FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Behavior Management and Self-adaptation in Mixed Systems

Adaptation Models, Strategies, and Management

DISSERTATION

zur Erlangung des akademischen Grades

Doktor/in der technischen Wissenschaften

eingereicht von

Dipl.-Ing. Harald Psaier
Matrikelnummer 9826727

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Prof. Dr. Schahram Dustdar

Diese Dissertation haben begutachtet:

(Prof. Dr. Stefan Tai) (Prof. Dr. Schahram Dustdar)

Wien, 22.02.2012

(Dipl.-Ing. Harald Psaier)

Behavior Management and Self-adaptation in Mixed Systems

Adaptation Models, Strategies, and Management

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor/in der technischen Wissenschaften

by

Dipl.-Ing. Harald Psaier
Registration Number 9826727

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Prof. Dr. Schahram Dustdar

The dissertation has been reviewed by:

(Prof. Dr. Stefan Tai) (Prof. Dr. Schahram Dustdar)

Wien, 22.02.2012

(Dipl.-Ing. Harald Psaier)

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Harald Psaier
Spengergasse 25/A/5 1050

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

First and foremost, I would like to thank my adviser Prof. Schahram Dustdar for introducing me to the world of service science and the multicultural life on his institute. I have been very lucky to have an adviser who gave me the liberty to explore different ideas and directions, and at the same time the guidance I needed to complete the present dissertation.

I would also like to thank Prof. Stefan Tai for the many precious comments and suggestions that pinpointed some deficiencies and helped me to improve this work.

In particular, I would like to thank my regular collaborators Daniel Schall and Florian Skopik for the fruitful discussions on the topics in this thesis, as many as, the other contributors to the research work and the project work I was involved in during my studies, namely Benjamin Satzger, Lukasz Juszczyk, Philipp Leitner, Martin Treiber, Christian Inzinger, Roman Khazankin, and Christian Mader.

A special acknowledgement goes also to the IBM Research ASR Group including furthermost Heiko Ludwig and Laura Anderson who introduced me to business coupled research.

Most importantly, this would have not been possible without the continuous support of my family. This dissertation is dedicated to my family.

Finally, I am grateful for the constant financial support from the European Union projects S-Cube (grant no. 215483) and WORKPAD (grant no. 034749), funding parts of the research discussed in this dissertation.

Harald Psaier
Feb, 2012
Vienna, AUT

Abstract

The many open, online, and participative communities on the Web invite everyone to join and to contribute. Today's crowds like to get involved in the activities of *mass collaboration* including creation of content, processing tasks, discussion of ideas, or rating of opinions and products. This inspired new designs in hosting large crowds on a platform and encouraged particular business models for mediating this multi-talented work-force. The investigations in this thesis focus on such platforms that negotiate outsourced activities from customers to capable workers in collaborative networks. In service-oriented architectures this recent business trend introduced the Mixed Systems model, a model combining Human-Provided Services with the traditional image of services, the Software-Based Services. However, this distinct combination comprises new degrees of unpredictability for traditional workflows. With the flows now depending also on human collaboration, their coordination becomes less reliable. The system's management is more challenging.

From the many variations of these online communities the focus in this thesis is on collaborative networks, and in detail, Open Enterprise System and crowdsourcing platforms and studies the challenges of their integration with the Mixed System model. Furthermore, this thesis presents methodologies to track interaction data, and suggestions how to extract models of misbehavior from the logs in Mixed Systems. A framework with self-adaptation capabilities for Mixed Systems is designed. In the case of task processing degradation, the online approaches initiate instant redirection of tasks to similar workers. The offline approaches consider worker profiling, work group formation with brokers, scheduling strategies meeting skill profiles, and worker formation as a method to self-organize and stabilize these mostly ad-hoc visited large-scale collaboration networks. In an outlook configuration management for Mixed Systems is discussed. The results presented comprise simulations of the considered platforms and their particular features. The conducted experiments demonstrate the applicability of the taken approaches.

Kurzfassung

Die vielen offenen, online verfügbaren und teilnahmefördernden Webgmeinschaften, laden jedermann ein sich anzuschließen und Beiträge zu leisten. Diese Ansammlung von Teilnehmern, auch "crowds" genannt, ist heutzutage motiviert an verschiedensten Tätigkeiten der Massenzusammenarbeit teilzuhaben. Diese beinhalten z.B. das Erstellen von Content, das Verarbeiten von Aufgaben, die Diskussion von Ideen, oder das Bewerten von Kommentaren und Produkten. Diese Situation inspirierte neue Pläne für das Bereitstellen von crowd-geeingeten mächtigen Plattformen und förderte eigene Geschäftsmodelle zur Vermittlung dieser Ansammlung von verschieden Fähigkeiten. Die Nachforschungen in dieser Dissertation stellen Plattformen, welche abgegebene Aktivitäten von Kunden an fähige Arbeiter in "collaborative networks" weitervermitteln, in den Vordergrund. Diese aktuelle Entwicklung führte zur Einführung des "Mixed Systems" Modell in die service-orientierten Architekturen. Dieses Modell vereint vom Mensch zur Verfügung gestellte Services (Human-Provided Services) mit dem traditionellen Bild von Services (Software-Based Services). Dadurch, dass die Prozesse nun auch von menschlicher Zusammenarbeit abhängen, wird deren Koordination weniger verlässlich und die Verwaltung des Systems an sich wird schwieriger.

Von den verschiedenen Variationen dieser "online communities" sind für diese Disseration jene interessant, die als collaborative networks bezeichnet werden. Im Detail liegt die Einschränkung auf "Open Enterprise Systems" und "crowdsourcing" Plattformen und den Aufgaben, welche bei deren Integration mit Mixed Systems anfallen. Zudem werden Methoden vorgestellt die Interaktionen erfassen und Anregungen erbracht Modelle von falschem Verhalten aus den Logs von Mixed Systems zu exportieren. Ein Framework mit auto-adaptiven Fähigkeiten für Mixed Systems wird modelliert. Im Falle einer Verschlechterung der Abarbeitung von Aufgaben gibt es online Ansätze, welche ein Umleiten von Aufgaben zu ähnlichen Arbeitern initiieren. Bei den offline Ansätzen wird versucht oft unregelmäßig frequentierten Netzwerke für Zusammenarbeit mit etwas Selbstorganisation zu stabilisieren. Dabei werden Persönlichkeitsprofile der Arbeiter erstellt, die Bildung von Arbeitsgruppen mit einem verantwortlichen "broker" gefördert, die Arbeitspläne den Fähigkeiten anpasst und die Arbeitskraft weitergebildet. In einer Zukunftsperspektive wird das Konfigurationsmanagement für Mixed System angesprochen. Die präsentierten Resultate entstammen Simulationen der betrachteten Plattformen und berücksichtigen deren Charakteristika. Die durchgeführten Experimente unterstreichen die Anwendbarkeit der dargestellten Ansätze.

Contents

List of Figures

Listings

List of Tables

List of Acronyms

BPEL	WS-Business Process Execution Language
BPEL4People	WS-BPEL Extension for People
BQDL	Broker Query and Discovery Language
FOAF	Friend of a Friend
G2	Genesis2 testbed generator framework
HPS	Human-provided Service
OES	Open Enterprise System
QoS	Quality of Service
SBS	Software-based Service
SLO	Service-Level Objective
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
WS	Web service
WS-HumanTask	Web Services Human Task
WSDL	Web Services Description Language
WSLA	Web Service Level Agreement
XML	Extensible Markup Language

List of Publications

Parts of the work presented in this dissertation have been published in the following conference papers, journal articles, and technical reports:

1. Satzger B., **Psaier H.**, Schall D., Dustdar S. (2011). *Stimulating Skill Evolution in Market-Based Crowdsourcing*. 9th International Conference on Business Process Management (BPM) , 28th August – 2nd September, 2011, Clermont-Ferrand, France. Springer.

2. **Psaier H.**, Skopik F., Schall D., Dustdar S. (2011). *Resource and Agreement Management in Dynamic Crowdcomputing Environments*. 15th IEEE International Enterprise Distributed Object Computing Conference (EDOC), 29th August - 2nd September, 2011, Helsinki, Finland. IEEE.

3. Schall D., Skopik F., **Psaier H.**, Dustdar S. (2011). *Bridging Socially-Enhanced Virtual Communities*. 26th ACM Symposium On Applied Computing (SAC), March 21-25, 2011, TaiChung, Taiwan. ACM.

