to workers, whose minimal wage is less or equal to the payment. Therefore, the more the customer is willing to pay, the more workers will be considered for assignment, and, as it is sensibly to assume, more suitable workers can be found. As for workers, setting the minimal wage is also a trade-off: lower values would result in more assignments, while higher values would provide the worker with higher-priced jobs, however, the assignment frequency in this case will highly depend on worker's skills due to greater competition.

Active jobs are assigned to workers according to worker-task suitability, negotiated SLAs, and short-term availability information provided by workers. If an assignment is refused by a worker despite his claim for availability, various penalty sanctions can be imposed to this worker. After all jobs of a task are done, the task is considered done. The factual QoS indicators are then compared to those specified in the SLA. If any objectives were violated, the provider might incur penalties towards the consumer. The assignment mechanism is presented in Section 4.5.

The suitability is calculated as a match between required skills for the task and the skills of a worker. Skills of workers are maintained in their profiles. Initially, skill information is provided by the workers themselves. However, the profile of a worker can be modified by the platform if the expected quality (suitability) differs from the real quality that was reported by the consumer as a feedback. The maintenance of workers' skills is performed in the platform by analyzing the feedback and trying to keep the skills in the profile aligned with the real skills of the worker. Skill maintenance is considered in detail in the next section.

4.3 Worker Skills and Job Quality

Skills of the workers are manifold. The tasks submitted to the platform are also diverse in their requirements. The platform thus needs to possess this knowledge to maximize the overall quality by assigning the most suitable workers to the task on hand. Also, the Service Level Agreements need to include quantitative measures regarding the quality of work.

We assume that for each worker-task pair a suitability measure can be calculated. Suitability is represented by a single real value in $[0, 1]$ (0 - not suitable at all, 1 - perfectly suitable) which summarizes the expectations regarding the quality of the result if the worker is assigned for a job of the task. Such notions as worker skills and skill requirements need to be formalized in a way that allows to calculate the suitability as a match between requirements and skills.

This generic approach allows to decouple the technique, which is used to calculate the suitability, from scheduling and prediction algorithms. Suitability can be thereby calculated using an arbitrary technique, which can even vary from one task to another. Therefore, the architecture is compatible with various skill and suitability assessment approaches, such as in [75] or [39].

In addition, the platform supports a feedback mechanism which allows consumers to report the quality of an assignment's outcome. If the quality of the outcome reported by a consumer differs from the calculated suitability, then the corresponding underlying characteristics of the worker should be revised, so the suitability is calculated correctly next time.

To demonstrate the approach and support the experiments, we propose a specific skill management model and a feedback processing algorithm.

4.3.1 Skills and Suitability

The model distinguishes a fixed set of skills. Each worker has a skill profile, where each skill is described by a real number $s \in [0, 1]$ that defines it quantitatively (0 - lack of skill, 1 - perfect).

Each submitted task has the required skills specified. Each required skill is also represented as a real number $r \in [0, 1]$. If $r = 0$ then the quality does not depend on this skill. If $r > 0$ then the best outcome quality is expected in case if the corresponding worker's skill s is $s >= r$. If $s < r$ then the expected quality is affected in the inverse proportion to $r - s$. The quality is again represented as a real number $q \in [0, 1]$. The suitability of a worker for a task is equal to the expected outcome quality. The exact matching formula is shown below.

Let WS_i - worker skills, RS_i - required skills of a task, $i = \overline{1, N}, N-$ number of skills. Then the suitability of the worker to the task is calculated as

$$S = 1 - \sum_{i \in M} \frac{Max((RS_i - WS_i)/RS_i, 0)}{|M|}$$

$$M : k \in M \Leftrightarrow k \in N, RS_k > 0$$

Thus, the more skills of a worker are proportionally closer to the required skills of a task, the more the worker is suitable to the task. If the worker's skill is equal or greater than the corresponding required skill, then this skill suits perfectly.

4.3.2 Feedback processing

At the beginning of her/his membership at the crowdsourcing platform a user registers with a profile representing the skills. Usually this information is not very accurate because users tend to over-/underestimate their skills. Hence at runtime, a monitoring module must run on-line and manage the profiles by updating the provided information. The task processing results and the expected quality outcome must be used as a reference for the real skills of a worker. The quality expectations on the tasks result are often detailed in the task description. At the AMT, for example, the result feedback contains usually only a task accept or reject. At our platform, with an agreement requiring the customer to give a feedback on the quality, the feedback contains crucial information for the algorithm presented in Listing 4.1 that can be used to update the skills of the reported worker profiles.

```
   Input  : QF quality feedback of the provider, QE quality expected by the provider
   Input  : worker processing worker and taskSkills required task skills
 1 workerSkills = worker.getSkills();
 2 if QE > ϑ_q then /* high quality result                                  */
       /* compare with latest history entry, update and continue on
          better QF                                                        */
 3     entry = setHistory(QF, taskSkills);
 4     for skill s ∈ workerSkills do
 5         reqSkill = getTaskSkills(s);
 6         diff = |s − reqSkill| × α_w;
 7         if s > reqSkill then
 8             workerSkills.set(s + diff);
 9         else
10             workerSkills.set(s − diff);
11         end
12     end
13     Return;
14 end
   /* low quality result                                                    */
15 wprofile = setOfProfiles.get(worker) ; /* set of registered profiles    */
16 diff = QF/QE ; /* difference between the qualities                       */
17 for skill s ∈ workerSkills do
       /* skill == 1 perfect knowledge                                     */
18     if skill × diff <= 1 then workerSkills.set(s × diff);
19 end
```

Listing 4.1: Profile monitoring.

As the scheduler requires skill knowledge, the profile update is twofold. If the worker only provided a low quality the update depends on the difference (Lines 15-19). If the quality is above a certain threshold ϑ_q and is better than a previous then we consider the required skills close to

Table 4.1: Agreement terms

Name	Description
Task characteristics	
Submission time	The time the task is going to be submitted
Number of jobs	Number of jobs the task comprises
Job duration	The amount of time a worker needs to spend on one job
Skill requirements	Skills that are required to perform a job
Payment	Monetary reward for a job per time period
Service Level Objectives	
Deadline	The time until all the jobs must be finished
Quality	Expected average quality of a job output. The average suitability of assigned workers for the task must be higher or equal to this value.
Minimal quality	Optional. Minimal quality of a job output. The suitability in each assignment must be higher or equal to this value.

the workers own (Line 3-13). Hence, the difference between the required and the worker's own skills (weighed by the factor α_w) influence the worker's skill update.

4.4 QoS and SLA

A crowdsourcing service essentially receives a task from a consumer and returns outputs for all contained jobs after they have been processes by crowd workers. Metrics in Table 4.1 characterize the task and the Quality of Service, and together form a Service Level Agreement.

The outlined metrics are applicable for a particular service invocation. On the one hand, service calls vary in terms of amounts of jobs and skill requirements therefore requiring different crowd capacities, on the other hand, the amount of available workers changes with time. Therefore, feasible guarantees can substantially vary for each service call and need to be considered individually, which requires such metrics as *submission time* and *deadline*.

It is possible to extend the model with more standard service quality performance metrics such as *throughput* or *response time*, and to make an agreement for a series of service invocations. In this case, to support such an extension, a number of agreements each for one service invocation from these series, needs to be established. Other QoS metrics such as *reliability* or *availability* can be inherited from the underlying technical architecture of the platform.

Before a consumer submits a task, an SLA for this task is negotiated. At first, the consumer provides the characteristics of the task. Secondly, the platform estimates possible options regarding the Service Level Objectives considering the status of the crowd and other active tasks or scheduled tasks. After that, the consumer decides, which option is the most suitable, and, finally, the agreement is established. If the offered options are not satisfying, the consumer can restart the negotiation procedure with different task characteristics (e.g., with different *Payment* or *Minimal quality* parameters). The platform takes the agreement into consideration when negotiating other agreements and scheduling the tasks. If the consumer doesn't have the actual task

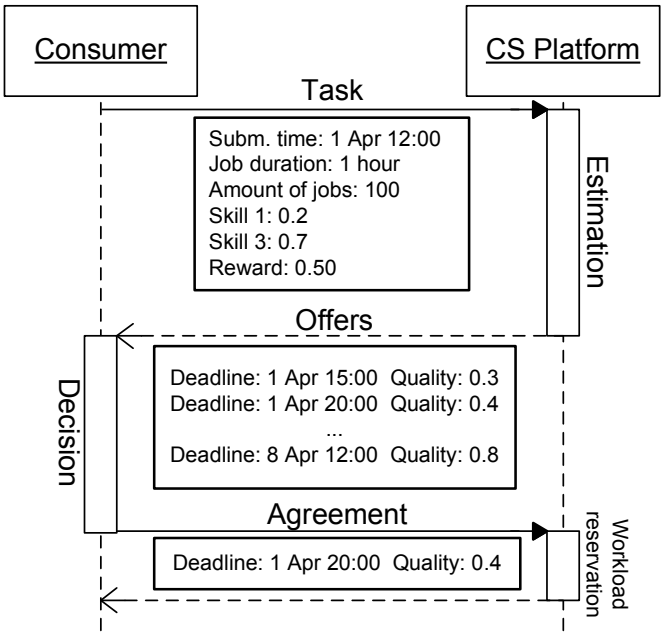


Figure 4.2: SLA negotiation

contents at the moment, but is certain to provide it in the near future and knows the parameters of the task, then the SLA can be negotiated in advance of the actual submission by setting the *submission time* accordingly. Figure 4.2 depicts the process of negotiating the SLA.

The most important ingredient of negotiation process is the estimation step which is considered in detail in Section 4.6.

4.5 Scheduling

Given tasks with negotiated SLAs and the information about worker skills and availability, the platform has to schedule the assignments. On the one hand, it has to fulfill the promises negotiated with consumers, on the other hand, the objective of scheduling is to maximize the overall quality. Estimation and scheduling components can not be decoupled, as one of them should support another: either scheduling should adapt to given estimates, or the estimation should be done according to the logic of scheduling. In a real system, even a hybrid approach can be applied. In our work, however, we choose the scheduling component to be the primary one, because it is more simple in approach and allows to evaluate the overall capabilities of scheduled crowdsourcing architecture regarding the quality it can produce.

The scheduling component of our framework therefore ignores the average quality objective of an SLA and makes it a responsibility of the prediction component to estimate a feasible value which can be then satisfied in the scheduling process. The considered algorithms thus only try to maximize the overall quality while fulfilling task deadlines. We assume that missing a deadline

cannot be justified by any quality gain, thus, meeting a deadline is the first-priority objective, and the quality maximization is the second-priority objective.

```
   Input  : currentTime current time
   Input  : tasks active tasks
   Input  : workers crowd workers
 1 for task ∈ tasks in the order of ascending task.deadline do
 2 |   stepsToDeadline = (task.deadline − currentTime+1) / task.duration - 1 ;
 3 |   if stepsToDeadline > 0 then
 4 |   |   if (task.deadline − currentTime + 1) % task.duration) > 0 then
 5 |   |   |   toTake = 0 ;
 6 |   |   else
 7 |   |   |   toTake = Trunc(task.numberOfJobsToDo/stepsToDeadline) ;
 8 |   |   end
 9 |   else
10 |   |   toTake = task.numberOfJobsToDo ;
11 |   end
12 |   while toTake > 0 AND some workers are still available do
13 |   |   Assign a job of task to most suitable available worker among those whose
   |   |   minimal wage is less or equal to task's payment and whose suitability is more or
   |   |   equal to minimal quality for the task ;
14 |   |   toTake = toTake − 1 ;
15 |   end
16 end
```

Listing 4.2: Greedy scheduling algorithm.

Listing 4.2 describes a base scheduling algorithm which is used in our platform. The idea behind the algorithm is that the best quality is achieved when a task is assigned to most suitable workers. The quality is higher when a task is performed by a smaller number of best workers, but this number should not be too small, so the task can be finished until the deadline. This number is calculated in $toTake$ for each active task.

For simplicity, a deterministic time model is used in algorithms and simulator, so the time is discreet and is represented by sequential equally long time periods. A time period can represent, e.g., 10 minutes or an hour.

Tasks for which SLAs were negotiated first are assigned in the first place, which ensures that the resources counted during the SLA estimation are not used for other tasks. This rule implicitly realizes the workload reservation.

As it can be noticed, worker availability data is only used to check whether a job with a certain duration can be assigned to a worker. However, this data is used more extensively in the estimation of feasible SLA offers.

In an attempt to improve the algorithm's efficiency, we tried a number of heuristic extensions:

- Based on reported short-time worker availability, assign less jobs at a given time to wait for more suitable workers to become available (while avoiding possible crowd "overloads")
- Assign more jobs at a given time if the suitability of additional workers is almost as good as the suitability of best workers.
- Having $toTake$ numbers calculated, optimize the worker-task assignments for each time period using an optimization framework.

However, as shown in Sect. 4.7, such extensions do not give a substantial improvement. We believe that the reason of such a weak improvement is the size of the crowd: if a worker cannot be assigned to a due task, in most of the cases a good enough replacement for the worker can be found. The refinement of the algorithm can be done according to the particular crowd characteristics that can be estimated only when the system is used by real users in the commercial operation.

4.6 SLA Offer Estimation

SLAs provide an additional value for services. However, when an SLA is negotiated with a customer, the platform has to make sure that this SLA is feasible and will not endanger other agreements.

If too many jobs are scheduled to the same period, there could be not enough available workers to withstand the workload, so some deadlines will be missed. The platform thus should determine the *earliest deadline* which the customer could set up for the his/her task in such a way so the timely execution of other tasks is not endangered.

Different outcome quality can be expected from different workers. If a close deadline it set, then more workers must be involved, and, consequently, the average result quality will be lower than in the late deadline case, where the smaller amount of best workers would do all the jobs. Therefore, there is a trade-off between the task deadline and the resulting average quality of the task. Estimating and explicitly presenting such a trade-off to the consumer clarifies what s/he can expect when submitting a task. To achieve this, the platform needs to predict *quality by deadline* efficiently for multiple deadlines.

As was discussed in the previous section, the platform is grounded upon the scheduling component. Therefore, the estimation procedure is bound to the scheduling algorithm. The estimation thus can be performed by simulating the scheduling process using the statistical data about workers' behaviour and availability, and considering earlier submitted or negotiated tasks. Then it is possible to predict which and how many workers might be available for the task which is being negotiated. This approach is implemented in our platform and considered in detail below.

Firstly, the platform estimates the realistic simulation parameters.

Worker's availability. Although it is impossible to predict whether a particular worker is available at particular time, the approximate availability of each worker can be predicted from the history and the reported short-time availability. In our implementation, the availability of each worker is generated randomly for the simulated period based on his/her recent schedule.

Job duration accuracy. The time that a worker needs to finish a job can differ from the specified job duration. The reasons can be an inaccurate estimation of the job duration from the consumer, the slow speed of the worker, or the difficulty of the particular job. We discuss this issue further in Section 4.9. The accuracy for already submitted tasks can be estimated based on prior assignments of these tasks. We estimate the overall job accuracy in our simulation.