4. **Psaier H.**, Skopik F., Schall D., Juszczyk L., Treiber M., Dustdar S. (2010). *A Programming Model for Self-Adaptive Open Enterprise Systems*. 5th Workshop of the 11th International Middleware Conference (MW4SOC), November 29 - December 3, 2010, Bangalore, India. ACM.

5. **Psaier H.**, Juszczyk L., Skopik F., Schall D., Dustdar S. (2010). *Runtime Behavior Monitoring and Self-Adaptation in Service-Oriented Systems*. 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), September 27 - October 01, 2010, Budapest, Hungary. IEEE.

6. **Psaier H.**, Skopik F., Schall D., Dustdar S. (2010). *Behavior Monitoring in Self-healing Service-oriented Systems*. 34th Annual IEEE Computer Software and Applications Conference (COMPSAC), July 19-23, 2010, Seoul, South Korea. IEEE.

7. **Psaier H.**, Ludwig H., Anderson L., Shaw B. (2011). *Identifying and Managing Variation Scope in Service Management*, Technical Report, Jannuary 2012, Almaden-Vienna, IBM.

CHAPTER 1

Introduction

A fundamental idea of the Internet is to connect heterogeneous systems for information exchange. Early on the Net supported the initial form of an electronic mail communication for collegial interaction between the involved scientists and researchers [101]. Today, network-based interactions are a quotidian activity [100] and the most popular activity for a majority of Internet users is social networking [44]. With the evolution of the Internet from a medium for elites to a medium of everyday's use and all the Web 2.0 features, nowadays, *Virtual Online Communities* [80] including message boards, chat rooms, social networks, etc. invite everyone connected and interested in the offered topics to enter virtual relationships spreading potentially the whole globe. In particular, modern online social and collaborative networks thrive with a substantial amount of curious users eager to register and participate [79]. The trend established new business models and opportunities. Dedicated platforms host and manage the virtual meeting places for the different interactive activities. Major companies either provide the platforms or compete for the users' interest. Typical types of activities include reviewing, creating and sharing artifacts, networking, or processing tasks [24]. The growing interest opposed to the loosely coupled, heterogeneous infrastructures of the Web pose a major challenge on the platform management. Management tasks include efforts to maintain a sound operation of the infrastructure, to guarantee the quality of content, to cultivate loyal contributions, and to satisfy both of the sides, users and customers.

1.1 Motivation

In recent years, there has been a growing interest in the complex 'connectedness' of today's society. With their constant presence in today's mass media and media promoting participation, also for their own cause, in the past years, platforms for online communities have grow in popularity (e.g., Facebook [30] and Twitter [112] most notably), and as a result, to the size of large-scale systems. Observable phenomena include network structures, participation incentives, and the aggregate behavior of groups [29].

Notable also, a major interest in platforms which involve human collaboration, and more precisely, human task computation. These are part of *collaborative networks* [16] and focus on the creation of artifacts required by the community or individuals. Well known examples are open-source projects (e.g., Linux Kernel [109]) or knowledge harvesting platforms (e.g., Wikipedia [117]) for which individuals and groups organize and contribute work for the benefit of the community. Another more business oriented category of platforms promotes virtual collaboration and offers interested customers the possibility to outsource to their network. In the context of this thesis, two particular types are examined: The first ones, are business oriented expertise communities in collaboration networks referred to as *Open Enterprise Systems (OESs)*, and the second ones, are loosely coupled and miscellaneous tasks processing *crowdsourcing* marketplaces.

An OES comprises various communities dedicated to process activities generally related to a particular human expertise as a service. The communities base on inter-connected members that publish their expertise with profiles. The collection of profiles supports the discovery of individual experts. Community members receive activity requests from external requester, process them and respond with appropriate answers. An example includes collaborative partnerships between small and medium-sized businesses. The partnership helps the partners to complement each other and to pursue their business goals in a collaboration. Another even more loosely coupled type of collaborative networks are profit oriented task market places which promote outsourcing activities, i.e. tasks, to an Internet-connected transparent cloud of workers namely the crowd (e.g., The Amazon Mechanical Turk [5]). In marketplace oriented crowdsourcing companies outsource to an intangible (and generally large) network of people in the form of an open call. This can result in the form of peer-production (when the job is performed collaboratively), but often is also undertaken by sole individuals. The crucial prerequisite is to effectively announce the open call task to a large crowd of potential laborers [39]. In crowdsourcing the main motivation for this "unreliable" kind of outsourcing is twofold: On the one side, there is the "The Wisdom of Crowds" [105], that emphasizes the collective intelligence of many potential workers which outperforms the capabilities of an individual. On the other side, payment schemes are different from conventional outsourcing and generally cheaper [42].

The studies in this thesis consider the aforementioned types of collaborative networks and their possible alignment with the *Service-oriented Architecture (SOA)* paradigm. The SOA community has already recognized the emerging requirement of human skills and knowledge, e.g., in service compositions. Some of the process' steps mapped to services still require support in complex decision making, and as a result, human expertise or knowledge to proceed. One proposed solution that considers the particularities of such a combination is the *Human-provided Service (HPS)* model [91]. The HPS model can be considered a proxy. On its one side, it bases on a common SOA infrastructure, e.g., on established Web service (WS) technology and its standardized formats and protocols (WSDL, SOAP, etc.) and hides the particularities of the standard SOA functionalities (i.e., publish, find, call) from the human. On the other side, it provides the means to define and expose the functionality, i.e. activity, provided by the human as a service in association with an individual profile. Thus, the design allows humans to actively contribute, participate, and provide transparently, however, in the same fashion as traditional services and in a uniform way [92], aligned to existing standards.

This model opens new possibilities of service interaction to the application fields of traditional services. Mixed service-oriented systems are environments that mix humans, HPSs, and traditional *Software-based Services (SBSs)* in one service system. In the context of this thesis the combination is denoted as *Mixed Systems*. Such a heterogeneous environment provides different interaction combinations. The work in [95] lists some examples. First, the environment still supports the traditional compositions of software services with SBSs. Also, human collaboration can be supported by a collection of SBSs, e.g., to coordinate collaboration and share related artifacts. Furthermore as explained earlier, human interaction can be required as part of a software service composition. For example, the BPEL4People together with the Web Services Human Task (WS-HumanTask) specification defines human interactions in business processes via a task specification adopted for human contributors. Finally, services of both types can initiate interactions towards humans in an activity push scenario.

One particular requirement for Mixed Systems and the HPS model is flexibility in their characteristics For example, it cannot be assumed that humans and their HPSs react and behave the same and as predictably as SBSs in interactions. Despite the advantages and the many new forms of interaction in Mixed Systems, one of the major challenge discussed in this thesis concerns the particular runtime behavior of HPSs in these mixed environments. In contrast to the rather static business logics of traditional services, normally the behavior and interaction habits of humans and their HPSs cannot be modeled a-priori. Instead, the interactions of HPSs are usually adhoc style and their behavior manifests at runtime as a result of emerging collaborations [88]. Fundamental issues in such collaborative networks are the monitoring of human tasks, trust, and reputation mechanisms. Thus, the challenges to face include both, technological and social aspects which shape the operation constraints [51].

In one of the restrictions of this thesis, the *scope of the investigations* focuses exclusively on *misbehavior of HPSs* and its side-effects. The *open environments* as the observed (c.f., to aforementioned OESs and crowdsourcing) base on human collaboration and it is the openness and its loose structure which allows different skilled users to join, but also, to misbehave. Whereas, the traditional services take the role of collaboration support resources. They help for example to manage the collaboration, provide resources of knowledge, or are services that distribute the activities. Thus, in open environments this type of service is considered, either under direct control of the environment infrastructure provider, or a related HPSs. It is their responsibility to maintain the SBSs and guarantee their correct functionality.

In contrast, because of the absence of a strict company structure and the loose coupling there is no direct control of the human participants. This inevitably leads to a multitude of management problems when combining those environments with the contemporary businesses. Without any direct control and the strict structures participants are generally also free to adopt behavior patterns which are unsynchronized, contradicting, and most unexpected. Unfortunately, this misbehavior disrupts organized activity processing, the quality of the result, and a satisfying throughput. The presented counter measures represent the *main contributions* in this work. Collaboration misbehavior patterns, i.e. models of delegation misbehavior, are identified and approaches to resolve the misbehavior by exploiting the multitude of similar resources are outlined. Methodologies illustrate how to encourage a previously intangible crowd of workers to become organized, responsible, and devoted members.