Suitability. As mentioned in Section 4.2, worker-task suitability is calculated by the platform and can be modified with time. If a task of a kind is submitted to the platform for the first time, the suitability can be calculated imprecisely. However, the following task submissions of this kind (with the same skill requirements) will use refined values. We discuss this issue further in Section 4.9. In any case, the simulation can only make use of the latest calculated suitability values and assume the quality of a job equal to the suitability of the performer and the corresponding task.

Secondly, given currently submitted tasks and the parameters (see Table 4.1) of the task at hand, the platform simulates the scheduling process and estimates feasible *earliest deadline* and *quality by deadline* for this task.

The simulation period starts with the current state of the real platform. The size of the period can be either fixed (e.g., predictions for up to 5 days) or depend on the quality increment (e.g., if by prolonging the deadline by 1 day the expected quality is increased by less than 0.05, then stop the prediction). Durations of jobs are simulated according to the estimated accuracy.

At each step, the submitted (or pre-submitted) tasks are assigned to workers using the greedy scheduling algorithm (see Listing 4.2). After conducting the assignment, the prediction algorithm is executed for the current step (see Listing 4.3): workers, that are available and were not assigned by the scheduling algorithm, are examined as candidates for the negotiated task `nTask`. This is performed each job duration period of `nTask` using so-called array of average suitability of best workers (`avgSuit`). The kth element of this array represents an approximated suitability value of the kth most suitable worker for all prior simulation steps. This element contains the summed suitability and the amount of workers that were considered at this position, so the average value can be calculated on each step. The algorithm thus adds the *suitability* value and increments the *amount* for each element that corresponds to an unassigned worker (lines 3-10 of Listing 4.3).

After that, if the total amount of available workers at previous steps exceeds the amount of jobs in `nTask` with certain excess, the prediction of quality is calculated for the current step. The algorithm assumes that the best available workers would be evenly assigned for the task (true for the greedy scheduling algorithm) and, using `avgSuit` array, it estimates the expected quality produces by most suitable available workers for all previous steps using `avgSuit` array (lines 11-21 of Listing 4.3). It is assumed that a worker is late with the job with probability of 0.5 (this assumption holds if the job duration is set accurately). Eventually, the average quality which represents the prediction for *quality by deadline* for the current step, as if it was the deadline, is calculated. The *earliest deadline* is the step where it was first estimated that the number of available workers at previous steps exceeds the amount of jobs.

As the scheduling algorithm prioritizes tasks by submit time, no “collisions” are expected: on the one hand, the prediction doesn’t take already reserved resources into account, on the other hand, if the task is submitted, the assessed resources will not be assumed available for the tasks

```
   Input  : time current time
   Input  : nTask negotiated task
   Input  : avgSuit the array of average suitability of best workers
   Input  : ttlAvWorkers total number of available workers
   Input  : Δ excess ratio (0.8 used)
 1 if (time − nTask.callTime) % nTask.jobDuration == 0 AND
   (time > nTask.callTime) then
 2     i = 0 ;
 3     for worker ∈ workers in the order of descending suitability do
 4         if worker is available, fulfills minimal quality, and his/her minimal wage is less
           or equal to nTask's payment then
 5             avgSuit[i].suitability+ = suitability of worker ;
 6             avgSuit[i].amount + + ;
 7             i + + ;
 8             ttlAvWorkers + + ;
 9         end
10     end
11     if ttlAvWorkers ∗ 0.5 ∗ Δ > nTask.numberOfJobs then
12         toTake = nTask.numberOfJobs ;
13         i = 0 ;
14         q = 0 ;
15         while toTake > 0 do
16             take = Max(1, floor(Min(avgSuit[i].amount ∗ 0.5, toTake))) ;
17             toTake− = take ;
18             q+ = avgSuit[i].suitability ∗ take ;
19             i + + ;
20         end
21         Return {time,q/nTask.numberOfJobs} ;
22     end
23 end
```

Listing 4.3: Prediction algorithm.

submitted afterwards. Therefore, when an agreement is established, the workload reservation is performed seamlessly.

4.7 Experiments

In this section we demonstrate the efficiency of scheduled crowdsourcing. To evaluate our platform and algorithms, we set up a simulated environment that comprises a crowd which perform tasks and consumers who submit tasks and provide feedback. Simulation of a real crowdsourcing environment is challenging due to the lack of comprehensive statistical data in this area. We tried our best to prognosticate the meaningful simulation parameters based on available data [34, 35] and common sense.

In the experiments we evaluated the efficiency and performance of algorithms for task scheduling, worker skills updating, and SLA estimation. We first provide a description of the overall setup, and then explain the types of experiments and show the corresponding results.

4.7.1 Experiment Setup

In our experiments we use a set of 10 skills for describing worker skills and task skill requirements.

Customers. The customers submit tasks to the platform and provide the feedback on completed jobs. Tasks are submitted randomly while ensuring the average crowd workload and avoiding overloads.

Each task comprises skill requirements, number of jobs, and a deadline. During each time period of the simulation, if the *Task Limit* has not been reached yet, a new task is submitted to the system with *Task Concentration* probability. The job duration is calculated as $Min(1 + abs(\phi/2 * \sigma), \sigma + 1)$, where ϕ is a normally distributed random value with mean 0 and standard deviation 1. The deadline is assigned randomly according to *Steps To Deadline* parameter. The number of jobs is calculated so that the crowd workload is near equally distributed among the tasks, and the average workload remains close to *Intended Schedule Density*. The parameters and their values are described in Table 4.2.

Skill requirements are generated so that each skill with approximately equal probability either equals 0 which means that this skill is not required for the task, or is in $(0, 1]$ range. The random values for the $(0, 1]$ range are normally distributed (mean = 0.4, variance = 0.3).

The feedback that a consumer provides for a job is generated using the real skills of the worker which were assigned for this job. In contrast to the estimated skills, these real skills are unknown to the platform and are only used to simulate the real outcome quality (by calculating the suitability with these skills). This quality is thus reported as the feedback.

Crowd workers. The workers are assigned for jobs and return the result of job processing. Each worker has the claimed skills that s/he initially reports to the platform, and the real skills. The real skills are generated randomly with normal distribution with 0 mean and variance of 0.3. Then, the reported skills are initiated as real skills with injected error (normally distributed with mean value equal to the real skill and variance of 0.2). Suitability of crowd workers for some randomly selected tasks is depicted in Figure 4.3.

Table 4.2: Task generation parameters

Name	**Description**	**Value(s)**
Tasks Limit	The total number of submitted tasks	200
Job Duration Sigma (σ)	Describes the deviation and the maximum for job durations	20
Steps To Deadline	Average maximum number of jobs of a task that a single worker can finish until the deadline.	50
Task Concentration	The probability of new task submission for each time period.	0.35
Intended Schedule Density	Target assignment ratio for each time period.	0.2 - 0.7 (step 0.1)

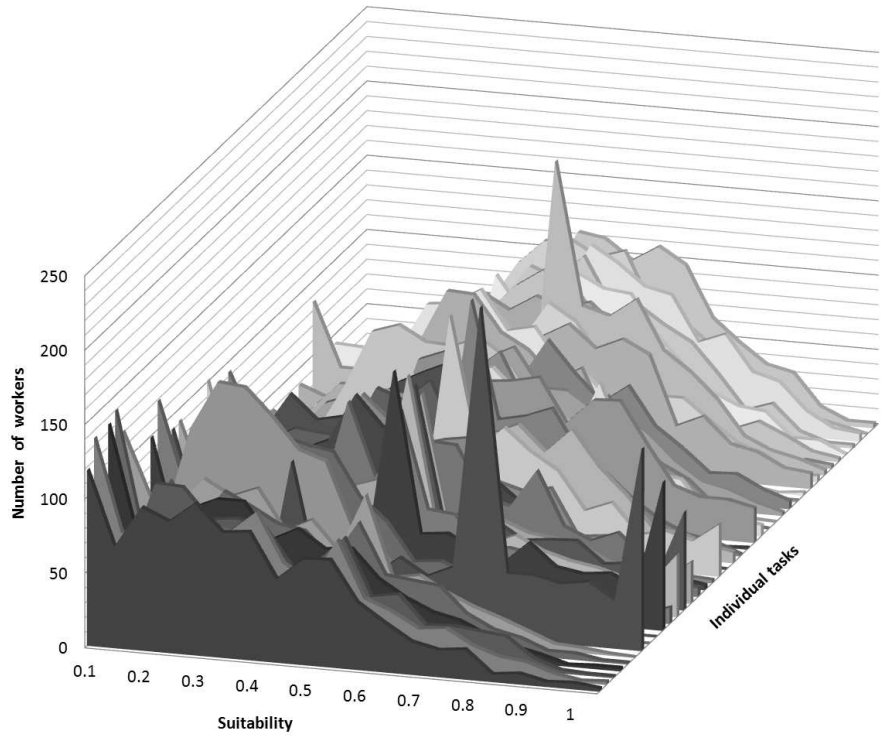
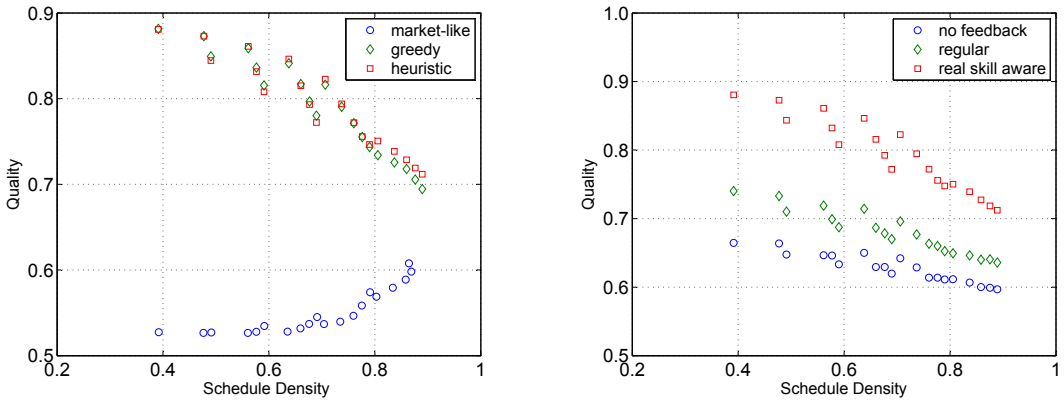


Figure 4.3: Suitability of the crowd for a random set of tasks



(a) Various schedulers.

(b) Skill update efficiency.

Figure 4.4: Task scheduling

The crowd size in experiments was 1000 workers. This size is big enough to enclose the diversity of workers, but still allows for fast simulation. We tried to use 10000 instead, but the results did not change substantially. Workers can be unavailable at certain periods. In our experiments we use a *Workers Unavailability* parameter which indicates the mean ratio of unavailable workers for each period of time (values used: $0.2 - 0.6$, step 0.1). The busy periods are generated randomly, but have a continuous form which reproduces human behavior. The amount of time that takes a worker to finish the job is the *Job Duration* with injected variations. In our experiments we used a value of 30%, which means that a job can be executed for $0.7 - 1.3$ of job duration. This reflects the random nature of the real world.

4.7.2 Task scheduling

Various schedulers. To demonstrate the advantage of skill-based assignment, a scheduler which mimics a market-like platform was compared with the greedy scheduler and the heuristically-enhanced greedy scheduler (See Sect. 4.5). In market-like scheduling, the assignment followed the logic that randomly chosen workers were picking the most suitable for them active tasks. The results are shown in Figure 4.4a. In tests with high schedule density (about 0.8 or more), market-like assignment performed better than in tests with low density, because workers had more tasks to choose from. However, about 15% of task deadlines were violated in these tests, because workers aimed to fulfill their own preferences rather than the goals of the system. For the rest of the tests, the average quality was 1.5 times better for skill-based scheduling in the large. This clearly shows the benefit of skill-based scheduling. The heuristics did not improve the greedy algorithm substantially, and for some tests even impaired it.

Skill update efficiency. To demonstrate the efficiency of skill update mechanism, we compared the regular simulation which implements the logic described in Sect. 4.3.2 (“regular” series) to upper and lower bounds. The series named “no feedback” represents the lower bound and only the initial information on the profiles is used for scheduling. An upper bound to the algorithm is shown by the series of “real skill-aware”. In this case the exact skills of a worker

are known to the system. The improvement of the skill update mechanism over the lower bound is evident and keeps performing better at any scheduling density. In the experiments of Figure 4.4b the improvement over no feedback remains between 10-15%. As Section 4.3 explains, the reason why it never reaches real skill awareness is twofold. First, the scheduling strategy need some input right from start when only few feedback is available. Second, the feedback is a single value that describes the performance depending on ten different skills. Also, a skill value greater than required calculates the quality with the lowest value required. Even if there was enough data an accurate calculation would not be feasible in all cases. Thus, we decided to stick to a simpler quicker update algorithm that provides almost constant quality improvement and, after all, supports quality negotiation with a considerable and steady lower bound to make agreements.

4.7.3 Prediction Accuracy

To explore the potentiality of prediction, we implemented and tested the algorithm in our simulator. The prediction mechanism operated completely separately from the simulated environment. The parameters of the prediction's simulator such as availability of workers and job duration accuracy, are estimated or generated based on the simulated environment's prior activity only. Also, the duration of individual assignments differs if those happen to take place in both simulators.

First, we made the predictions for 50 tasks in the middle of simulation for 500 time periods. It better reflects a real crowdsourcing environment, as there are both tasks being in progress and new tasks being submitted. Then, we checked each prediction by varying the deadlines of corresponding tasks and running the simulation in the identical setting. The resulting accuracy is depicted in Figure 4.5. From the total of 2041 experiment, in 98% the deviation was less than 0.1, in 85% of experiment - less than 0.05. The average deviation was approximately 0.025. Evidently, the prediction is less accurate for early deadlines, and more accurate for late dealines.

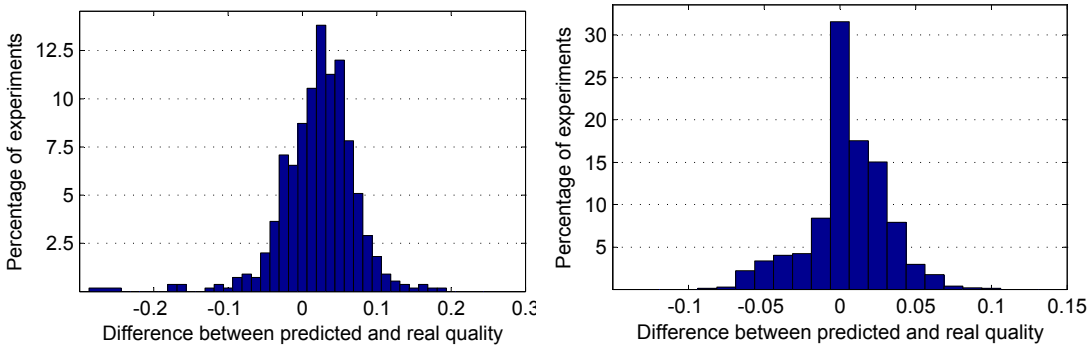
The results indicate that the algorithm can be successfully applied for negotiating the agreements. Moreover, the guaranteed values can be calculated depending on deadline remoteness. For example, the guarantee can be given as the predicted quality reduced by 0.2 in case of early deadlines, and reduced by 0.1 in case of late deadlines.