A generally accepted model to handle autonomously the dynamics of large-scale systems is subject to the investigations in autonomic computing [41] and systems with self-* properties [85], respectively. Today's heterogeneous and loosely coupled environments overwhelm the administrator with a substantial effort in maintenance. For collaborative networks such as OESs and crowdsourcing marketplaces, guaranteeing a certain Quality of Service (QoS) towards the expectations of the customers is of particular importance. This demands for a management that tackles the challenges associated with providing a mix of HPSs and SBSs together with an adequate performance, reliability, robustness, exception handling, integrity, and availability. The traditional solution model in these research fields is an adaptation-loop design, the fundamental construction pattern of a self-aware architecture. The successful adoption of this model in many fields of research and application motivated the work in this thesis to design automated/semi-automated self-adaptation/organization approaches for collaborative networks as outlined. The contributions cope with the aforementioned problems of misbehavior and encourage the delegation of high value tasks to such open, loosely coupled environments.

1.2 Research Challenges and Questions

This thesis investigates approaches of self-adaptation/-organization in collaborative networks. The challenges tackled result from the management complexity related to these human centric environments. These are often prone to unpredictable behavior by the participants. The presented questions include problems that arise from the goal to support and guide the management of such networks with automation.

Thus, the research questions and challenges investigated in the scope of this dissertation are twofold. They comprise research challenges, in terms of designing concepts and methodologies, as well as engineering challenges for developing prototype implementations as a proof of concept. In detail, the research challenges are the following:

- **To analyze interactions in current collaboration environments and find common models of misbehavior:**

  - What common misbehaviors affect collaborative networks? How do they reveal? What are their implications and side-effects? How can they be modeled?

  - How can one identify and detect misbehavior based on the models? How to contain resulting system degradation? How to deploy applicable remediation plans?

- **To define and design a framework with all the components of a self-adaptive loop for Mixed Systems integration:**

  - What are the essential components required? What are the necessary extensions in the scope of a Mixed System infrastructure?

  - How can adaptation policies be expressed? What are considerable approaches for an acceptable intrusiveness and interference with a running system?

- **To evaluate the integration of the adaptation framework with Mixed Systems:**

  - What are the integration prerequisites and requirements? Which are the reusable standards and interfaces?

  - How to simulate the main behavior properties of Mixed Systems? What are their variations? How do the adaptation strategies perform? What are the advantages of simulation?

- **To support group formation in loose crowds and delegation of responsibility:**

  - Who can be considered an ideal crowd broker? What are a broker's prerequisites and essential capabilities? How to search and identify brokers in the network?

  - How to transfer responsibility and settle agreements? What are particular objectives of an agreement? What are reasonable criteria for a broker ranking in collaborative networks?

- **To maintain a loyal crowd of participants in crowdsourcing:**

  - What is a practicable approach for a platform to learn the real skills of the crowd? How to increase the confidence in the crowd's capabilities? How to motivate workers to return?

  - How can auctions and ranking support a fair distribution of tasks? Do training tasks pay off?

- **To provide automated configuration management for crowdsourcing environments:**

  - How can the task processing logs be used to standardize worker formation? How should one react to future trends?

  - What can be learned from past broker agreements? How can costs be estimated?

1.3 Contributions

Figure 1.1 illustrates the context and collection of contributions provided by this thesis. The studies are within the field of social and collaborative networks and focus on two particular variants: Open Enterprise Systems and Market-based Crowdsourcing environments. The contributions detail the alignment of these environments to Mixed Systems and provide detailed approaches for the necessary misbehavior management. In detail the contributions are:

- *Misbehavior modeling, detection, and adaptation:* Starting from the observation of human behavior in collaboration scenarios of online communities the first contribution is the design of a model to express misbehavior scenarios. Although derived from a common network model, the additional adoptions allow highlighting the influential factors and the results of misbehavior. This further provides the tools to outline the functionality of the thesis' detection and adaptation algorithms. The algorithms presented are examples of misbehavior adaptation applicable to an SOA environment.

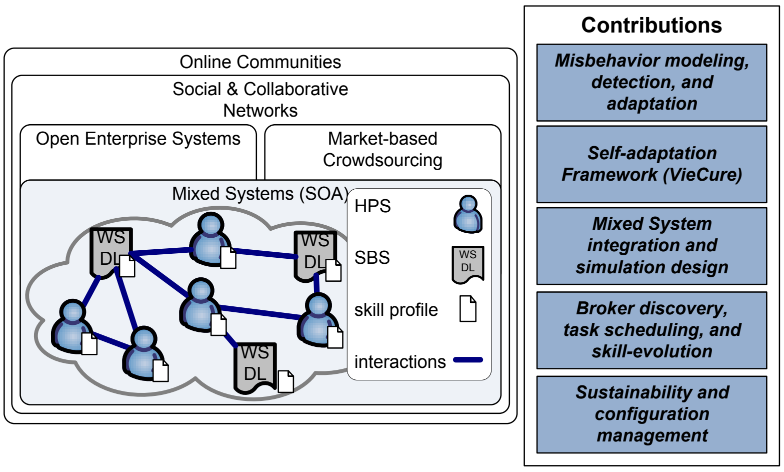

Online Communities
Social & Collaborative Networks
Open Enterprise Systems
Market-based Crowdsourcing
Mixed Systems (SOA)
WSDL
HPS
SBS
skill profile
interactions
Contributions
Misbehavior modeling, detection, and adaptation
Self-adaptation Framework (VieCure)
Mixed System integration and simulation design
Broker discovery, task scheduling, and skill-evolution
Sustainability and configuration management

Figure 1.1: Contributions and research area.

- *Self-adaptation Framework (VieCure):* The design challenges and operation details of *VieCure*, a framework that enhances Mixed Systems with self-adaptive properties, are discussed. The adoption particularities of the adaptation-loop design are detailed including the crucial interaction logging and policy definitions.

- *Mixed System integration and simulation design:* The integration of the framework with a testbed hosting a mixed SOA environment (G2) underlines the applicability of the novel framework design in various simulations. The G2 programming model is used to design, simulate, and control Mixed System. Furthermore, a programming model for policies is presented.

- *Broker discovery, task scheduling, and skill-evolution:* In parallel to the online adaptation approaches, some novel offline methodologies to avoid misbehavior and establish loyal and reliable personal in crowdsourcing are presented. In the context of market-based crowdsourcing a novel language to discover potential brokers is presented. A ranking scheme allows scheduling work more effectively. Finally, a method for skill-evolution is presented that motivates workers to continue their relationship with their platform. Furthermore, this relationship assists the platform to estimate more accurately the members' real skills.

- *Sustainability and configuration management:* A final chapter concludes the studies on sustainability for Mixed Systems. Starting from a generic approach of a normative model for variations and configurations, discussions about a more boundary guided management of crowdsourcing platforms are provided.

1.4 Organization of the Thesis

The content this work accumulates the research conducted and results published during the course of the author's PhD studies. It extends and unifies the substantial volume of work in the line with the thesis' contributions illustrated in Figure 1.1. The following chapter table connects a chapter's brief content description to the corresponding papers' references.

After the motivation and the introductory part in this very chapter the next Chapter 2 "State-of-Art" comprises and discusses the most influential work related to the content of the dissertation. It includes the main related fields of research, Mixed Systems basing on SOA infrastructures and crowdsourcing platforms. Furthermore, self-adaptive properties research is highlighted as the main direction in the solution approach. Thereafter, three parts arrange the results of the research work:

- **Part I:Framework Design and Misbehavior Models.**

  - Chapter 3 - Misbehavior Models and Adaptation in Mixed Systems [76]:

  From the many possible misbehavior affecting Mixed Systems in this chapter two common models are detailed: delegation *factory* and *sink behavior*. Detection and adaptation strategies are discussed and a first proof-of-concept in the form of a complete adaptation algorithm is presented.

  - Chapter 4 - VieCure: An Adaptation Framework for Mixed Systems [76, 74]:

  This chapter outlines an extensible framework for self-adaptation in Mixed Systems: the *VieCure* Framework. It comprises an adaptation-loop design and the necessary interfaces for a Mixed System environment integration. The concepts of potential integration are explored in detail. Examples of communication and control artifacts are provided.

- **Part II: SOA Integration and Adaptation Strategies**

  - Chapter 5 - Testbed Integration: Programming Model and Runtime Behavior Monitoring [78, 74]:

  The content of this chapter provides the details on an integration of the *VieCure* Framework with the Genesis2 testbed generator framework. The latter provides tools to design, deploy, control, and monitor Web service testbeds. In particular, also Mixed Systems and their behavior can be modeled, deployed, and tested. Moreover, the G2 programming model allows integrating the *VieCure* Framework and its adaptation loop. The combination offers a simulation of a misbehaving Mixed System that can be monitored and adapted by the *VieCure* Framework. The integration of the frameworks is extended by an additional programming model for the definition of adaptation policies. The presented results contain detailed information on various simulation runs and a discussion.