4.7.4 Performance

The performance of scheduling and skill updating in a high workload test (10000 workers and 1000 tasks) was good enough for a period size of one minute. Thus, the performance is not a concern, since the real period size is likely to be bigger (e.g., 10 - 60 minutes).

We ran the prediction in the same setting while varying the size of the crowd from 1000 to 10000. The prediction overhead is depicted in Figure 4.6. System parameters were Intel Core 2 Quad 2.40 GhZ with 6 GB of RAM (the algorithm is not parallelized so only one core was actually used).

The results show that the approach can be used in a near real-time setting. The overhead of several seconds would not play a huge role when negotiating the agreement and is therefore acceptable.



(a) Early deadlines ($<$ 10 job durations)

(b) Late deadlines ($>$ 10 job durations)

Figure 4.5: Prediction accuracy. Histograms describe the amount of experiments (in percentage of the total number of experiments performed) that produced one or another accuracy. Subfigure (a) corresponds to experiments in which the deadline was set to be less than 10 job durations of the task, SLA of which is being negotiated; Subfigure (b) corresponds to experiments in which the deadline was set to be more than 10 job durations.

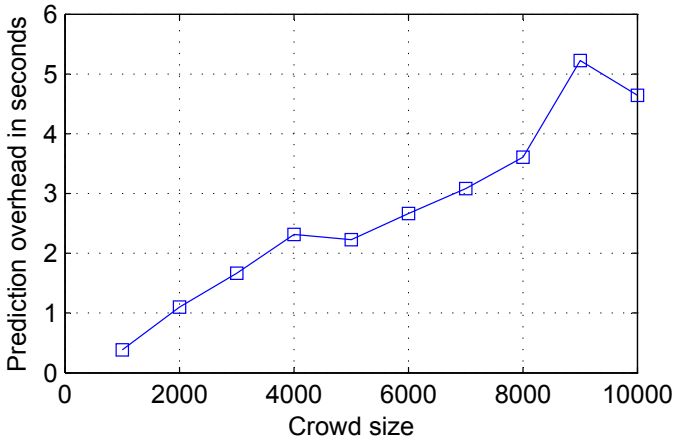


Figure 4.6: Prediction performance

4.8 Related Work

In this work we position crowdsourcing in a service-oriented business setting by providing automation. In crowdsourcing environments, people offer their skills and capabilities in a service-oriented manner. Major industry players have been working towards standardized protocols and languages for interfacing with people in SOA. Specifications such as WS-HumanTask [24] and BPEL4People [4] have been defined to address the lack of human interactions in service-oriented businesses [49]. These standards, however, have been designed to model interactions in closed enterprise environments where people have predefined, mostly static, roles and responsibilities. Here we address the service-oriented integration of human capabilities situated in a much more dynamic environment where the availability of people is under constant flux and change [11].

AMT offers access to a large number of crowd workers. With their notion of HITs that can be created using a Web service-based interface they are closely related to our aim of mediating the capabilities of crowds to service-oriented business environments. Despite the fact that AMT offers HITs on various topics [35], the major challenges are to find on request skilled workers that are able to provide high quality results for a particular topic (e.g., see [3]), to avoid spamming, and to recognize low-performers. To the best of our knowledge, these problems are still not faced by AMT. In this work we focus on those issues.

Another shortcoming of most existing real platforms is the lack of different and comprehensive *skill information*. Most platforms have a simple measure to prevent workers (in AMT, a threshold of task success rate can be defined) from claiming tasks. In [77], the automated calculation of expertise profiles and skills based on interactions in collaborative networks was discussed.

In [39], a quality management approach for crowdsourcing environments is presented. Unlike our profile management, this work doesn't support multiple skills, but concentrates on a single correctness dimension. On the other hand, if there is a specific need for such a quality management technique, the profile management can thus be replaced with it by correllating the correctness and suitability, as this module is decoupled from the rest of the platform as mentioned in Section 4.2.

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 [63] 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, [79] near-optimal resource allocations and reallocations of human tasks were presented. Staff scheduling related to closed systems was discussed in [10, 20]. 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 [41] details the challenges for collaborative workforce in crowdsourcing where activities are coordinated, workforce contributions are not wasted, and results are guaranteed.

Although the idea of QoS-enchanced crowdsourcing was discussed before [40], to the best of our knowledge, no work was devoted to deadline- and quality-centric predictions and guaran-

tees in crowdsourcing. In [70], semi-automatic assignment mechanism was proposed. This work assumes that SLAs are established with the workers, and Community Brokers are hold responsible for assignments in various crowd segments. However, no SLAs with crowdsourcing service consumers are considered. Advanced market-based crowdsourcing platform which takes the suitability of workers to tasks into account was proposed in [75]. For a given task, it organizes an auction among only the most suitable workers to increase the overall quality and motivate workers to improve their skills. Again, this model doesn't provide predicting capabilities and doesn't support SLAs.

The considered scheduling problem is based on worker-task suitability, task deadlines, and workers availability with the objective of maximizing the job quality does not correspond to any of well-known scheduling problems [66]. The problem can be formulated as unrelated machines in parallel with deadlines, but the optimization objective is different from objectives related to processing time which are commonly studied in the domain. Staff scheduling [10] designs workers' schedules which should fulfill certain requirements and cover the tasks that need to be done in a given planning horizon. Crowdsourcing's idea is the opposite: the workers define their schedule by themselves, and the platform assigns available workers to tasks in progress dynamically.

4.9 Discussion

In this section we discuss some disputable aspects of our approach.

The operation of the platform is influenced by several subjective characteristics provided by consumers, such as skill requirements, job duration, or feedback. However, it is of consumer's interest to specify them accurately. For example, if s/he underestimates job duration, then some deadlines might be broken or the resulting quality can be lower than agreed. This situation can be spotted by the platform as the majority of workers would do the jobs longer than expected. Then, the input data from the consumer can be considered misleading, and the agreement can be denounced. If the consumer overestimates the job duration, then agreements are not endangered as most of workers would be faster than expected, however, this would produce underestimated predictions.

The same can be applied for other characteristics: if the platform spots that the majority of assignments do not correspond to expectations - then they were probably specified inaccurately. Moreover, the guarantees are given according to the platform's sight, so the agreement can include the condition that the platform cannot be responsible for the outcome quality if the input data was inaccurate, and the consumer is responsible for the accuracy of his/her input. The consumer, on the other hand, can rely on the prior experience or submit some sample tasks to adjust these parameters.

Crowdsourcing presumes a substantial amount of registered workers, and the scheduled crowdsourcing puts even more restrictions on what the workers can do and when. The feasibility of real-world deployment of such a platform can thus be questioned. Some contrary arguments, however, are that about 20% of AMT workers consider AMT their primary income, and about 20% of AMT workers complete more than 200 jobs per week [34]. Considering, that AMT claims that more than 500 000 workers are registered in the platform, one can conclude

that there is significant amount of people who are willing to perform jobs at the regular basis. Moreover, the payments for jobs in scheduled crowdsorcing can be bigger due to the added value of SLAs. Finally, the scheduling mechanism can be upgraded to take account of workers' preferences, while keeping the assignments compliant to established agreements.

4.10 Summary

In this chapter we presented a skill-aware crowdsourcing platform model which allows to provide crowdsourcing services with SLAs and to control the task performance quality, and a prediction technique for negotiating feasible agreements. In contrast to existing crowdsourcing platforms such as AMT, which follow a task market-oriented approach, our platform model is based on services computing concepts. Such a model is typically applied in enterprise workflow systems using, for example, the WS-HumanTask specification to design human interactions in service-oriented systems. However, WS-HumanTask and related specifications lack the notion of human SLAs and task quality. In our approach, negotiated SLAs and monitoring help to assign task requests to suitable workers. Thus, our platform ensures quality guarantees by selecting skilled workers. We introduced the proof-of-concept implementation with particular algorithms for task scheduling and worker profile management. The applicability of the platform design was proved in a simulated environment. The experimental results shows the clear advantage of skill-based scheduling in crowdsourcing, as the average quality is certainly better in the large comparing to the case when the workers choose tasks by themselves. The skill monitoring and updating mechanism improves the overall quality by 10-15%.

The potentiality of the prediction technique is evaluated thorough experiments which show that such an environment is predictive in spite of its inherent uncertainty. The proposed base algorithm can be considered rather precise (average quality deviation 0.025), and, therefore, can be applied for negotiating the agreements. The experiments show that the prediction accuracy depends on deadline remoteness, the guaranteed values therefore can be adjusted accordingly. These results show that crowdsourcing platforms can be organized to provide quality guarantees for the consumers. They can strengthen the certainty and predictability in process planning and design, and enable Service Level Agreements with customers. From a Business Process Management perspective, such guarantees provide an additional value, thus promoting more advantageous crowdsourcing services.

CHAPTER 5

Optimized Execution of Business Processes with Crowdsourcing

The crowdsourcing approach presented in Chapter 4 is applicable for simple tasks that can be generally finished in small amount of time and don't require the worker to know the context of the task, corporate standards, conventions or practices. Crowdsourcing of complex tasks however requires workers to be affiliated with the consumer to certain extent. Firstly, because of security reasons, as the worker who gets the task might need to have an insight into the company's processes and to have an access to company's data (e.g., development of features for software components requires the partial knowledge of the systems and infrastructures deployed in the company). Secondly, because such tasks are more costly and more critical for business, and therefore require certain confidence in possible candidates. Due to these reasons, crowdsourcing of such tasks cannot provide the similar scalability potential compared to crowdsourcing of simple tasks, and makes it unrealistic to perform direct assignments alike to the approach presented in Chapter 4. It still however can deliver more flexibility and allow to reduce costs via *competition*.

In this chapter we propose a framework for adaptive execution of business processes on top of an *internal* crowdsourcing platform. Based on historical data gathered by the platform we mine the booking behavior of people based on the nature and incentive of the crowdsourced tasks. Using the learned behavior model we derive an incentive management approach based on mathematical optimization that executes business processes in a cost-optimal way considering their *deadlines*. We evaluate our approach through simulations to prove the feasibility and effectiveness. The experiments verify our assumptions regarding the necessary ingredients of the approach and show the advantage of taking the booking behavior into account compared to the case when it is partially of fully neglected.

5.1 Overview

Crowdsourcing has the potential to give companies flexible access to a talent pool of almost unlimited size. In fact, according to an internal strategy document that leaked out in early 2012, IBM plans to employ a radically new business model [80]. It involves to let the company run by a small number of core workers. A dedicated Web-based platform is used to attract specialists and to create a virtual "cloud" of contributors. Similar to cloud computing where computing power is provided on demand, IBM's people cloud would allow to leverage a flexible on-demand workforce. Today's crowdsourcing systems are still relatively simple and only suitable for non-critical, atomic tasks requiring minor efforts. In particular, Amazon offers a task-based crowdsourcing marketplace called Amazon Mechanical Turk (AMT) [5]. Requesters are invited to issue human-intelligence tasks (HITs) requiring a certain qualification to the AMT. The registered customers post mostly tasks with minor effort that, however, require human capabilities (e.g., transcription, classification, or categorization tasks [35]).

We foresee that in the future companies will increasingly use crowdsourcing to address a flexible workforce. However, it is still an open issue how to carry out business processes leveraging crowdsourcing. In case if a crowdsourcing platform provides guarantees (see Chapter 4) regarding QoS, then it is possible to use QoS-based optimization (see, e.g., [73]) to fulfill the process-level guarantees while minimizing costs. However, as discussed above, providing such guarantees for particular tasks within an internal enterprise crowdsourcing platform is not realistic. Nevertheless, if there are enough historical information about the prior assignments, it is possible to estimate the needed parameters for the published tasks to fulfill process-level guarantees. The main problem is, however, that people book tasks voluntarily in such competition-based crowdsourcing, which means the only way to influence booking and execution times of single tasks is to either change incentives or modify other aspects of a task, e.g., define a later deadline.

The contribution of this chapter is an enterprise crowdsourcing approach that executes business processes on top of a crowdsourcing platform. For each single task in the business process we reason about the optimal values for incentive and time allotted when crowdsourcing them. The goal is to carry out the business process with minimal investments before the deadline. During execution of the business process we constantly monitor the progress and adjust these values for tasks that have not been booked by a worker yet. Our approach for calculating optimal values is based on mining historical data about task executions, e.g., which influence higher rewards have on the booking time, analyzing the current status of the business process, and quadratic programming, which is a mathematical optimization formulation that can be solved efficiently. We evaluate our approach through simulations for different process sizes and structures. The experiments show the effectiveness of the approach and demonstrate its adaptivity to the poorly predictable crowdsourcing environment. We prove that booking time is one of the key features for optimizing process execution in context of crowdsourcing, and that taking it into account can reduce the deadline misses up to 14%.

Type	Description	Ready to start	Effort	Time Allotted	Reward	
JavaScript	Click here	now	21h	5d	$2,090	Book
UI Design	Click here	2012-04-01 14:00	12h	7d	$810	Book
.NET	Click here	2012-04-03 10:00	3h	14d	$110	Book
...	...	...	...	...	...	...

Figure 5.1: Schematic UI for the enterprise-internal crowdsourcing platform of a large software company

5.2 Motivating Scenario

We consider a scenario of a large software company that plans to deploy an enterprise crowdsourcing platform for software development tasks. Figure 5.1 schematically shows how tasks are presented to the employees. Each task is described by a type and a textual description. The third column gives an estimate when the employee will be able to start working on the task. Since a task may require input from other tasks the actual start date and time can deviate from the announced one. Effort provides information about the time effort necessary to finish the task. Time allotted defines the time frame in which the task is supposed to be processed. It starts when the task is ready to start or when the task is booked, whatever is later. The reward tells the employee how much he will get for successfully processing the task on time. Instead of money, rewards could also consist of more abstract reward points. When an employee books a task s/he is responsible for delivering the results within the allotted time. Tasks can be booked before they are ready to start. As long as tasks are not booked the system may modify allotted time and reward. The crowdsourcing platform generates and stores log information for each processed task, as illustrated in Figure 5.2. We assume that at least information regarding the task type, time effort, time allotted, the reward for which the employee actually booked the task, and the time it took from publishing to booking is stored for each task processed via the platform. Usually paying much for a task would reduce its booking time; also, having a high allotted time should make tasks more attractive compared to tasks with a tight deadline.

Type	Effort	Time Allotted	Reward	Booking Time
.Net	24h	14d	$520	1d
.Net	12h	4d	$325	5h
.Net	36h	5d	$570	6d
.Net	36h	5d	$850	5h
UI Design	12h	8d	$405	3h
...	...	...	...	...