- **Part III: Sustainability, Maintenance, and Definition of Scope for Crowd Environments**

  - Chapter 6 - Broker Query and Discovery Language: Connecting and Finding Communities [90]:

The following two chapters are designated to the advantages of promoting group formation in crowds. The main concept is a broker that takes part of the responsibility in task processing and organizes a minor part of an otherwise loose crowd environment. At the beginning, and this defines the content of this chapter, suitable nodes capable of taking over the role of brokers need to be discovered. An approach for a novel broker query language is described. The details of the language structure are presented. Finally, the functionality is tested on collected simulation data.

- Chapter 7 - Crowdcomputing: Agreement management and ranking [77]:

This chapter extends the broker concept with details on a potential agreement definition that arranges the particularities of the transfer of responsibilities from a crowd platform provider to a broker. Additionally, a ranking algorithm is evaluated that allows identifying the best fitting broker for a task according to the settled agreements.

- Chapter 8 - Skill Evolution and Auctioning [87]:

Some final thoughts about methodologies of "self-sustainability" for crowd environments are the subject of this chapter. The approach explores the idea of skill evolution in order to combine supplemental training tasks with the advantage of more precise skill estimation, increasing confidence in the skills, and crowd satisfaction. An auctioning approach for task distribution increases the QoS when inviting capable crowd members only to the auctions.

Chapter 9 - Variation and Configuration Management [75]:

The chapter chooses a generic approach to discuss the idea of a normative model for variation management. In the context of this thesis example applications of the model for crowdsourcing platform configuration management are highlighted.

The work is summarized by a conclusion which hints ideas for future extensions.

CHAPTER 2

State of the Art: Mixed Systems, Self-adaptive Systems, and Crowdsourcing

2.1 Overview

The research on Mixed Systems, Self-adaptive Systems, and crowdsourcing influenced the contributions presented in the chapters of this thesis the most. *Mixed Systems* are a particular type of service-oriented systems integrating, both, traditional and services provided by humans. *Crowdsourcing* emerged in recent years and refers to an online, distributed problem-solving and production model. Finally, *Self-adaptive Systems* help to cope with system complexity when facilitating administration with appropriate self-management functionality. The chapter covers the main challenges of the related research fields and presents their results connected to the research in this thesis. The following sections give the reader a comprehensive introduction to all of the three topics.

2.2 Mixed Systems

Web services (WSs) and Service-oriented Architectures (SOAs) have become the de-facto standard for designing distributed and loosely coupled applications. They allow modeling modular information systems in distributed environments [4, 72]. Usually orchestrated by a process definition, SOAs' compositions of services collaborate on various activities to complete the process. However, part of the activities in this collaboration can also require human knowledge and expertise. Therefore, many of these applications demand for a mix of interactions between humans and software services. An example is a process definition which includes steps that can only be processed by humans. Typical steps that require human involvment include activities that cannot dispense with human considerations, decision making, and approval to progress or to complete.

Mixed Systems [88] extend the solely software implemented capabilities of traditional service-oriented systems with human capabilities through *Human-provided Services (HPSs)*. The integration of humans and Software-based Services (SBSs) is driven by the difficulties to adopt human expertise into software implementations. Rather than dispense with the expertise, in HPSs a human handles tasks behind a traditional service interface. For all the WS related functionality, such as, communication and discovery, the HPS framework [91] bases on well known SOA standards including the specifications of Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP), and others from the WS-* family. Furthermore, the Mixed Systems concept can be adopted to support flexible service-oriented collaborations across multiple organizations and domains [91] similar to emergent collectives as defined by the networks of interlinked valued nodes (services) in [73].

In short, the concept of HPSs closes the gap between pure SBSs environments and humans desiring to provide their skills and expertise as a service in a collaborative process. Note, that with the humans in the process the interactions change from a strict predefined process flow [58, 106] to ad-hoc contribution requests and dynamically structured process collaborations.

Mixed Systems and WS Specifications

Major industry players have been working on standardizing protocols and languages to allow people to interface with WS environments. To address the lack of human interactions in service-oriented businesses [58], specifications including BPEL4People [3] and WS-HumanTask [61] have been defined.

The **WS-BPEL Extension for People (BPEL4People)** is an extension to the WS-Business Process Execution Language (BPEL) specification. It addresses the lack of human interactions in the specification and contains a new type of basic activity. This new type allows including human tasks in a process or attaching such a task from outside to the process. This extension is based on the WS-HumanTask specification.

The **WS-HumanTask** introduces a definition of human tasks to business processes, including their properties, behavior and a set of operations used to manipulate human tasks. It specifies work for humans as part of business processes. Furthermore, it describes interaction between humans which are invoked as services, whether as part of a process or otherwise. A coordination protocol allows controlling autonomy and life cycle of service-enabled human tasks in an interoperable manner.

Consider however, both standards have been designed to model interactions in an agreed processes sequence for closed enterprise environments where people have predefined, mostly static roles and responsibilities. However, Mixed Systems must also consider also dynamic, loose structures and a flexible integration of HPSs into the activities. Thus, a simple adoption of the standards is not feasible.

Open Service-oriented Environments

This thesis considers a combination of SOA concepts and collaboration networks (e.g., [16]). The modular concept of HPSs provides an ideal grounding for different types of interactions, levels of participation, and roles that reflect the properties of existing communities on the Net, and in particular, the later discussed crowdsourcing platforms. In these *open service-oriented environments* services can be added and leave the system at any point in time. Also, following the open world assumption, humans actively shape the properties including availability of services. This leads to a dynamic environment where the availability of people is under constant flux and change [17]. A main challenge tackled by this thesis is to cope with the dynamic properties of such a system. Next, a representative studied in many scenarios of this thesis is outlined.

2.3 Open Enterprise Systems

Open Enterprise Systems (OESs) are open service-oriented environments and collaborative networks organized in communities. Communities are established by members with the same interests and skills. This information is provided by the members' profile. A typical scenario includes a requester that requires human assistance with an activity. The profiles help to find the matching expert. The requester selects the expert, submits the activity and awaits the response. Examples of OESs employing SOA infrastructures have been studied in [16, 93]. These include small and medium-sized companies and their bilateral alliances to compete with global players. This protects the partners against the dynamics of economy and business, supporting them also to harvest business opportunities that a single partner cannot take. The result of these associations is referred to as *virtual organizations* supporting enterprise collaboration. Particular instances have been explored in [97]. The example outlines a science collaboration network. It comprises scientists, members from national and international research labs, and experts from the industry. Furthermore, professional virtual communities are discussed in [98] that provide help and support on requests of each other, e.g., law firms and insurance companies.

Next is another example of service related environments studied more in detail, however, not necessarily basing on an SOA infrastructure. There is a growing interest in crowdsourcing environments. As human centric environments they where of major interest for the content and research presented in this thesis.

2.4 Crowdsourcing Environments

The recent trend towards *collective intelligence* and crowdsourcing can be observed by looking at the success of various Web-based production platforms that have attracted a huge number of users. Crowdsourcing applications [13] are online, distributed problem-solving and production models that have emerged in recent years. They are typically open Internet based platforms where problem-solving tasks are distributed among a group of humans. Crowdsourcing follows the 'open world' assumption and generally comprises three roles identified in [114]. First, there is the crowdsourcing platform and its owner, also referred to as the platform *provider*. A crowd *customer* is interested in outsourcing tasks to the crowd. The registered vast number of crowd *workers* with different skills offers then individual solutions to the outsourced problem. Apart from its benefits of multiple redundant workforce and collective intelligence, many of the challenges of crowdsourcing are related to its distributed and open nature. The main challenges remain how to organize and manage the crowd and identify potentially missing skills [13]. In the following different types of crowdsourcing environments are described.

Platform	Description	Category	Market-place	Specia-lized	Profit-oriented
Wikipedia [117]	online encyclopedia	DK,CF[2]	–	✓	–
iStockphoto[3]	multi media stock	CC	–	✓	✓
Threadless[4]	clothing design	CC	–	✓	✓
uTest [113]	software testing	OI	✓	✓	✓
Marketocracy[5]	virtual advisory board	DK	–	✓	✓
InnoCentive[6]	R&D challenges	OI	✓	–	✓
MTurk [5]	task market	CL	✓	–	✓
Samasource[7]	review tasks	CL	✓	✓	✓

Table 2.1: Classification of existing crowdsourcing platforms.