Figure 5.2: The crowdsourcing platform maintains a database containing information related to the processing of each task

However, the software company has problems to map its business workflows to the crowdsourcing platform. Figure 5.3 shows a workflow describing a business process the company wants to execute. The aim is to integrate a new plugin into an existing software product. The plugin consists of two features that together make up the functionality of the plugin. Each fea-

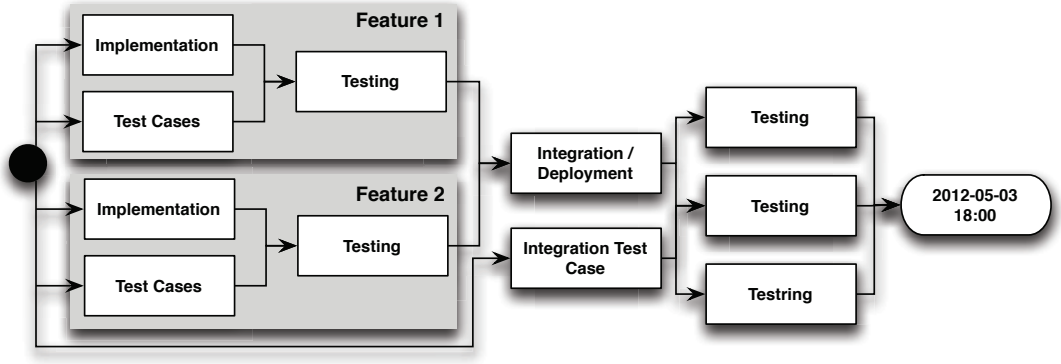


Figure 5.3: An exemplary business workflow describing the development of a new plugin for a software product. The plugin consists of two features that need to be developed, integrated, and deployed into the software product. Each implementation step is followed by testing. There is a deadline for the completion of the whole plugin.

ture consists of three tasks, the actual implementation and the writing of test cases, which can be done in parallel, and the testing of the implementation using the test cases. After both features have been implemented and tested, an integration and deployment step is necessary to ensure proper installation into the software product. Three different testing tasks are to ensure the high quality of the plugin; all three are based on the integration test case.

The introduced business process is simple yet helps to understand the challenges addressed by this work. The question is how to crowdsource the tasks of the workflow using the crowdsourcing platform, i.e., how to set the values for the crowdsourced tasks. Type, description, and estimated effort of each single task typically are already available; the approximate ready-to-start times can be computed by the crowdsourcing platform once it has scheduled all predecessor tasks. This is relatively straightforward yet not trivial since it involves the computation is based on constant monitoring and recalculations to cover deviations from the schedule, e.g., delayed or early finished tasks. However, there is no obvious solution at all for how to determine the values for time allotted and reward. Time allotted should be assigned in a way to ensure the adherence to the deadline. Rewards should consider the usual “market prices” of the respective tasks, but also sometimes be increased to strengthen the competition among employees and ensure that tasks critical to the success of the workflow are timely booked. Also, in case the allotted time is short to process the task, then this should be reflected in the reward.

5.3 Approach

The goal of our solution is to ensure the timely execution of business processes that contain crowdsourced tasks while minimizing the expenses associated with crowdsourcing rewards. We assume that the crowdsourcing platform allows to specify allotted time and reward for each task (as described in Section 5.2). The expenses can be reduced by setting lower rewards for tasks, however, if the reward is too small, a task might stay not booked for too long, if booked at all.

Such situations can significantly affect the execution of the process, and become a reason of missed deadlines. The allotted time also affects expenses: it is less likely that an employee decides to take an urgent task for a regular reward. Hence, there can be more employees interested in a non-urgent task at a lower reward, because some of them might have less experience in this type of tasks, and would like to improve, but need more time allotted. Obviously, the time allotted also directly influences the process completion time. The main idea of our approach is to find a most beneficial trade-off between rewards, allotted times, and expected booking times for crowdsourced tasks in the process.

Specifically, we address the following questions:

How to estimate booking time? The time it takes someone to book a task after it is announced in the platform, or the *booking time*, can be influenced by various factors. However, we are convinced, that it is driven mostly by the strength of the competition among employees. Therefore, tasks, whose time allotted and reward combination satisfies demands of more people, are generally booked earlier, and vice-versa. Undoubtedly, even if an employee is satisfied with the time allotted and the reward offered, s/he can still refuse to book a task, because of being too busy, not interested in this particular type of tasks, or just not being in the mood. Nevertheless, if the crowd is large enough, the trend should remain. Our approach is to determine this trend using platform logs (see Section 5.2).

How to optimize allotted times and rewards? The optimization should consider the dependency between booking times, rewards, allotted times, and the structure of the process. Also, it can happen that something goes not as expected (e.g., a worker delays a critical task, or completes it significantly earlier), so either it becomes necessary to get some tasks done faster to cope with a deadline or an opportunity to cut more costs emerges. The optimization therefore should perform adaptively, and consider the process state as well.

When tasks should be published in the crowdsourcing platform? If a task is booked by an employee, then the platform undertakes a commitment and can't demand the employee to perform faster or change the reward for this task any more. However, as mentioned above, an adaptive behavior can be advantageous, and it can be more beneficial to publish tasks later. But this should not be done too late and be aligned with the process execution state. Our approach is to publish a task when the sum of its optimized booking and allotted times, and expected execution time of subsequent activities in the process is almost as long as the time left before deadline. In Section 5.4 we experimentally prove that such an approach produces optimal results.

We thus map the described functionality to components and propose a framework for a deadline-driven reward optimization for processes containing crowdsourced tasks (See Figure 5.4). The estimator collects the statistical data from the platform logs and estimates the functional dependency for each type of task ①. The optimization component retrieves structure and state of processes ③, booking state of already published tasks ④, functional dependencies ②, and determines optimal values for booking time, reward, and time allotted. These values are further used by publishing component which announces tasks at the platform at appropriate time and updates them if needed.

Subsections below provide a detailed description of the corresponding components of the framework. Estimation should be performed for all task types before rewards can be optimized. After some time, it can be re-executed to conform to the crowd's changing characteristics. The

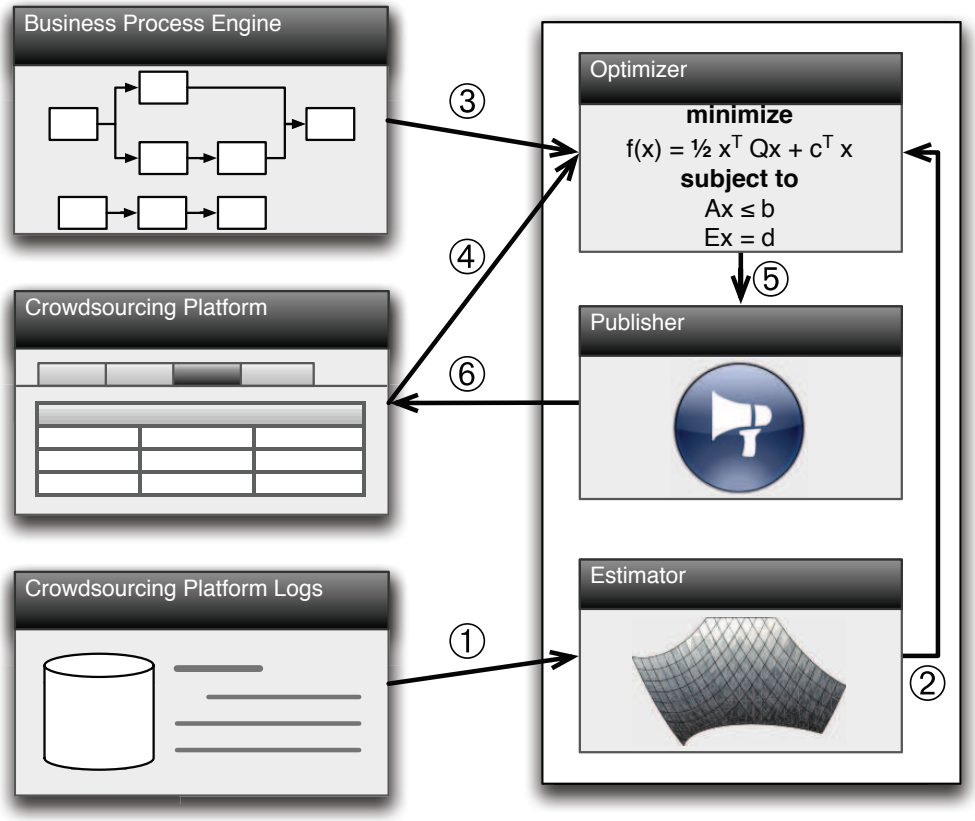


Figure 5.4: The architecture of the framework

optimization and publishing components are activated periodically thus realizing the adaptive behaviour.

5.3.1 Estimation component

As described in Section 5.2, each log entry corresponds to a processed task and includes task type, weight (e.g., in hours of effort), allotted time, booking time, and reward. Let us reference to these values as T, w, t, bt, r. The estimation for each task type is done independently. Therefore, for a particular type of task, an entry can be represented as $< w, t, bt, r >$.

Assuming that reward and time allotted linearly depend on the weight, and booking time depends on the combination of reward and time allotted, we can consider a mapping $bt' = f(t', r')$ which is populated with log entries as follows: $t' = t/w; r' = r/w; bt' = bt$, which reflects tasks with weight 1.

Further, we need to estimate a function $ub(t, r)$ which is an upper bound for mapping f at each (t, r). Even with a weak competition, a task can be booked fast due to a coincidence. Therefore, for stable prediction in the context of deadline fulfillment, we are interested in the

maximum time that it takes a task with specified reward and time allotted to get booked. The particular methods for upper bound estimation can vary according to the real setup.

Finally, using the discovered upper bound, we need to approximate the function $g(t, bt)$ that reflects the reward that need to be set for a specified time allotted and expected booking time. This function will be used in an objective function for optimization. The dataset for approximating g is obtained as a set of tuples $< r_i, t_i, ub(t_i, r_i) >$ for each (r_i, t_i) that are defined in mapping bt. We argue that g should be approximated with a 2nd degree polynomial, because (i) polynomial-based optimization is well studied [51], (ii) many optimization frameworks support quadratic programming [57] , and (iii) the 2nd degree is a good trade-off between optimization complexity and fitting accuracy for the problem. In our experiments, the difference in accuracy between 2nd and 5th degree polynomial approximation was less than 5%, whereas it was more than 20% between 1st and 2nd degree. The estimation should also determine minimum and maximum values for all arguments. The approximated function should therefore have the following form:

$$g(t, bt) = a_1 * t^2 + a_2 * t * bt + a_3 * bt^2 + a_4 * bt + a_5$$

An example of an approximated function is illustrated in Section 5.4.

5.3.2 Optimization component

The goal of the optimization is to fulfill process deadline, while trying to minimize the offered rewards. Therefore, based on the state and structure of the process and estimated dependencies, we formulate a quadratic programming problem. The constraints ensure that the process can be finished before the deadline considering all the booking and allotted times, and the optimization objective is the sum of the rewards that will be paid for tasks contained in the process.

We formally represent a process as a directed acyclic graph, where nodes represent tasks and edges represent control flow. The graph has 2 special nodes that represent the beginning and the end of the process, namely *in* and *out*. Thus, for each node in the graph, there exists a path from *in* to *out* which contains this node. A task can be started when all its incoming edges are adjacent to already finished tasks. Besides crowdsourced tasks, a process can contain simple activities, whose execution times are regarded as constants. For simplicity, *in* and *out* nodes can be considered dummy, i.e., simple activities with execution time 0. Such representation is more general than, e.g. combination of *flow* and *sequence* activities in BPEL. In this work we do not consider constructs such as conditions or loops for the sake of simplicity, although the approach can be extended to support such elements.

Each task has a property that indicates its status, which can be either `unavailable`, `published`, `ready`, `started` or `finished`. The process engine can thus launch only the tasks that are `ready`. Tasks are marked as `published` when published in the crowdsourcing platform, and change their status to `ready` when booked. For each not yet booked crowdsourced task, decision variables for allotted time and for booking time are included for optimization. The variables are restricted using the minimum and maximum values provided by the estimation component. Two categories of constraints are included into the optimization model:

1. Constraints covering all the processing and allotted times throughout all possible process execution paths from the current state. The combination of these constraints ensures that the slowest branch will complete before the deadline.

2. Constraints covering booking time for `unavailable` and `published` tasks and all possible subsequent process execution paths. These constraints ensure that booking times will not endanger the deadline fulfillment.

If a task is already booked or represents a simple activity, then its execution time is fixed. If it has been already started, then its completion time can be estimated for current situation. In both of these cases, the execution time is regarded as a constant from the perspective of the optimization. The exact algorithm for building the constraints is described in Algorithm 5.1.

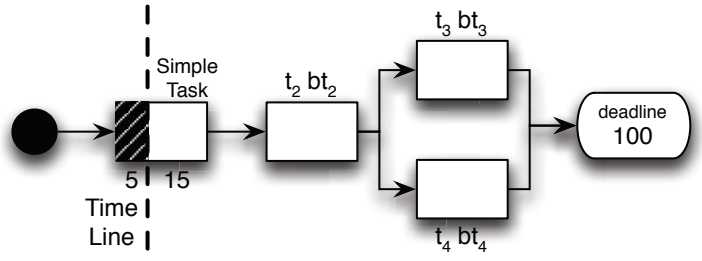
An example of the algorithm's functionality is shown in Figure 5.5. A simple activity has already started and has been processed for five time units, as illustrated by the time line. Since the expected processing time for the activity is 20 time units, the optimizer assumes that the activity finishes in 15 time units. Since there is only one started activity, the optimizer adds two constraints of first category because there exist two paths from this this activity to *out* activity. As all the other tasks are either unavailable or published, the constraints of the second category should be created for them. Therefore, two constraints for booking time t_2 are generated, covering both successor paths. For both t_3 and t_4, only one constraint is generated for each single path to *out*.

The optimization objective is composed as a sum of rewards using the scaled values of estimated dependency functions:

$$min \sum_{s \in S} (g_{type_s}(t_s/w_s, bt_s) * w_s)$$

where S - all tasks in the process, g_{type_s} - dependency function for the type of task s, t_s, bt_s - decision variables, w_s - weight of task s.

If the optimization problem turns out to be infeasible, the optimizer should try to extended the deadline and try again, until a feasible solution is found. This will ensure that even if the deadline cannot be met, then the best possible solution will be provided.



$$15 + t_2 + t_3 < 100$$
$$15 + t_2 + t_4 < 100$$
$$bt_2 + t_2 + t_3 < 100$$
$$bt_2 + t_2 + t_4 < 100$$
$$bt_3 + t_3 < 100$$
$$bt_4 + t_4 < 100$$

Figure 5.5: Example for the generation of optimization constraints for the quadratic programming problem formulation

```
input : timeToDeadline, processGraph
call : createConstraints ([], processGraph.outNode)

createConstraints( list path, task t)  {
  add t to the beginning of path;
  foreach (incoming edge e of t) {
    get adjacent node t' which is source of e;
    if ( status of t' is not finished ) createConstraints(path, t');
  }
  if (there were no incoming edges with adjacent not finished nodes)
    addConstraint(path, 1st type); else
    addConstraint(path, 2nd type);
  remove t from path;
}

addTermsToExpression(list path, constraint expression expr)  {
  foreach (t in path)
  if ( status of t is unavailable or published)
    add decision variable for allotted time of t to expr; else
    add expected execution time left for t as a constant to expr;
}

addConstraint( list path, constraint type cType) {
  t = first task in path;
  if (cType is "2nd type" and status of t is not either unavailable or published) return ;
  create constraint expression expr;
  if (t is unavailable) add decision variable for booking time of t to expr;
  if (t is published) add predicted time for t to be booked as constant to expr;
  addTermsToExpression(path, expr);
  add optimization constraint [ expr < timeToDeadline ];
}
```

Listing 5.1: Algorithm for creating optimization constraints

5.3.3 Publishing component

A task is published in the platform when the sum of its expected booking time, allotted time, and expected execution time of subsequent activities in the process is almost as current time to deadline. In other words, the time to deadline when it should be published can be determined by (i) finding the longest path from the task node to *out* node in the process graph, where weights of edges are set to the determined optimal values for time allotted of their source nodes, and (ii) adding the booking time of the task to this value. A task can also be updated after being published if it has not been booked yet.