Crowdsourcing: Classifications and Types

Currently, crowdsourcing tasks are usually short term activities albeit demand human skills. Nevertheless, different categories of platforms and task topics can be identified. An initial survey on the most prevalent types of tasks processed by crowds is available in [13]. The work in [114] offers an initial list of distinct business categories. The more recent list used in Table 2.1 derives from the *Crowdsourcing Industry Website*[1]. The categories include *distributed knowledge (DK)* with platforms for open Q&A, user-generated knowledge systems, news, citizen journalism, and forecasting. Next, there is *cloud labor (CL)*. Following the concept of cloud computing this form of crowdsourcing tries to mediate resources, i.e. human workers, from a dynamic distributed virtual worker pool. Mediated labor, i.e. tasks, ranges from simple to complex activities. Then there is *crowd creativity (CC)* with online communities of creative developers which design original products and concepts. There is also a category *open innovation (OI)*. Platforms of this type generate, develop and implement new ideas. Finally, *crowdfounding (CF)* has three models: (i) donations, philanthropy and sponsorship with no expected financial return, (ii) lending and (iii) investment in exchange for equity, profit or revenue sharing.

Two operating modes have been identified by [114]. There is *marketplace oriented* and *competition based* crowd platforms. In marketplace oriented crowdsourcing, crowds are organized by providers. These take the functionality of brokers that attract customers and bid for tasks. In a second step they provide the means to announce and distribute the requests in the crowd. In competition based crowdsourcing the request is similar to an open call. In this scenario the platform is only used as a medium to announce the task to the registered crowd. Usually the results are then collected by the customer and the winning submission is picked. The crowdsourcing operating mode considered in the examinations of this thesis base on a marketplace oriented

[1]Crowdsourcing for Industry: `http://www.crowdsourcing.org/`
[2]e.g., in the holiday season for own founding
[3]iStockphoto: `http://www.istockphoto.com/`
[4]Threadless: `http://www.threadless.com/`
[5]Marketocracy: `http://marketocracy.com/`
[6]InnoCentive: `http://www.innocentive.com/`
[7]Samsource: `samasource.org`

crowdsourcing environment where mediators manage the crowd for the customers. Further, classifications are available in [38] where the authors differ between *non-profit* and *profit driven* crowdsourcing and, regarding the tasks' content, *specialized* and *non-specialized platforms*. Table 2.1 gives an overview of different existing platforms and their classification.

Next, two crowdsourcing platforms of well know and established Internet companies Amazon and Yahoo are outlined. The following section will give insight in their activities and crowd management methodologies.

Crowdsourcing Platforms: Organization and Management Examples

The **Yahoo! Answers (YA)** platform focuses on information sharing. The crowd of this platform is involved in creating and collecting the information. The platform provides three types of activities for their users. The first is to ask questions, the second is to answer the questions, and the third allows searching for existing answers. The work model of YA is mainly based on interactions between members. Questions are asked and answered by humans. Thereby YA has a rewarding scheme based on points to motivate their users. Participants get 100 points by signing-up to the platform [120]. For each answer being provided, users get additional points (more points if the answer is selected as best answer). However, users get negative points if they ask questions, thereby encouraging members to provide answers. An interesting aspect of YA concept is the role of *two-sided markets* analyzed in [54]. Two-sided markets arise when two different types of users may realize gains by interacting with one another through one or more platforms or mediators. Based on the rewarding scheme in YA, users tend to have either role – being answerer or asker – instead of having both roles. In the context of YA and human-reviewed data, [104] provided an analysis of data quality, throughput and user behavior.

The Amazon Mechanical Turk (MTurk) offers access to the largest number of crowdsourcing workers (according to Amazon 400000 workers registered by 2010 [83]). It presents itself as a marketplace for work, where work is on-demand and workforce scalable. An Amazon customer can sing-up as a Turk Worker or a provider of Human Intelligence Tasks (HITs). With their notion of HITs that can be created using a Web service based interface they are closely related to the aim of mediating the capabilities of crowds to service-oriented business environments. According to one of the latest analysis of MTurk [42], HIT topics include, first of all, transcription, classification, and categorizations tasks for documents and images. Furthermore, there are also tasks for collecting data, image tagging, and feedback or advice on different subjects. Various research work gives insight into the demographics of the MTurk [38,83,43]. Most of the MTurk's workers are either located in Asia and the United States. MTurk's customers are mainly from the United States, also because currently a United States bank account is required. With the huge number of participants at the MTurk, the major challenges are to find skilled workers on request that are able to provide high quality results for a particular topic (e.g., see [2]), to avoid spamming, to recognize low-performers, and cheating participants. Also, there is no means to monitor the progress of the task during processing from the customers perspective [25]. Meanwhile, the problem is solved by the vast redundancy of crowd workers. The same low price task is distributed multiple times and the best results are manually verified, selected or aggregated. However, there is no means that guarantee a high-quality or even a single correct result.

Crowdsourcing: Criticism and Challenges

Croudturfing is a novel term that denotes the combination of both traditional crowdsourcing systems and astroturfing behavior [116]. Astroturfing refers to information dissemination campaigns that are sponsored by an organization that likes to spread false, also illegal information, such as defamatory rumors, false advertising, or suspect political messages. Although astroturfing predates the Internet, crowdsourcing systems give astroturfing a new dimension by providing the ability to quickly mobilize large groups for their cause. These platforms essentially leverage one of the weak points and criticisms about crowdsourcing, the trend towards cheap labor [13].

The adaptation and improvement approaches presented in this thesis do not consider crowdsourcing platforms that expose a malicious behavior as a whole. Nevertheless, the results propose a trend towards tasks of higher value where the quality of the result counts and the pricing schema is fair. Moreover, this thesis copes with the traditional challenges of crowdsourcing. As mentioned before the two main challenges in crowdsourcing are management and identification of potential leaks. Analyzing the two major competitors in the previous section, one can conclude that one shortcoming of most existing real platforms is the lack of detailed worker *skill information*. Platforms either don't care or allow the workers to create their own profile. But finding skilled workers that are able to provide high quality results in open Web-based communities is exactly the non-trivial problem (e.g., see [2]). Also, skill profiles need to be updated. Humans tend to change their preferences and acquire new skills. Most platforms have simple measures to prevent workers (in MTurk, a threshold of task success rate can be defined) from claiming tasks. Otherwise a task can be claimed by any worker interested. The method fits the purpose of low cost tasks as offered by MTurk. Platforms that wish to offer more complex and valuable tasks require better knowledge about their workforce. In this thesis an advanced, metric supported ranking of workers in the context of a task is presented. More complex task often need the collaboration of a whole team. Furthermore, finding broker and assisting workers in team formation is subject to the studies conducted for this thesis.

2.5 Self-adaptive Systems

Management of large-scale systems is a challenge in every aspect. It begins with the assembling of heterogeneous parts to a purposefully collaborating, and also profitable system. Known difficulties include poor security, dependability, and maintenance along with many other difficulties [48]. Within this category are also today's WS environments. From the whole multitude of management challenges the focus in this thesis is on *behavior management*. Originally, behavior management in WS context related to maintaining the expected behavior of a service during its execution described by its service behavior description, e.g., workflow process [71]. With aforementioned Mixed Systems the behavior management challenges open a new dimension: the *unpredictable behavior of humans*. The remainder of this section gives an insight on the research on systems with self-* properties. The emphasis is on SOA related works.

The design of large-scale systems comprising heterogeneous, tightly, and loosely coupled components result in a plethora of management challenges. Apart from size and complexity, at runtime, these systems need to remain dynamic and are often subject to necessary changes and improvements. Because of, e.g., the many resulting dependencies and inter-connectivities these environments become difficult to control and cause increasing maintenance costs. The IBM autonomic computing initiative [70, 40, 84, 47] realized the problem of humans overwhelmed by the effort to properly control such systems assemblies. In their vision they propose a *self-awareness* design for the described systems. Following the life-cycle of a system their design comprises four main self properties necessary for a self-aware system that still serves as the de facto standard in this domain [85]:

- *Self-configuring* is the ability of a system to readjust itself "on-the fly".
- *Self-healing* is the ability to discover, diagnose, and react to disruptions.
- *Self-optimization* tries to maximize resource utilization to meet end-user needs.
- *Self-protection* can anticipate, detect, identify, and protect itself from attacks.