In practice, the time to deadline value which is provided to optimization and publishing components can be lower than the real value in order to keep a small fraction of time reserved for handling unexpected events.

5.4 Evaluation

As we mention in Section 5.5, best to our knowledge there are no similar approaches. Therefore, we were not able to make a comparative evaluation. Because the distinguishing feature of our approach is consideration of competition in crowdsourcing and booking time, we compare the effectiveness of the optimization with cases when booking time is partially of fully neglected. We also empirically prove the optimal choice of task publishing time, and evaluate the overall performance overhead of the optimization component.

To evaluate our approach, we examined a prototype implementation of the framework in a simulated environment. We used MATLAB surface fitting for functional approximation, and GUROBI [60] framework for solving the quadratic optimization problem. We used discrete time model, so time was measured in arbitrary integer units.

5.4.1 Simulation setup

Workers. The size of the simulated crowd size was assumed to be 1000 workers. Platform logs were generated assuming that, generally, at any point only 5% of them are willing to use the crowdsourcing platform (at varios times those can be different workers). For every task type, each worker was assigned two values: the least time allotted that s/he needs to finish a task of this type with weight 1, and the minimum acceptable reward. These values were generated using the normal distribution. Then, for a random sampling of time allotted and reward pairs, 200 log entries were created. To pick a sensible booking time for a log entry, the competition value was calculated as a number of workers from the crowd, whose least time allotted and minimum acceptable reward were less than the log entry had. Then, assuming that only 5% of the potential competitors would actually compete for the task, the booking time estimation was guided by the probability that at least one of competing workers books the task before time n, assuming that the time of booking the task by one worker follows the normal distribution. The actual value was determined by stepwise increasing n and comparing a uniformly distributed random value with this probability. Once it happens that the random value is less than the probability, n is the booking time. Such a method covered coincidental fast bookings while generally exhibiting the trend associated with competition. This method was also used to simulate actual booking while performing experiments, i.e., the crowd simulator was not informed about the booking time chosen.

Tasks. Three types of crowdsourced tasks were simulated. Each task was described by average reward and allotted time, booking time by one worker, and corresponding deviations. The types of tasks and their generation parameters (left) as well as the estimated dependencies between booking time, allotted time, and reward for tasks of Type 1 (right) are shown in Figure 5.6.

Processes. The simulator randomly generated processes with different sizes. We simulated small processes (5-10 tasks) and big processes (10-30 tasks), including all types of crowdsourced tasks and simple activities. We believe that bigger numbers are not realistic in a real setup, because usually business logic is clustered into concise compositions that are then managed on a higher level. Weights for tasks were selected randomly from the range [0.5,5].

Property	Type 1		Type 2		Type 3	
	Avg	Dev	Avg	Dev	Avg	Dev
Reward	100	15	50	7	80	10
Time allotted	20	3	15	3	13	2
Booking time for one worker	30	9	20	8.5	15	5

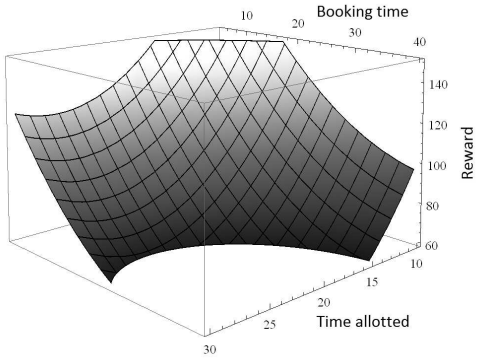


Figure 5.6: Task generation parameters are shown in the table on the left. The figure on the right illustrates the estimated quadratic polynomial that describes the dependencies between booking time, allotted time, and rewards for tasks of Type 1.

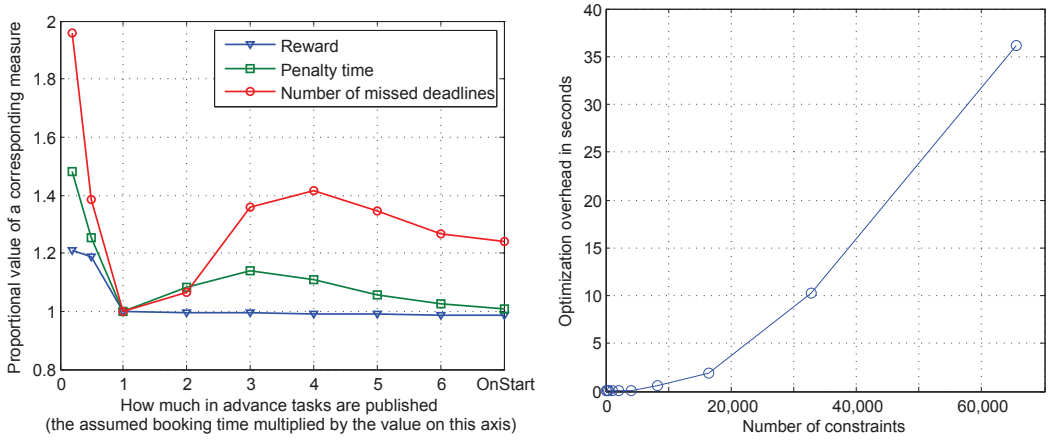
5.4.2 Experiments

We ran the prototype in different simulation settings by randomizing process structures, and by emulating the inaccuracy of task execution and booking times (the results from different random generation seeds were averaged). The actual execution time for tasks was set using normal distribution with deviations of 0.1 and 0.2 of the supposed execution time. We considered both cases when the deadline could and could not be adhered thus exploring how different parameters of the approach impact both critical and not critical situations. We compare the results based on average reward, total time penalty (time penalty is a delay of a process with regard to the deadline for cases where deadline was missed), and number of missed deadlines produces, as these indicators fully reflect the goal of the approach.

Publishing time. In our solution a task is published when the difference between the time left before deadline and the sum of the task's allotted time, and expected execution time of subsequent activities in the process (let us refer to this value as *booking buffer*) is equal to its decided booking time. In order to prove that this is the optimal choice, we performed experiments where tasks were published at earlier and later times. The results are shown in Figure 5.7a. We used booking buffer values equal to 0.2,0.5,1,2,3,4,5,6 multiplied by the task's decided booking time, and, finally, we tested the case when the tasks were published in the platform immediately after a process was started (*OnStart* mark in the figure).

It can be clearly seen that booking buffer equal to the expected (decided) booking time produces the optimal results. When it is lesser than this value, the tasks are not booked in time, so the optimizer has to compensate that by putting higher awards, and, regardless of that, more deadlines are missed because of these delays. When booking buffer is greater than the decided booking time, then there is less room for maneuvering to handle uncertainty in execution and booking times, because tasks become booked earlier and their parameters cannot be changed any more. It results into more missed deadlines and bigger penalties. This behaviour then gradually

changes in an opposite way, which can be explained by the fact that the real booking time can be longer than the estimated one, and therefore the impact of this inaccuracy is reduced for bigger values of booking buffer. However, the number of missed deadlines remains at least 25% greater than in the ultimate case when all the tasks are published when process is started, and rewards and time penalties in this case are almost the same.



(a) Publishing time

(b) Performance overhead

Figure 5.7: Figure (a) shows the effect when the booking buffer is varied and tasks are published not as suggested by our approach (1 on the X axis). Figure (b) shows the dependency of the performance overhead on the number of constraints in the optimization problem for one run.

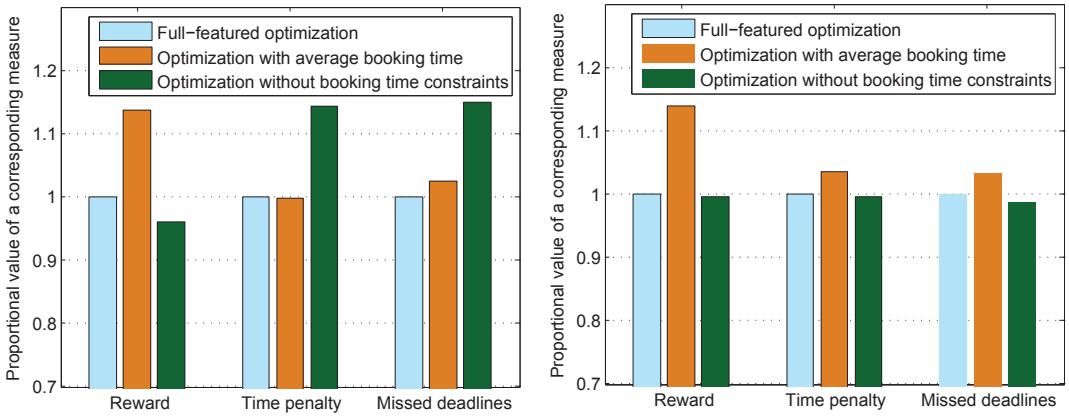
Booking time. Consideration of booking time is a key feature in our approach. To analyze the effectiveness of this feature, we compare the full-featured optimization to the case where booking time constraints (Second type of constraints, see Section 5.3) are completely removed from the optimization problem (tasks are still published at the appropriate time according to the expected booking time which inferred from the decided rewards and allotted times), and to the case where average booking time is chosen for each task. The results for small and big processes are shown in Figures 5.8a and 5.8b respectively, depicting the same indicators as in the previous set of experiments.

As the results show, choosing an average booking time always results in approximately 13% more expenses for all process sizes, and produces more or almost as much penalties and deadline misses as the full-featured optimization does. This happens because some task do not need to be booked fast, and the full-featured optimization would pick less competitive and therefore less expensive values for rewards.

Disabled booking time constraints do not affect the indicators for big processes. This can be explained by the fact that there is almost always enough time for booking the tasks in long processes. Only first tasks of the process can be delayed, but it does not affect the overall performance. Also, booking times are always chosen to be maximal in this case because they are not constrained and it reduces the paid rewards. However, for smaller processes, the consideration

of booking time is crucial, as it can stronger affect the relatively short process execution time. The unconstrained case produces 14% more deadline misses and penalties.

One can argue that the full-featured optimization should perform at least with the same performance as two other approaches. In perfect conditions this assertion would hold. However, on the one hand, the booking time is estimated as an *upper bound*, therefore, the booking can often take less time than predicted. On the other hand, when the competition is too low, it can have the opposite effect: a task, which is assumed to be booked and executed earlier and costs more, can be eventually delayed more than a task which costs less and was expected to be booked later. Therefore, estimating realistic booking times is one of the key requirements of this approach. The accuracy of booking time in our experiments was 92%, which resulted in at most 2% of more missed deadlines and penalties.



(a) Small processes (5-10 tasks)

(b) Big processes (10-30 tasks)

Figure 5.8: Effect of neglecting the booking time in optimization

Performance overhead. The overhead of optimization depends on the number of variables and the number of constraints. However, the number of variables for our problem is proportional to the number of tasks involved, which was less than 30, and the variance within this limit did not affect the overhead. However, the number of constraints is proportional to the total amount of all possible paths that go through *in* and *out* activities (See Section 5.3), and this number depends on the process structure and scales from 1 to tens of thousands. Figure 5.7b depicts the overhead dependency on the number of constraints for one optimization run[1].

For small processes (up to 10 tasks), the worst case is when there is a sequence of 5 constructs each with 2 activities in parallel, the number constraints is less than $2^5 * 2 = 64$, which implies that an optimization run for a small process always takes less than 0.01 second. For bigger processes, the worst case is $2^{15} * 2 = 65535$, so an optimization run can take up to 35 seconds in this case. In both cases, the overhead is acceptable for performing periodical adaptations in processes with human tasks which can span from several minutes to hours or days.

[1]Hardware used: Intel Core 2 Quad 2.40 GhZ with 6 GB of RAM

5.4.3 Discussion

The results clearly show that booking time should be considered when publishing tasks to achieve the best adaptable behavior, because the best results are achieved when the booking buffer equals to the estimated expected booking time. Booking time constraints for optimization are however not always favourable. They should not be used for bigger processes (more than 10 tasks in out setting), but become important for smaller processes. Such smaller processes can emerge in, e.g., agile software development environments, where work is organized into short cycles with small sets of tasks.

5.5 Related Work

Major industry players have been working towards standardized protocols and languages for interfacing with people in a service oriented way, which may be used as technical foundation for implementing our ideas in real businesses. Specifications such as WS-HumanTask [24] and BPEL4People [4] have been defined to address the lack of human interactions in service-oriented businesses [49]. Although some prospective features for dynamic resource management were outlined for these standards [74], however, they have been designed to model interactions in closed enterprise environments where people have predefined, mostly static, roles and responsibilities.

The area of QoS-aware composition of Web services has many similarities to the topics addressed in this work. Web service compositions create value added services by composing existing ones. Here the question arises which services to chose for participation in a composite service, given that there are many available Web services providing equivalent functionality. Liangzhao Zeng et al. [93] propose a QoS-aware middleware for selecting Web services that maximize user satisfaction modeled as utility functions. They define multiple quality criteria, i.e., execution price, execution duration, reputation, successful execution rate, and availability. The authors propose service selection based on local optimization and global selection, considering aforementioned quality criteria. In the local optimization case service selection is done for each task individually, while the global planning also considers the interrelations between services. Integer Programming is used to solve the global planning problem. Canfora et al. [8] argue that genetic algorithms, while being slower than integer programming, represent an alternative, more scalable option. Another work [73] focuses on the optimization of large-scale QoS-aware compositions at runtime based on QoS specification based on constraint hierarchies. Multiple well-known metaheuristic optimization approaches are applied to solve the optimization problems. The major difference between Web service composition and crowdsourcing of business processes is that Web services which are to execute a functionality can be directly chosen while humans in the crowd are self-determined and act autonomously.

The recent trend towards *collective intelligence* and crowdsourcing can be observed by looking at the success of various Web-based platforms that have attracted a huge number of users. Well known representatives of crowdsourcing platforms include Yahoo! Answers [88] (YA) and the aforementioned AMT [5]. The difference between these platforms lies in how the labor of the crowd is used. YA, for instance, is mainly based on interactions between members. Ques-

tions are asked and answered by humans, thereby lacking the ability to automatically control the execution of tasks. In contrast, AMT offers access to the largest number of crowdsourcing workers. With their notion of HITs that can be created using a Web service-based interface they are closely related to our aim of mediating the capabilities of crowds to service-oriented business environments. According to one of the latest analysis of AMT [35], HIT topics include, first of all, transcription, classification, and categorizations tasks for documents and images. Furthermore, there is also tasks for collecting data, image tagging, and feedback or advice on different subjects.