Over the years the list of properties has been extended and is currently covered by the term self-*, self-X respectively, properties research (refer to glossary in [102] for an overview of the fields). Approaches can be found in various fields of research and application. These include the higher system layers such as models and systems' architecture [22, 18] application layer, and large-scale agent-based systems [20, 108, 11]. Middleware related approaches are described in [12, 55], and at a lower layer designs include operating systems [107, 94], embedded systems, networks, and hardware [32].

In the area of WS adaptation recent contributions of self-adaptive strategies can be found for service discovery, processes design, and QoS optimizing methods [37, 36, 82, 8, 68, 57].

Self-adaptation Concepts

From the studies of the previously mentioned related work results a number of generally valid concepts for self-adaptive systems. The three most important to this thesis related concepts are described next.

Adaptation Loop. A common concept to enable self-awareness is a loop design also denoted as autonomic element [41]. The loop comprises a manager that holds four distinct functions with individual tasks. The *monitor* function gathers status information from the system through sensors and pre-processes it for the analyze task. The *analyze* function determines whether the received monitored information must follow a designated action. This is generally done by comparing status information to system specific thresholds. The *plan* function provides an accurate, sound, and step-by-step deployment of the actions demanded. Thereby considering also the changing conditions of a running system. The *execute* function executes the parts of previously conceived plans on the managed element. Together they are known by the acronym MAPE loop. Later a fifth function was added. The *knowledge* function represents a knowledge base consumed and produced by all four previously mentioned functions providing a learning process and extending the loop to MAPE-K.

Adaptation Policies. In the context of self-adaptation a policy is a formal behavioral guide for the observed system. Hence, policies allow configuring the MAPE loop and providing a configuration management for the loop's operation mode. In autonomic computing three types of policies have been defined [49]: Action, Goal and Utility Function. *Action policies* are considered reactive policies and similar to an `IF(Condition)THEN(Action)` statement. *Goal Policies* are situation aware. They consider a set of states to calculate a response. Finally, *Utility Function Policies* try to increase the systems utility by extending their system knowledge to information including history, system capabilities, current system state, and current environment state. There has also been work on frameworks to define policies. Related is for example the WS-Policy specification [119]. It provides a syntax to describe policies of entities in a Web service-based system.

Intrusiveness. This refers to the potential degree of interference with the observed running system. While some of the previous described examples design and provide well defined maintenance interfaces for all four of the self-* properties, in the process centric context of WSs with different providers and loosely coupled compositions the approach has to be different. Instead of providing sophisticated maintenance interfaces, service implementations generally either outdate or are replaced by newer versions of implementation [56]. Thus, recovery strategies for malfunctioning services consider non-intrusive methods. As long as the process structure and engine provides transparent invocations of the process' embedded services, services can be replaced on-the-fly, calls can be redirected, failing instances isolated, or any combination of the three. This non-intrusive approach is also a valid for HPSs in the here considered context of Mixed Systems.

Part I

Misbehavior Models and Framework Design

CHAPTER 3

Misbehavior Models and Adaptation in Mixed Systems

3.1 Chapter Overview

Open Enterprise Systems comprising communities of experts are often victims to their flexibility in the collaborations. This brings new challenges to the Mixed System model. One of the reasons is that in such systems the collaboration is rather activity push based. Activities can be delegated and re-delegated. This chapter is after the possible unpredictable and unintentional side-effects of such delegation chains in flexible systems. In particular, if delegation behavior between the participants degenerates and activities are not processed or completed as a results of misbehavior. The content of this chapter presents a network model that illustrates the resulting complexities. An algorithm for delegation misbehavior detection and adaptation is detailed.

Chapter Outline. In the following Section 3.4 one of the results of flexible interactions and compositions in Mixed Systems, i.e., delegations and their characteristics are highlighted. Thereafter, in more detail misbehavior patterns resulting from delegations are illustrated. In Section 3.5 a detection and adaptation methodology is presented in the form of an algorithm.

3.2 Background

Service-oriented Architectures (SOAs) are an emerging paradigm to realize extensible large-scale systems. As interactions and compositions spanning multiple enterprises become increasingly commonplace, organizational boundaries appear to be diminishing in future service-oriented systems. Today, processes in collaborative enterprise networks are not restricted to single companies alone but may span multiple organizations, sites, and partners. External consultants and third-party experts may be dynamically involved in certain steps of such processes. These actors perform assigned tasks with respect to prior negotiated agreements. Also, single task owners may consume services from external expert communities.

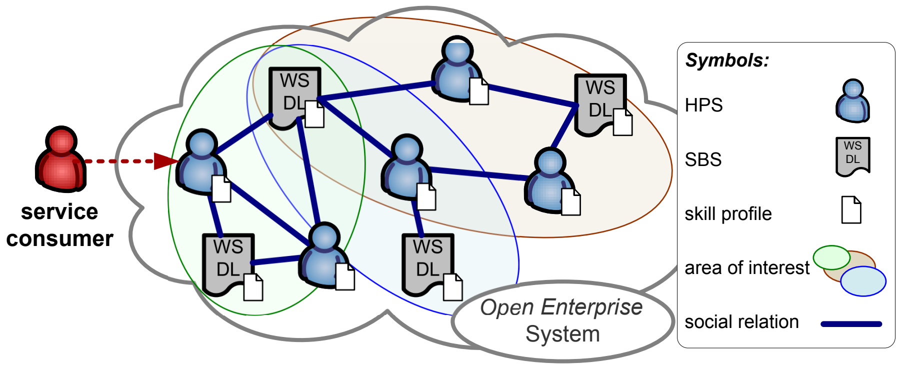


Figure 3.1: Open Enterprise System on a Mixed System infrastructure.

3.3 Mixed System Infrastructure

Mixed service-oriented systems, or Mixed Systems for short, have already been introduced in the related work of the previous Chapter 2. They represent a combined SOA environment including humans and software services. Following the example of an Open Enterprise System (OES) with characteristics also outlined in the previous chapter Figure 3.1 illustrates such a collaboration environment organized by communities aligned to a Mixed System infrastructure. A Mixed System infrastructure consists of HPSs and SBSs that in the example belong to different communities. The members and boundaries of these communities are discovered based on their skill profiles and main areas of interest (depicted as shaded areas), and are connected through certain relations, e.g., Friend of a Friend (FOAF) ontology [14] describing members, their activities and their relations i.e., roles and hierarchies in this context, to other services (human or software). Activities (e.g., see [67]) are a concept to structure information in flexible collaboration environments, including the goal of ongoing tasks, involved actors, and utilized resources such as documents or services. They are either assigned from the outside of a community, e.g., belonging to a higher-level process, or emerge by identifying collaboration opportunities. In the example community members receive requests as their activities from external service consumers, process them and respond with appropriate answers. Including HPS technology has both, the advantage of including and offering human expertise into business processes in a loose coupled fashion. The environment uses standardized SOA infrastructures, relying on widely adopted standards, such as SOAP and the WSDL, to unify humans and software services in one harmonized environment.

Whilst adaptations of SBSs and their environments have already been discussed in many related works, for the motivation of the models and framework presented in the upcoming sections and chapters SBSs are considered rather reliable in contrast to HPSs and redundantly available . Also, in the scenarios a major human involvement is considered and the interactions with SBSs are limited to collaboration support activities. Thus, the adaptation work presented focuses on failures and performance degradation related to HPS when operating an Mixed Systems. For instance, performance degradation can be expected when a minority of distinguished humans become flooded with tasks while the majority remains idle. Such load distribution problems can be compensated by adapting the *delegation behavior* of actors as discussed in [96]. Furthermore, consider shifting interests and evolving skills profiles. In that cases skill profiles of HPSs as well as people's interaction behavior may change over time, thus, requiring again adaptations of the underlying infrastructure.

The remainder of the chapter gives insight into the main components of the *VieCure* Framework designed to avoid and adapt the side-effects of misbehavior in Mixed Systems. As the studies of the following chapters will demonstrate, the framework's adaptation methodologies can be applied to any environment basing on Mixed System infrastructures.

3.4 Misbehavior in Mixed Systems

The same difficulties that hamper large-scale systems affect also Mixed Systems with their particular configuration. However, what separates Mixed Systems from the other is the major human involvement. In more detail, Mixed System applications are difficult to manage due to changing interaction patterns, behavior, and faults resulting from varying conditions in the environment. The focus of this thesis is on Mixed Systems' unreliability resulting from misbehavior of the involved humans. In contrast to software, a human activity usually fails because of an unexpected behavior. For example, despite the expectation the responsible humans involved fail to deliver their results on time possibly also because of inter-dependencies.

Let us consider a Open Enterprise System (OES) as outlined in Figure 3.1. The roles in such a system include the service consumer or *customer* which sends an human activity request to the network of experts. HPSs and SBSs take the role of collaborating expertise resources, more generally, the *workers*. In this example collaboration is reduced to *delegations*. Some experts act as an intermediate. These split the received activities into smaller tasks which together complete the activity. This type of resource is referred as the *broker*.

Flexible Interactions and Compositions

Various circumstances may be the cause for inefficient task assignments in Mixed Systems. In an OES, e.g., performance degradation can be expected when a minority of distinguished workers become flooded with tasks while the majority remains idle. One could further assume that workers delegate work they are not able to perform because of missing skills or due to overload conditions. Delegation receivers can accept or reject task delegations. In collaborative networks with community character, for example, members usually have explicit incentives to accept tasks, such as collecting rewards for successfully performed work to increase their community standing (reputation). In a business related system these incentives join the monetary reward. Delegations work well as long as there is some agreement on members' *delegation behavior*: How many tasks should be delegated to the same partner in a certain time frame? How many tasks can a community member accept without neglecting other work? However, if misbehavior cannot be avoided in the network, its effects need to be compensated.

Consider a scenario as depicted in Figure 3.1 where the OES is organized in areas of interest, i.e. communities, comprising networks of expert workers: Someone is invited to join a community, e.g., computer scientists, in the expert network. Since s/he is new and does not know many

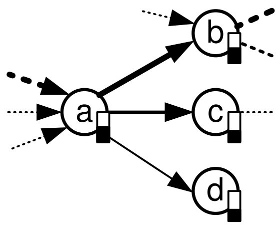


Figure 3.2: Delegation model components.

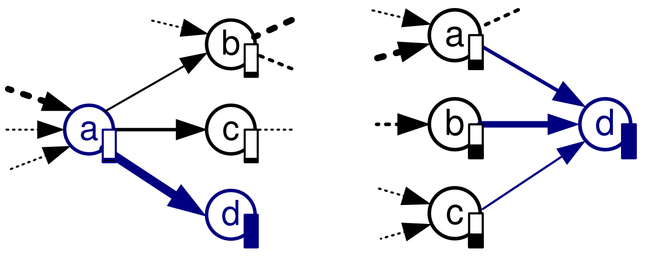
other members, s/he is not well connected in the network. In the following, s/he will receive tasks that match her/his public expertise profile, but is not able to delegate to other members. Hence, s/he may get overloaded if several tasks arrive in short time spans. A straightforward solution is to find another member with similar capabilities that has free capacities. A central question is how to support this process in an effective manner considering global network properties. The system is an ad-hoc expert network and failures impact the network in a harmful manner by causing degradation. In particular, the challenge is to identify common misbehavior of community members and highlight concepts lend, e.g., from self-healing research to recover from degraded states in SOA-based systems comprising human and software services.

Delegation Patterns

The characteristics of human misbehavior in Mixed Systems can be manifold. A thorough survey on the many variations is not subject of this thesis. Instead next, two common pattern of misbehavior related to aforementioned environments with delegations are presented. A delegation sink is the result of intensive delegation to a particular node. An overload of the very node is the result. The other, the delegation factory is considered a node that delegates without an obvious reason most of its work to the neighbors. The following paragraphs present the particularities of the models. These are then subject of the related investigations on misbehavior management in this thesis.

Starting from the previous example of an Expert Network let us consider each node, i.e., community member, has a pool of open tasks. Therefore, the load of each node varies with the amount of assigned tasks. In Figure 3.2 the load of nodes is depicted by vertical bars. If a single node cannot process assigned tasks or is temporarily overloaded, it may delegate work to neighbor nodes. The usual delegation scenario is shown in Figure 3.2. In that case, node a delegates work to its partner nodes b, c, and d, which are connected by channels. A channel is an abstract description of any kind of link that can transport various information including communication, coordination, and collaboration. In particular, a *delegation channel* has a certain *capacity* that determines the amount of tasks that may be delegated from a node a to a node b in a fixed time frame. None of the nodes is overloaded with work in the ‘healthy state’. In Figure 3.3 two possible patterns resulting from misbehavior in such a situation are illustrated and detailed next.

Delegation Factory. As depicted in Figure 3.3a a delegation factory produces unusual amounts (i.e., unhealthy) of task delegations, leading to a performance degradation of the entire



(a) Delegation Factory model. (b) Delegation Sink model.

Figure 3.3: Delegation misbehavior models.

network. In the example, node a accepts large amounts of tasks without actually performing them, but simply delegates to its neighbor node d. Hence, a's misbehavior produces high load at this node. Work overloads lead to delays and, since tasks are blocked for a longer while, to a performance degradation from a global network point of view.

Delegation Sink. A delegation sink behaves as shown in Figure 3.3b. Node d accepts more task delegations from a, b, and c as it is actually able to handle. In the collaborative network, this may happen due to the fact that d either underestimates the workload or wants to increase its reputation as a valuable collaboration partner in a doubtful manner. Since d is actually neither able to perform all tasks nor to delegate to colleagues (because of missing outgoing delegation channels), accepted tasks remain in its task pool. Again, the misbehavior observation is made at the delegation receiver. It causes blocked tasks and performance degradation from a network perspective.

Identifying the actual misbehavior source with these patterns is a difficult task. First of all, a delegation factory might not be perceived as one from the customer's point of view who has no knowledge of the delegations. Of course, the customer might notice the network degradation at one point but cannot pinpoint the source and counteract decidedly. Also, the delegation source might not derive from the over-delegating node but might be caused by, e.g., false information. Consider a node is related to some outdated profile information and despite of getting a certain task frequently it decides to forward all those tasks to neighbor nodes. The node has been forced into this misbehavior. The situation is different with the delegation sink. This could be a node that tries to increase its reputation or benefit artificially by accepting as many tasks as possible. Another possibility was discussed previously where a new node joins the system and is overloaded by the misbehavior of its neighbors.

3.5 Regulation of behavior

The presented self-adaptive algorithm for Mixed Systems applies the regulation of a node's behavior in a non-intrusive manner. Instead of directly interfering with the misbehavior sources and adapting the nodes, the behavior is influenced by restricting the delegations, establishing new delegation channels, and by redirecting work. Note, that the scenario considers cross-boundary collaborations of nodes. Their misbehavior might not be directly controllable. From a point of view of the infrastructure provider, not every service provider might be interested in providing service maintenance interfaces. For example, s/he might never agree to provide direct contact information (e.g., phone numbers) to her/his experts, or generally approve service processing style regulations without authorization. Therefore, as an infrastructure provider the idea of regulating the connections (i.e., channels) was considered the most appropriate, an assumption continued in the following chapters.

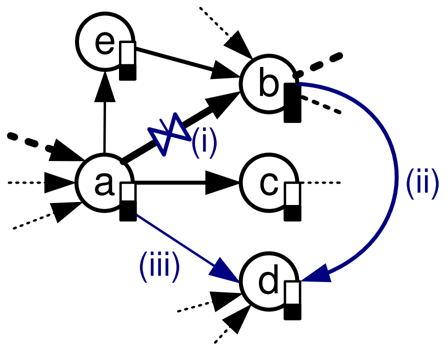


Figure 3.4: Self-adaptive recovery actions for a failure affected node.

Next, the modular structure of the self-adaptation mechanism in Algorithms 3.1, 3.2, and 3.3 are outlined and with respect to the concepts of the misbehavior scenario in Figure 3.4.

Trigger. The first module outlined in Algorithm 3.1, a trigger, represents a filter for the failure scenario in Figure 3.4. As a prerequisite any agreements and constrains monitored by this self-adaptation approach need to be expressed as threshold values. These values are integral part of the decision logic of a trigger module.

Diagnosis. A recognized violation fires the second module Algorithm 3.2 , the diagnosis. It defines the necessary recovery actions by analyzing the result of the task history evaluation of the failing node.

Recovery Actions. The possible resulting recovery actions are listed in the last three modules of Algorithm 3.3 . The first balances load of a failing node by restricting incoming delegations. The second provides the failing node with new delegation channels for blocked tasks.