While this work focuses on how to take a workflow and optimally convert the subtasks into crowdsourcing tasks, there is also research about how to map crowdsourcing tasks to suitable workers. A possibility is to use auctioning mechanisms for implementing such a mapping [75]. All workers that meet the minimum requirements for a particular are invited to submit a bid to an auction created for assigning the task. The winning bid is determined by a combination of the workers suitability for the task and the bid's price. For improved reliability and training of workers a single task may be crowdsourced multiple times.

5.6 Summary

This chapter presented an approach for deadline-driven adaptive execution of business processes on top of an internal competition-based crowdsourcing platform. The main feature that distinguishes our approach from other workflow and process optimization methods is consideration of time that it takes a crowd to book a task. We proposed a method for estimating the functional dependency of booking time by using statistical data, presented an algorithm for constructing an optimization problem, and empirically determined the optimal task publishing technique.

The results show that our model is effective for adapting the properties of tasks in a crowdsourcing platform to adhere to process deadlines and to minimize the rewards. We discovered that booking time should be considered when publishing tasks to achieve the best adaptable behavior, and that taking booking time into account in optimization can reduce the deadline misses up to 14%. The approach can also be used to predict feasibility and expenses for a specified deadline by running a simulation like ours, therefore allowing to explicitly observe the tradeoff between processing time and associated costs.

CHAPTER 6

Private and Confidential Data Propagation Control in SOA

Law regulations, corporate standards, and desire for customer satisfaction require enterprises to provide certain guarantees for the data obtained from services that are offered to customers or partners. As data integration proliferates, private or confidential information can be spread across the system extensively. Striving for protection of possessed sensitive information, enterprises thus need comprehensive means to control such propagation. In this chapter we propose a private data propagation control framework (Providence), which aims to give a comprehensive view on private or confidential data usage throughout the system and to facilitate decision making regarding the appropriate security-related SLOs for offered services and internal policies for this data.

6.1 Overview

The proliferation of data integration intensifies the propagation of data throughout the enterprise, since the result of one activity can serve as a source for another. The more sources are involved, the easier it is to overlook the inappropriate use of data, as it comes to be maintained in various locations by different parties. Indeed, proliferation of resource virtualization and cloud computing requires even more delicate consideration of privacy concerns [65, 90]. Thus, striving for protection of possessed sensitive information, enterprises need comprehensive means of control over its propagation.

We address the problem in the general case of SOA where actual implementation of services is inaccessible and their functionality is unrestricted. This implies, for example, that data might be stored by one service and retrieved or transferred later by another service, so this fact can't be established by workflow analysis. To our best knowledge, there are no solutions to date that address this problem.

This chapter presents the *Pri*vate *D*ata Propagatio*n* *C*ontrol Fram*e*work (Providence). Monitoring the message exchange, it employs content inspection that is successfully applied in Data Loss Prevention (DLP) solutions[1]. The framework detects and logs private data disclosures that happen in the SOA environment. The log is then used to give a comprehensive view on private data usage throughout the SOA and to facilitate privacy-related decision making. The rationale behind the framework is to have a practically-applicable solution that requires as few integration efforts as possible.

6.2 Motivating Scenario

[86] outlines the scenario with harmful inappropriate use of private data, where Alice, who has a child with a severe chronic illness, buys a book about the disease from online book store. Later she applies for a job and gets rejected because the employer somehow received the information about her purchase and flagged her as high risk for expensive family health costs.

Below we show how the aforementioned scenario can take place due to inappropriate control over the private data in manifold data integration activities in SOAs.

Consider an enterprise Things'n'Books, which runs several businesses including online book trading. It has complex infrastructure with various departments, each responsible for a particular business. Things'n'Books also has a delivery department handling all delivery needs of a company and a joint human resources department. To achieve interoperability between departments, Things'n'Books employs SOA based on Web services.

When the client orders books on-line, the order details are put into orders database exposed as a data service (OrderService). Once the payment is received, a delivery workflow, implemented in BPEL, is initiated. It takes care of order preparation and eventually calls the Web service exposed by the delivery department (DeliveryService) which takes delivery items and the recipient contact data on input.

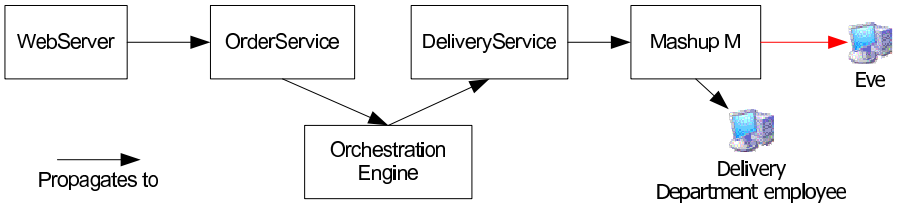


Figure 6.1: Private data propagation scenario

The delivery department, in time, besides using the received data to perform delivery, stores this data for both accounting reasons and further usage in delivery routes analysis and optimization through the instrumentality of an enterprise mashup *M*. The human resources department also uses an enterprise mashup platform and benefits from partial re-use of mashup *M*. Also, it outsources some duties to an independent HR company, and a representative of that company, *Eve*, works in Things'n'Books's office.

[1] `http://www.cio.com/article/498166/Best_Data_Loss_Prevention_Tools`

Thus, if Alice purchases a book from Things'n'Books's, Eve has an opportunity to access this information via the mashup *M* having no access to its origin source, the online orders database. The propagation chain is illustrated on Figure 6.1.

6.3 Providence Framework

In this section we introduce the Providence framework. We show the architecture of the framework, give formal specifications for control paradigms and replaceable components, specify implementation remarks, and outline limitations of the framework.

6.3.1 Architecture

We start from specifying required adoptions in SOA, then we describe each component of the framework. The design of the framework is illustrated in Figure 6.2. The framework demands two adoptions to be performed in SOA infrastructure, requiring it to:

1. Submit private data entries to the registrator service of the framework, whenever such entries appear in the system (e.g., when private data is entered manually, received from a partner organization, or submitted to a service in SOA).

2. Intercept messages exchanged in the integration and submit them together with the context information to the monitor service of the framework for inspection (e.g., SOAP messages that travel on Enterprise Service Bus). Context information enables the framework to distinguish between different integration activities. Examples of context elements are shown in Table 6.1.

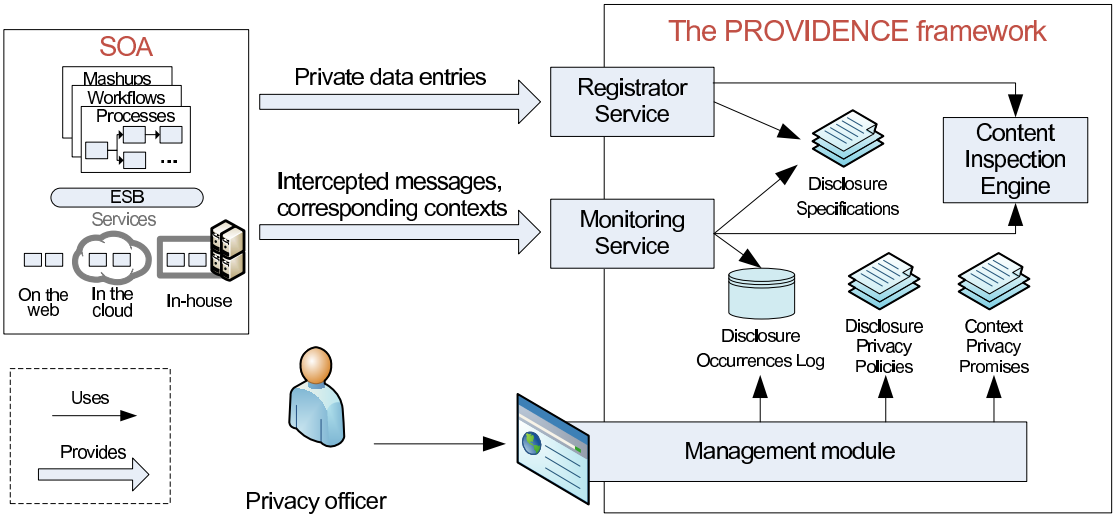


Figure 6.2: The Providence framework

Table 6.1: Examples of context elements

Level	Context element
Network	Requestor host Responding endpoint
Application	Consuming application's identifier Requestor's credentials from responder's perspective Requestor's credentials in requestor application, e.g. in mashup platform
Process	Corresponding process identifier, e.g. in business process engine

Table 6.2: Examples of primitive types

Type	Value	Possible detectable forms
Name	John Johnson	John Johnson, J. Johnson, JOHNSON J
Date	02.01.2010	02/01/10, 1 Feb 10
Amount	50000	50 000, 50.000

Registrator Service. Private data entries are submitted to the Registrator Service in form of *disclosures*. Each disclosure contains typified pieces of private data, *primitives*, and a logical rule, whose variables correspond to those primitives. Also, each disclosure has a type. For example, a disclosure of private information about a 50000$ bank loan issued to John Johnson who lives at 1040 Paulanergasse 2 can be specified as primitives p_1 of type $Name$ which has value "`John Johnson`", p_2 of type *Address* which has value "`1040 Paulanergasse 2`", and p_3 of type *Amount* which has value `50000`, with rule $(p_1\ or\ p_2)\ and\ p_3$ and type "`PersonalLoan`". It means that if the message exchange contains the amount together with either address or name, possibly in the different form (e.g., "`J. Johnson`"), then the disclosure occurs. The type of primitive indicates how its form can vary. When a disclosure is submitted, primitives are separated from rule and group identifier, so only the mapping is kept. The rule and group identifier are stored to Disclosure Specification Repository, the primitives are registered at the Content Inspection Engine.

Monitoring Service. The monitoring service receives messages exchanged in data integration processes and corresponding context information. The messages are forwarded to the Content Inspection Engine that detects previously registered primitives in messages' content. If any primitives are detected, the corresponding disclosures are retrieved from Disclosure Specifications Repository and their rules are checked against detected primitives. Occurred disclosures are logged together with the context.

Content Inspection Engine. The Content Inspection Engine is responsible for detecting primitives in the messages content. It receives primitives from Registration Service and content for inspection from the Monitoring Service. The type of a primitive defines the possible transformations of its data value. Examples of such types are shown in Table 6.2. There are various content inspection techniques and algorithms [52, 78] whose explanation and evaluation is beyond the scope of this thesis.

Management Module. This is the core component of the framework. Using the information from disclosure occurrences log, this module provides means to control private data propagation to the privacy officer. She can (i) assign privacy promises to contexts according to real data usage practices in those contexts, (ii) assign privacy policies to disclosure types, and, based on actual disclosure occurrences log and assigned policies, (iii) get answers to following types of questions:

1. Which privacy policy violations happened?
2. Which disclosures happen in specified context?
3. In which contexts disclosure of specified type happens?
4. What promise is enough for specified context to keep compliant with current private data usage practices?
5. How is the private data of specified type actually used?
6. How was the particular piece of private information used?
7. What if we want to set another policy for private data or context, what violations will it produce for the current environment?

The explanation of corresponding answers is given in the next section.

6.3.2 Formal Specifications

The framework is not coupled to a particular policy and context specification. Nevertheless, the framework's logic heavily relies on these components. Therefore, we specify formal requirements for these components and formally explain the logic of the management module as follows:

Policy. We assume that both private data policies and context promises are expressed in the same form. The policy specification should reflect both requirements (policies) and promises for data usage. It can encapsulate allowed usage purposes, receiving parties, and required obligations. The implementation must provide a means to check whether data treatment under one policy satisfies another policy (promise) and to calculate the intersection and union of policies.

Formally, let P - set of all possible policies. There must be defined relation $satisfies$ on P which we mark as $\leftharpoondown$, so that $p_1 \leftharpoondown p_2; p_1, p_2 \in P$ indicates that data treatment under policy p_1 satisfies policy p_2.

Further, there must be defined union operator $\cup : P \times P \rightarrow P$ that associates two policies with a policy that is satisfied by either of them: $\forall p_1, p_2 \in P \; p_1 \leftharpoondown (p_1 \cup p_2), p_2 \leftharpoondown (p_1 \cup p_2)$.

Finally, there must be defined intersection operator $\cap : P \times P \rightarrow P$ that associates two policies with a policy that satisfies both of them: $\forall p_1, p_2 \in P \; (p_1 \cap p_2) \leftharpoondown p_1, (p_1 \cap p_2) \leftharpoondown p_2$.

Context. Context specification should distinguish activities that happen in SOAs in a way that it is possible to assign feasible privacy promises to those activities. The implementation must provide a means to check whether one context is a subcontext of another. Occurrence of a

disclosure in some context implies its occurrence in all contexts, of which the given context is a subcontext.

Formally, let C - set of all possible contexts. There must be defined relation $\subseteq$ on C, so that $c_1 \subseteq c_2; c_1, c_2 \in C$ indicates that c_1 is a subcontext of c_2.

Configuration. The privacy officer manages context promises and private data policies. Formally, we can consider that for a given point in time there is *configuration*, which contains these mappings. As policy or promise might be unassigned, we introduce unassigned policy Ω, and extend P to $P' = P \cup \{\Omega\}$. The rationale of unassigned policy is passivity in computations, thus we refine $\leftharpoondown, \cap, \cup$ for P' as follows:

$$\Omega \leftharpoondown \Omega, \Omega \cap \Omega = \Omega, \Omega \cup \Omega = \Omega;$$

$$\forall p \in P \Rightarrow \Omega \leftharpoondown p, p \leftharpoondown \Omega;$$

$$\forall p \in P \Rightarrow \Omega \cap p = p, p \cap \Omega = p;$$

$$\forall p \in P \Rightarrow \Omega \cup p = p, p \cup \Omega = p.$$

Configuration is a tuple ($Promise$, $Policy$), where $Promise : C \rightarrow P'$, $Policy : T \rightarrow P'$, where T is a set of all disclosure types. $Promise$ maps contexts to privacy promises assigned, $Policy$ maps disclosure types to privacy polices assigned. For any context, its promise must always satisfy a promise of any its subcontext: $\forall c, c' \in C : c' \subseteq c \Rightarrow Promise(c) \leftharpoondown Promise(c')$.

Management Actions Logic. Based on the current configuration and a part of the log (e.g., log for last month), management module enables the privacy officer to get answers on question templates.

Formally, let $L = \{\langle c_i, d_i \rangle\} : c_i \in C, d_i \in D, 1 \leq i \leq N$ - disclosure occurrences log, where D is a set of all registered disclosures, N is number of log records, d_i is a disclosure and c_i is a context that corresponds to log entry i. Let $Type(d), d \in D$ indicate the type of disclosure d. Now, given current configuration $\Lambda = (Promise, Policy)$, for any part of log $L' \subset L$ answers can be given as specified in Table 6.3:

6.3.3 Implementation

The prototype of Providence framework was designed and implemented considering separation of concerns. The core components implement the principal logic using unified interfaces to replaceable components that are responsible for content inspection and privacy policies storing. Replaceable components were implemented in an elementary way to demonstrate functionality of the framework. The prototype's work is demonstrated in the screencast[2]. The evaluation of results is given in Section 6.5.

The success of the framework will depend on the selection of particular adoption points and replaceable components. The context elements should be chosen in a way that (i) disclosures that occur during the same data integration activity share as much contexts as possible, and (ii) disclosures that occur during different activities share as few contexts as possible.

[2] `http://www.infosys.tuwien.ac.at/prototype/Providence/`

Table 6.3: Privacy-related questions and answers

Question	**Answer**
1. Which privacy policy violations happened?	$\langle c, d\rangle \in L' :$ $Promise(c) \not\leftarrow Policy(Type(d))$
2. Which disclosures happen in context c	$d : c' \subseteq c, \langle c', d\rangle \in L'$
3. In which contexts disclosure of type T happens?	$c : Type(d) = T, \langle c, d\rangle \in L'$
4. What promise p is enough for context c to keep compliant with current private data uses?	$p = \bigcap_{c' \subseteq c, \langle c', d\rangle \in L'} Policy(Type(d))$
5. What policy p reflects the actual usage of private data in disclosures of type T?	$p = \bigcup_{Type(d)=T, \langle c,d\rangle \in L'} Promise(c)$
6. What policy p reflects the actual usage of private data in disclosure d?	$p = \bigcup_{\langle c,d\rangle \in L'} Promise(c)$
7. If we want to change the configuration, what violations will the new configuration Λ' produce?	(1.) for configuration Λ'.

6.3.4 Limitations

Several limitations of the framework can be foreseen:

- Context elements must be accessible by message submitting adoption point. However, it is a technical matter.

- Content inspection might give false negatives for complex data integration patterns. False positives might occur as well. The balance between false positives, false negatives, and the detection rate in general will always depend on particular information systems, business specifics, IT architecture and inspection engine settings. Therefore, to estimate these rates, it is necessary to test the approach in real business environments. However, fine-tuned DLP tools provide 80-90% accuracy and almost no false positives in commercial product tests[3].

- It is impossible to obtain the data for inspection from encrypted messages. However, the adoption points can be selected in such a way, that data will be sent to inspection before encoding. As the monitoring service does not forward the data any further and is supposed to be deployed in-house, this should not raise any security issues.

- Content inspection of internally exchanged messages brings computational overhead in the system. However, submitted messages can be analyzed asynchronously using dedicated resources, neither interrupting nor slowing down business processes. We thus regard the framework's performance as a secondary concern.

[3] `http://www.cio.com/article/498166/Best_Data_Loss_Prevention_Tools`

- The privacy officer will need to obtain actual context privacy promises. However, it is inevitable due to assumption of inaccessibility of service implementations. Moreover, given the solution that is able to conclude about actual treatment of data by a service (e.g., using source code analysis), it can be coupled with the framework, thus automating context promises assignment.

6.4 Related Work

To the best of our knowledge, none of current research fully addresses the issue of private data propagation control within the organization SOAs.

Works like [6, 37, 54] eventually come to rule-based access restriction to private data. However, none of these solutions can control the usage of private data *after* it gets released from the guarded resource.

Approaches like [14, 27, 50] address the issue only within bounds of processes or workflows, whereas integration techniques go beyond this paradigm. For instance, services can call each other independently of the workflow engine, or they can store private data, so it can be later picked up by other processes.

[87] proposes a framework for building privacy-conscious composite Web services which confronts service consumer's privacy preferences with component services' policies. This framework does not address scenarios where private data is firstly stored by a component service and then retrieved outside of the composite service's scope, whereas such scenarios are inevitable for data integration.

[72] proposes to record and store logs of private data usage together with this data throughout its lifespan. It requires services, which consume private data, to record all actions performed with this data to the log, and return this log together with the response. Such logs can then be analyzed to conclude about appropriate use of data. Unlike our framework, this approach interferes with the business logic of services, requiring them to report performed actions. Therewith, it does not consider cases, when private data is stored by one service and then retrieved and used by another.

[55] proposes to encrypt the messages that are exchanged between enterprises, so that a request to the trusted authority must be performed to decrypt them. Using this technique, trusted authority can check privacy promises of enterprises and control the access to the private data. This technique neither applies to private data control within the enterprise, nor addresses the usage of data once access to it was granted.

[68] provides tool support for solving a number of problems in data usage control. However, the authors rely on their own model of a distributed system [67] which assumes that each concerned peer has a secure data store that possesses all the private data available to this peer, and that the peer routes this data through the usage control mechanisms whenever the data leaves this store. Such assumption makes the work inapplicable for SOAs in the general case.

[7] addresses similar problems of private data misuses. The authors build their work upon the Private Data Use Flow expressed as a state machine. However, they give neither instructions nor any explanations of how to build this state machine in an automated fashion having a live system. Unlike it, our approach was designed to work in real SOAs.

Data Loss Prevention solutions are different to our approach in a way that they protect the system from casual leakages of private data from end users, whereas our approach aimed to detect systematic inappropriate uses of data which ensue from SOA complexity.

6.5 Evaluation

To evaluate our work, we emulated (created and deployed) the SOA environment and business logic form the scenario in Section 6.2. Genesis 2 testbed [36] has been used for emulation. By using its service interceptors we were able to monitor the data exchange and deliver the disclosures to the prototype for inspection. Besides elements and logic outlined in Section 6.2, the testing environment included printery department which executes private orders and employs the delivery department for shipping. After performing the business logics simulation, we tested the management module's functionality to proof the concept.

The business logic was executed as enumerated below. Figure 6.3 depicts the testing environment and log records inserted during the emulation.

1. Two online book orders are made, thus two *BooksPurchase* disclosures are registered in the framework.

2. *PlaceOrder* operation of *OrderService* is called for each purchase, disclosures are detected, and log records 0,1 are generated.

3. For each purchase orchestration engine retrieves details via *GetOrder* operation of *OrderService* (log records 2,4) and later submits it to *Deliver* operation of *DeliveryService* (log records 3,5).

4. Delivery department uses *MashupService* to run the mashup, which features *BooksPurchase* disclosures (log records 6,7).

5. HR department accesses private information via *OrderService* (log record 8), *DeliveryService* (log records 9,10), and *MashupService* (log records 11,12)

6. Printery department composes private printery order, thus a *PrinteryOrder* disclosure is registered in the framework.

7. The order details are submitted to the *Deliver* operation of *DeliveryService* thus generating log record 13.

8. Delivery department runs the mashup which now involves printery order (log record 14).

Disclosure type policies and context promises of the testing environment are shown in Figure 6.4. Data from *BooksPurchase* disclosures is allowed to be used for system administration and research and development; *PrinteryOrder*'s policy is not assigned. Data submitted to *OrchestrationHost* within bounds of *OrderProcess* is known to be used for system administration and individual analysis. Data propagated to *DeliveryServiceHost* is known to be used for historical preservation. If data is sent to *MashupHost* and remote login is *DeliveryWorker*, then

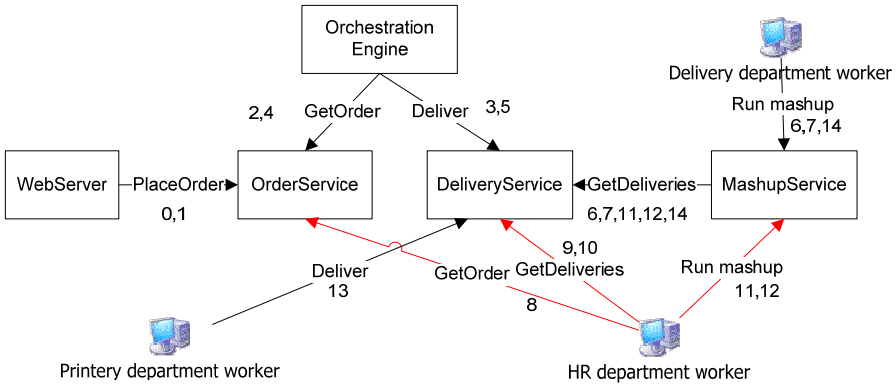


Figure 6.3: Testing environment

Disclosure type	Policy
BooksPurchase	Self=[admin, develop]
PrinteryOrder	Unassigned

Context	Promise
process = OrderProcess destinationHost = OrchestrationHost	Self=[admin, individual-decision]
destinationHost = DeliveryServiceHost	Self=[historical]
destinationLogin = DeliveryWorker destinationHost = MashupHost	Self=[develop]
destinationHost = HRDepartmentHost	ThirdParty=[other]
destinationLogin = HRWorker destinationHost = MashupHost	ThirdParty=[other]

Figure 6.4: Privacy policies and promises in testing environment

it is used for research and development. Data propagated to *HRDepartmentHost* or retrieved by *MashupHost* with remote login *HRWorker* is known to be exchanged with third party for undefined purposes.

Table 6.4 shows the output of the framework's management module for violation detection. Having discovered such violations, privacy officer can decide to strengthen context promise (e.g., as in case of orchestration engine, log records 2,4), loose private data policy (e.g., as in case of delivery service, log records 3,5) or assume administrative measures to prevent similar access to private data (e.g., in case of HR worker, log records 8,9,10,11,12).

The management module is able to give answers on question templates from Section 6.3.2, such as itemizing disclosure types that take place in a context (e.g., `BooksPurchase`, `PrinteryOrder` for the context `destinationHost = DeliveryServiceHost`), itemizing the contexts, in which disclosure of particular type happens (e.g., `PrinteryOrder` takes place in two contexts, see Figure 6.5), or inferring the actual policy for particular disclosure type (e.g., `Self=[historical, develop]` for `PrinteryOrder`).

Thus, the framework explicitly detects any disclosures that happen during the data integra-

Table 6.4: Detected violations

Record N	Context	Policies
2,4	process = OrderProcess host = OrderServiceHost destinationHost = OrchestrationHost	Private data policy:{Self=[admin, develop]} Context promise:{Self=[admin, individual-decision]}
3,5	process = OrderProcess host = OrchestrationHost destinationHost = DeliveryServiceHost	Private data policy:{Self=[admin, develop]} Context promise:{Self=[historical]}
8	host = OrderServiceHost destinationHost = HRDepartmentHost	Private data policy:{Self=[admin, develop]} Context promise:{ThirdParty=[other]}
9,10	host = DeliveryServiceHost destinationHost = HRDepartmentHost	Private data policy:{Self=[admin, develop]} Context promise:{ThirdParty=[other]}
11,12	hostLogin = MashupService destinationLogin = HRWorker host = DeliveryServiceHost destinationHost = MashupHost	Private data policy:{Self=[admin, develop]} Context promise:{ThirdParty=[other]}

host	= PrinteryDepartmentHost
destinationHost	= DeliveryServiceHost

hostLogin	= MashupService
destinationLogin	= DeliveryWorker
host	= DeliveryServiceHost
destinationHost	= MashupHost

Figure 6.5: Contexts where PrinteryOrder disclosure takes place

tion and helps to prevent the inappropriate access to the data even after it is propagated throughout the system.

6.6 Summary

This chapter discussed the challenges of ensuring security-related SLOs and controlling the propagation of sensitive data in SOAs. We introduced the Providence framework which is able to give a comprehensive view of private data usage throughout the enterprise and to facilitate privacy-related decision making. It can be used to infer the feasible guarantees for consumers of the corresponding services, and reduces the impact of human factors, as it is able detect inappropriate uses of data initiated by actors who disregard the data usage policies. It also helps to detect particular reasons of violations by maintaining the detailed context information, and facilitates service change management by providing means for “what-if” analysis. All the considered limitations regarding the adoption of the framework in SOA are mostly technical and do not prevent it from being implemented and deployed in a real system.

CHAPTER 7

Conclusions and Future Research Directions

This thesis has introduced models, algorithms, and architectures for SLA provision in service-oriented environments with the focus on integrated human labour and human factors that can affect the performance in such systems. Proposed approaches enable the adaptive behaviour in service-oriented systems to cope with SLAs under unstable conditions such as flexible and unanticipated demand, unexpected failures, and poorly predictable human behaviour in open systems. The considered cases includes both internal provision of services within a company, and provision of services to partners or consumers. The thesis discussed provider's and consumer's perspectives on these problems, and proposed the enhancements for state-of-art technologies such as business process orchestration engines and crowdsourcing platforms.

The evaluation has been done through simulation of corresponding systems and components. All the experiments in the thesis indicated strong potentiality of the proposed approaches. Although the proposed enhancements require certain software engineering and organizational effort, no insurmountable obstacles are foreseen for them to be applied in practice. This thesis can therefore serve as a reference for implementing these ideas in industry.

7.1 Future research directions

The work in this thesis allows to make a step forward towards autonomous SLA provisioning, seamless integration of human labour into automated processes, minimization of human factor impact and associated risks and overheads, and achievement of high reliability in mixed systems. There is, however, a potential for enhancement of presented methods. The following specific directions can be highlighted:

- The proposed methods can be complemented with work from the area of workforce optimization to take into account satisfaction factors and consideration of social and psychological effects impact. For example, scheduled crowdsourcing can aim to assign jobs

of the same type to a particular worker for some period, so it gets easier for him/her to concentrate, but to change the task type once in a while, so his/her activity can be less tedious and monotonous.

- The quality of externally provided crowdsourcing services can be further improved by designing more agile control mechanisms that allow to bring together suitable workers and requesters with guaranteed minimized temporal overhead. Having enough participants, such systems can enable *near real-time* crowdsourcing which could open entirely new possibilities for this market. From the perspective of workers, these platforms can be improved by taking into account personal and professional preferences and supply the workers with tasks in more personalized and convenient manner.

- Internal crowdsourcing has a high potential for flexible organization of work inside the company. To improve the adaptation capabilities of such systems, standardized policies for assignment cancellation and re-negotiation should be developed, and the dependency of competition among workers on task parameters should be studied in more detail. These improvements would allow to create more sophisticated optimization and control mechanisms on top of such platforms, which in turn could be better tailored to business requirements.

- Privacy and security-related objectives depend not only on the underlying infrastructure of the provided service, but on the entire system. Therefore, all parts of the system that can access the data of that service, should be aware of the policies applied. While the presented approach allows for detection of such violations, it does not prevent them. This can be achieved by enhancing system design tools (e.g., tools for application and data integration and business process design) with SLA control mechanisms at design time.

Bibliography

[1] C. Adams. Managing crowdsourcing assignments. *Cutter IT Journal*, 24(6):6–11, 2011.

[2] Carl Adams and I. Ramos. The past, present and future of social networking and outsourcing: impact on theory and practice. In *UK Academy for Information Systems Conference Proceedings 2010: Information Systems: past, present and looking to the future*, 2010.

[3] Eugene Agichtein, Carlos Castillo, Debora Donato, Aristides Gionis, and Gilad Mishne. Finding high-quality content in social media. In *WSDM '08*, pages 183–194. ACM, 2008.

[4] A. Agrawal et al. WS-BPEL Extension for People (BPEL4People)., 2007.

[5] Amazon Mechnical Turk. http://www.mturk.com. Accessed: 20.06.2012.

[6] Paul Ashley, Satoshi Hada, Günter Karjoth, and Matthias Schunter. E-p3p privacy policies and privacy authorization. In Sushil Jajodia and Pierangela Samarati, editors, *WPES*, pages 103–109. ACM, 2002.

[7] S. Benbernou, H. Meziane, and M. Hacid. Run-time monitoring for privacy-agreement compliance. In Bernd Kraemer, Kwei-Jay Lin, and Priya Narasimhan, editors, *Service-Oriented Computing – ICSOC 2007*, volume 4749 of *LNCS*, pages 353–364. Springer Berlin / Heidelberg, 2010.

[8] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. An approach for qos-aware service composition based on genetic algorithms. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 1069–1075, New York, NY, USA, 2005. ACM.

[9] Cinzia Cappiello, Marco Comuzzi, and Pierluigi Plebani. On automated generation of web service level agreements. In *Proceedings of the 19th international conference on Advanced information systems engineering*, CAiSE'07, pages 264–278, Berlin, Heidelberg, 2007. Springer-Verlag.

[10] Alberto Caprara, Michele Monaci, and Paolo Toth. Models and algorithms for a staff scheduling problem. *Math. Program.*, 98(1-3):445–476, 2003.

[11] Claudio Castellano, Santo Fortunato, and Vittorio Loreto. Statistical physics of social dynamics. *Reviews of Modern Physics*, 81(2):591–646, May 2009.

[12] Y.K. Che. Design competition through multidimensional auctions. *The RAND Journal of Economics*, 24(4):668–680, 1993.

[13] Jen-Hsiang Chen, R. Anane, Kuo-Ming Chao, and N. Godwin. Architecture of an agent-based negotiation mechanism. In *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, pages 379 – 384, 2002.

[14] W.K. Cheung and Y. Gil. Towards privacy aware data analysis workflows for e-science. In *AAAI Workshop on Semantic e-Science 2007*, pages 22–26, Menlo Park, CA, USA, 2007. AAAI Press.

[15] CloudCrowd. https://www.cloudcrowd.com/. Accessed: 20.06.2012.

[16] Marco Comuzzi and Barbara Pernici. A framework for qos-based web service contracting. *ACM Trans. Web*, 3(3):10:1–10:52, July 2009.

[17] Crowdflower. http://crowdflower.com/. Accessed: 20.06.2012.

[18] Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54:86–96, April 2011.

[19] Thomas Erl. *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007.

[20] 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*, 153(1):3 – 27, 2004. Timetabling and Rostering.

[21] A.T. Ernst, H. Jiang, M. Krishnamoorthy, B. Owens, and D. Sier. An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 127:21–144, 2004. 10.1023/B:ANOR.0000019087.46656.e2.

[22] M. Ferber, S. Hunold, and T. Rauber. Load balancing concurrent bpel processes by dynamic selection of web service endpoints. In *Parallel Processing Workshops, 2009. ICPPW '09. International Conference on*, pages 290–297, 2009.

[23] QoS for Web Services: Requirements and Possible Approaches. http://www.w3c.or.kr/kr-office/tr/2003/ws-qos/. Accessed: 20.06.2012.

[24] Mark Ford et al. Web Services Human Task (WS-HumanTask), Version 1.0., 2007.

[25] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[26] Ganna Frankova, Magali Séguran, Florian Gilcher, Slim Trabelsi, Jörg Dörflinger, and Marco Aiello. Deriving business processes with service level agreements from early requirements. *Journal of Systems and Software*, 84(8):1351–1363, 2011.

[27] Y. Gil and C. Fritz. Reasoning about the appropriate use of private data through computational workflows. In *Spring Symposium on Intelligent Privacy Management*. AAAI Press, 2010.

[28] Ignacio Grossmann. Enterprise-wide optimization: A new frontier in process systems engineering. *AIChE Journal*, 51(7):1846–1857, 2005.

[29] Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[30] Mamoun Hirzalla, Peter Bahrs, Jane Cleland-Huang, Craig Miller, and Rob High. A predictive business agility model for service oriented architectures. In Gerti Kappel, Zakaria Maamar, and Hamid Motahari-Nezhad, editors, *Service-Oriented Computing*, volume 7084 of *Lecture Notes in Computer Science*, pages 653–660. Springer Berlin / Heidelberg, 2011.

[31] J. Howe. *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*. Crown Business, 2008.

[32] P.C.K. Hung, Haifei Li, and Jun-Jang Jeng. Ws-negotiation: an overview of research issues. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, page 10, jan. 2004.

[33] Crowdsourcing in IBM. http://personneltoday.com/articles/article.aspx?liarticleid=55343. Accessed: 20.06.2012.

[34] P. Ipeirotis. Demographics of mechanical turk. *New York University, Tech. Rep*, 2010.

[35] Panagiotis G. Ipeirotis. Analyzing the Amazon Mechanical Turk Marketplace. *SSRN eLibrary*, 17(2):16–21, 2010.

[36] Lukasz Juszczyk and Schahram Dustdar. Script-based generation of dynamic testbeds for soa. In *ICWS*, pages 195–202. IEEE Computer Society, 2010.

[37] Guenter Karjoth, Matthias Schunter, and Michael Waidner. Privacy-enabled services for enterprises. In *DEXA Workshops*, pages 483–487. IEEE Computer Society, 2002.

[38] Ehud Karnin, Eugene Walach, and Tal Drory. Crowdsourcing in the document processing practice. In Florian Daniel and Federico Facca, editors, *Current Trends in Web Engineering*, volume 6385 of *Lecture Notes in Computer Science*, pages 408–411. Springer Berlin / Heidelberg, 2010.

[39] Robert Kern, Hans Thies, and Gerhard Satzger. Statistical quality control for human-based electronic services. In Paul Maglio, Mathias Weske, Jian Yang, and Marcelo Fantinato, editors, *Service-Oriented Computing*, volume 6470 of *Lecture Notes in Computer Science*, pages 243–257. Springer Berlin / Heidelberg, 2010.

[40] Robert Kern, Christian Zirpins, and Sudhir Agarwal. Managing quality of human-based eservices. In George Feuerlicht and Winfried Lamersdorf, editors, *ICSOC 2008 Workshops*, volume 5472 of *Lecture Notes in Computer Science*, pages 304–309. Springer Berlin / Heidelberg, 2009.

[41] Gioacchino La Vecchia and Antonio Cisternino. Collaborative Workforce, Business Process Crowdsourcing as an Alternative of BPO. In *Current Trends in Web Eng.*, volume 6385, pages 425–430, 2010.

[42] D. Davide Lamanna, James Skene, and Wolfgang Emmerich. Slang: A language for defining service level agreements. *Future Trends of Distributed Computing Systems, IEEE International Workshop*, 0:100, 2003.

[43] Business Process Execution Language. Accessed: 20.06.2012.

[44] Extensible Markup Language. http://www.w3.org/tr/xml/. Accessed: 20.06.2012.

[45] Web Services Description Language. http://www.w3.org/tr/wsdl. Accessed: 20.06.2012.

[46] P. Leitner. *On Preventing Violations of Service Level Agreements in Composed Services Using Self-Adaptation*. PhD thesis, Vienna University of Technology, 2011.

[47] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. Monitoring, prediction and prevention of sla violations in composite services. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 369–376, 2010.

[48] Philipp Leitner, Waldemar Hummer, and Schahram Dustdar. Cost-based optimization of service compositions. Technical report, Vienna University of Technology, 2011.

[49] Frank Leymann. Workflow-based coordination and cooperation in a service world. In *CoopIS*, volume 4275. Springer Berlin Heidelberg, 2006.

[50] Yin Li, Hye-Young Paik, and Jun Chen. Privacy inspection and monitoring framework for automated business processes. In *Web Information Systems Engineering*, pages 603–612. Springer, 2007.

[51] Zhening LI. *Polynomial Optimization Problems, Approximation Algorithms and Applications*. PhD thesis, The Chinese University of Hong Kong, 2011.

[52] Po-Ching Lin, Ying-Dar Lin, Yuan-Cheng Lai, and Tsern-Huei Lee. Using string matching for deep packet inspection. *IEEE Computer*, 41(4):23–28, 2008.

[53] ITSM IT Service Management. http://www.itsm.info/itsm.htm. Accessed: 20.06.2012.

[54] Marco Mont, Siani Pearson, and Robert Thyne. A systematic approach to privacy enforcement and policy compliance checking in enterprises. In *Trust and Privacy in Digital Business*, pages 91–102. Springer Berlin / Heidelberg, 2006.

[55] Marco Casassa Mont, Siani Pearson, and Pete Bramhall. Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services. In *DEXA Workshops*, pages 377–382. IEEE Computer Society, 2003.

[56] Oliver Moser, Florian Rosenberg, and Schahram Dustdar. Non-intrusive monitoring and service adaptation for ws-bpel. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 815–824, New York, NY, USA, 2008. ACM.

[57] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, 2nd edition, 2006.

[58] Business Process Modeling Notation. http://www.bpmn.org/. Accessed: 20.06.2012.

[59] Liam O'Brien, Paulo Merson, and Len Bass. Quality attributes for service-oriented architectures. In *Proceedings of the International Workshop on Systems Development in SOA Environments*, SDSOA '07, pages 3–, Washington, DC, USA, 2007. IEEE Computer Society.

[60] Gurobi Optimization. Gurobi optimizer reference manual, 2012.

[61] Michael P Papazoglou. *Web Services: Principles and Technology*. Prentice Hall, 2008.

[62] 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, nov. 2007.

[63] D.C. Parkes and J. Kalagnanam. Models for iterative multiattribute procurement auctions. *Management Science*, 51(3):435–451, 2005.

[64] C. Patel, K. Supekar, and Y. Lee. A QoS oriented framework for adaptive management of web service based workflows. In *Database and Expert Systems Applications*, pages 826–835. Springer, 2003.

[65] Siani Pearson. Taking account of privacy when designing cloud computing services. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 44–52. IEEE Computer Society, 2009.

[66] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Verlag, 2008.

[67] A. Pretschner, M. Hilty, D. Basin, C. Schaefer, and T. Walter. Mechanisms for usage control. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, pages 240–244, New York, NY, USA, 2008. ACM.

[68] Alexander Pretschner, Judith Ruesch, Christian Schaefer, and Thomas Walter. Formal analyses of usage control policies. In *ARES*, pages 98–105. IEEE Computer Society, 2009.

[69] Simple Object Access Protocol. http://www.w3.org/tr/soap/. Accessed: 20.06.2012.

[70] H. Psaier, F. Skopik, D. Schall, and S. Dustdar. Resource and agreement management in dynamic crowdcomputing environments. In *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*, pages 193–202, Sep 2011.

[71] Shuping Ran. A model for web services discovery with qos. *SIGecom Exch.*, 4(1):1–10, March 2003.

[72] C. Ringelstein and S. Staab. Dialog: A distributed model for capturing provenance and auditing information. *International Journal of Web Services Research*, 7(2):1–20, 2010.

[73] Florian Rosenberg, Max Benjamin Müller, Philipp Leitner, Anton Michlmayr, Athman Bouguettaya, and Schahram Dustdar. Metaheuristic Optimization of Large-Scale QoS-aware Service Compositions. In *Proceedings of the 2010 IEEE International Conference on Services Computing (SCC'10)*, pages 97–104, Washington, DC, USA, 2010. IEEE Computer Society.

[74] Nick Russell and W.M.P. Van Der Aalst. Work distribution and resource management in bpel4people: Capabilities and opportunities. In *Proceedings of the 20th international conference on Advanced Information Systems Engineering*, CAiSE '08, pages 94–108, Berlin, Heidelberg, 2008. Springer-Verlag.

[75] Benjamin Satzger, Harald Psaier, Daniel Schall, and Schahram Dustdar. Stimulating skill evolution in market-based crowdsourcing. In Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 66–82. Springer Berlin / Heidelberg, 2011.

[76] D. Schall. *Human Interactions in Mixed Systems - Architecture, Protocols, and Algorithms*. PhD thesis, Vienna University of Technology, 2009.

[77] Daniel Schall and Schahram Dustdar. Dynamic context-sensitive pagerank for expertise mining. In *SocInfo '10*, pages 160–175. Springer-Verlag, 2010.

[78] I. Sourdis. *Designs & Algorithms for Packet and Content Inspection.* PhD thesis, Delft University of Technology, 2007.

[79] J.P. Sousa, V. Poladian, D. Garlan, B. Schmerl, and M. Shaw. Task-based adaptation for ubiquitous computing. *IEEE Transactions on Systems, Man, and Cybernetics*, 36(3):328 –340, May 2006.

[80] Spiegel Online (In German). http://www.spiegel.de/wirtschaft/unternehmen/0,1518,813388,00.html. Accessed: 20.06.2012.

[81] QiMing Tian, Li Li, Ling Jin, and XinXin Bai. A novel dynamic priority scheduling algorithm of process engine in soa. *Web Services, IEEE International Conference on*, pages 711–718, 2009.

[82] Amazon Mechanical Turk. http://www.mturk.com/. Accessed: 20.06.2012.

[83] W.M.P. Van Der Aalst, M. Rosemann, and M. Dumas. Deadline-based escalation in process-aware information systems. *Decision Support Systems*, 43(2):492–511, 2007.

[84] The World Wide Web Consortium (W3C). http://www.w3.org/. Accessed: 20.06.2012.

[85] Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, April 2007.

[86] Daniel J. Weitzner, Harold Abelson, Tim Berners-Lee, Joan Feigenbaum, James Hendler, and Gerald Jay Sussman. Information accountability. *Commun. ACM*, 51(6):82–87, 2008.

[87] W. Xu, V. N. Venkatakrishnan, R. Sekar, and I. V. Ramakrishnan. A framework for building privacy-conscious composite web services. In *4th IEEE International Conference on Web Services (Application Services and Industry Track)(ICWS)*, 2006.

[88] Yahoo! Answers. http://answers.yahoo.com/. Accessed: 20.06.2012.

[89] R.T. Yu-Lee. *Essentials of Capacity Management*. Essentials Series. Wiley, 2002.

[90] N. Yuhanna, M. Gilpin, L. Hogan, and A. Sahalie. Information fabric: Enterprise data virtualization. *White Paper, Forrester Research Inc*, 2006.

[91] Chrysostomos Zeginis and Dimitris Plexousakis. Web service adaptation: State of the art and research challenges. *Cycle*, 2:1–66, 2010.

[92] L. Zeng, H. Lei, and H. Chang. Monitoring the QoS for Web services. *Service-Oriented Computing–ICSOC 2007*, pages 132–144, 2010.

[93] Liangzhao Zeng, B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *Software Engineering, IEEE Transactions on*, 30(5):311–327, may 2004.

[94] F. Zulkernine, P. Martin, C. Craddock, and K. Wilson. A policy-based middleware for web services sla negotiation. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 1043 –1050, 2009.