Algorithm 3.1: Detection of misbehavior and recovery actions.

```
Require: Monitoring of all nodes
Require: Listen to Events
Trigger triggerQueueOverload(event) ;
begin
    node ← event.node                          /* Affected node */
    if $q > \vartheta_q$ then
        fire diagnoseBehavior(node)
    end
end
```

Algorithm 3.2: Diagnose sink and factory behavior.

```
1  Diagnosis diagnoseBehavior(node) ;
2  begin
3      $recActs \leftarrow \emptyset$          /* Set of returned recovery actions */
4      $recActs$.add(addChannel($node$)) ;
5      analyzeTaskHistory($node$);
6      for $neighbor \leftarrow$ affectedNeighbors $(node)$ do
7          if (rankTasks $(node) > \vartheta_{pref}) \vee (p < \vartheta_p)$ then
                                      /* root cause: sink behavior */
8              $recActs$.add(redDeleg($neighbor$))
               $recActs$.add(ctlCapacity($neighbor, node$))
9          end
10         else if $(q < \vartheta_q) \wedge (d > \vartheta_d)$ then
                                   /* Root cause: factory behavior */
11             $recActs$.add(ctlCapacity($neighbor, node$));
12         end
13         else
                              /* Root cause: transient degradation */
14             $recActs$.add(redDeleg($neighbor$))
15         end
16     end
17     return $recActs$
18 end
```

The last assists neighbors by providing new delegation channels to alternative nodes.

A loop-style data-flow allows observing changes between guarded system and the self-adaptation mechanism. Changes leading to possible misbehavior are recognized by the mechanism by directing the data-flow through the trigger modules' logic. In Algorithm 3.1 Trigger `triggerQueueOverload` filters events which indicate a threshold violation of the task queue capacity of a node (Line 4). Such an event causes `triggerQueueOverload` to fire the related diagnosis `diagnoseBehavior` passing on the failure affected $node$ information. E.g., in Figure 3.4 the congestion of node b is reported as such an event.

As a first precaution in `diagnoseBehavior` in Algorithm 3.2 balances the load at $node$ and adds recovery action `addChannel` to the recovery result-set $recActs$. The idea is to relieve $node$ by providing $node$ with new delegation options to nodes with sufficiently free capacities. The task of this recovery action is to discover a node that has capabilities similar to $node$. Once the delegation channel is added, in `ctlCapacity` method $estimateCapacity$ estimates the maximum possible of task transfer regarding the discovered nodes' processing capabilities. Finally, $setCapacity$ controls the throughput accordingly. Next, in $analyzeTaskHistory$ the diagnosis derives a root cause from the reported node's task history. A repository of classified failure patterns is compared to the last behavior patterns of the node and the corresponding root cause returned. In a loop (line 6), by selecting the affected neighbors, behavior is analyzed.

Algorithm 3.3: Recovery actions.

```
                        /* Recovery action:  control capacity */
Recovery Action ctlCapacity(neighbor, node) ;
begin
    cap ← estimateCapacity(neighbor, node);
    setCapacity(cap);
end
                        /* Recovery action:  control capacity */
Recovery Action addChannel(node);
begin
    $simNode$ ← lookupNodeSameCapabilities (node);
    addDelChannel(node, $simNode$);
    ctlCapacity(node, $simNode$);
end
                     /* Recovery action:  redirect delegations */
Recovery Action redDeleg(neighbor);
begin
    $simNode$ ← lookupNodeRequiredCapabilites(neighbor);
    addDelChannel(neighbor, $simNode$);
    ctlCapacity(neighbor, $simNode$);
end
```

Sink Behavior. Algorithm 3.2 Line 7 identifies sink behavior. The result of the pattern analysis shows that $node$ is still accepting tasks from any neighbor, however, prefers to work on tasks of a certain neighbor and delays the tasks of the other nodes. The second misbehavior of a sink is to perform tasks below an expected rate ($p < \vartheta_p$). The additional counter actions try to provide options for the set of affected delegating neighbor nodes and to decouple the sink. Recovery action `redDeleg` finds the alternatives and again estimates the adequate capacity of the new delegation channels. Recovery action `ctlCapacity` sets the delegation rate between sink and its neighbors to a minimum. The situation is depicted in Figure 3.4. Delegation channel (ii) is added from b to similar capable node d and allows b dispensing a certain amount of capability matching tasks.

Delegation channel (iii) from a to d is a result of `redDeleg`. In the example, d has enough resources to process blocked (from b) and new tasks (from a). The amount of recently delegated tasks is balanced in `estimateCapacity`. Thereafter the capacity of delegation channel (i) is minimized. A limitation of the delegations depends on the content of b's task queue. The example assumes that it mostly contains tasks from a. If the capacity of delegation channel (iii) is too low for a's delegation requirements, it might consider to process the tasks itself, or discover an additional node for delegation. The whole scenario is also applicable for a factory behavior of a. In that case, further uncontrolled delegations of a are avoided and no new delegation channel (iii) would be added.

Factory Behavior. Algorithm 3.2 Line 10 detects a delegation factory behavior. A factory is identified by moderate use of queue capacity ($q < \vartheta_q$) in contrast to high and exceeding delegation rates ($d > \vartheta_d$) causing overloaded nodes despite available alternatives. Recovery restricts the delegations from the factories to $node$, expecting that the factories start increasing their task processing performance or find themselves other nodes for delegations. Besides releasing the load from $node$, `ctlCapacity` ensures that the delegation of tasks from a factory to $node$ is set to a minimum.

Transient Behavior. In Algorithm 3.2 Line 13, if neither factory nor sink behavior are recognized `diagnoseBehavior` must assume a temporal overload of $node$. As a second precaution the algorithm estimates alternative delegation nodes in `redDeleg` for the neighbors of $node$.

After outlining a framework that integrates the algorithm the next chapter presents experiments that evaluate the effectiveness of previously presented recovery action algorithms in a simulated mixed SOA environment.

3.6 Conclusion

This chapter represents a first analysis of misbehavior affecting Mixed Systems. The misbehavior considered regards delegations as present in OESs. For example, OESs hosting communities of experts can easily become victims to misbehavior in delegations. The impact considered in this chapter is an unbalanced flooding of a small group of experts with tasks while the majority remains idle. Affected are typically the nodes that are at the end of a delegation chain and not as well connected as the established ones. This led further to the assumption that this will concern in particular the newly joint nodes that have explicit incentives to accept tasks to increase their reputation rather quickly. In contrast, established members might prefer to delegate some of the activities, however, also unintentionally overload the over-ambitious members.

Towards a solution of the problem, at the beginning a network model that abstracts the situation in these networks is detailed. Basing on two selected common delegation misbehavior models the characteristics of *Delegation Factory* and *Delegation Sink* behavior are presented. The factory behavior is the result of an over delegation at a node denoted as *Delegation Factory*. The sink behavior is caused by a too ambitious and now overburden node usually at the end of a delegation chain; the *Delegation Sink*. A modular algorithm for self-adaptation demonstrates how both misbehavior models can be detected and resolved in a network of nodes. It is clear that a Mixed Systems can expose many other possible forms of misbehavior; in particular including SBSs. However, here SBSs are considered only with a minor involvement as rather reliable, redundant collaboration resources. As the next chapter will highlight in the experiments, the impact of such common types of delegation misbehavior on HPSs can already degrade sensibly the throughput of the system when left unattended.

CHAPTER 4

VieCure: An Adaptation Framework for Mixed Systems

4.1 Chapter Overview

Following the idea of misbehavior monitoring and adaptation presented in the previous chapter, the studies in this chapter apply the notion of self-adaptation to systems comprising a mixture of human and traditional service interactions. In particular, at the center of the discussions is the model of a framework with self-adaptation properties suitable for SOA and Mixed Systems. The presented *VieCure* Framework is such an extensible adaptation framework. It combines the necessary components to assemble an adaptation-loop and the interfaces to integrate with an SOA environment. As before, the situation in the observed Mixed Systems and their successful operation mode largely depends on the collaboration behavior of the involved humans. Thus, a particular extension highlighted is the integration of a behavior registry.

Chapter Outline. In Section 4.2 the framework scope is detailed and the frameworks technical details are given in Section 4.3. Afterwards, in Section 4.4 the adaptation algorithm of Chapter 3 is integrated and evaluated in a simulated Mixed System environment affected by the studied misbehavior patterns.

4.2 The *VieCure* Framework: Overview and Scope

Continuing the studies with the model of a Mixed System basing on an SOA infrastructure, Figure 4.1 outlines the alignment of an adaptation framework to a Mixed System environment comprising three main building blocks: the *Mixed System Environment* consisting of human-provided and software services, the *Monitoring and Adaptation Layer* the interface to observe and control the actual environment and the *VieCure Framework* detailed later and providing the main features to support self-adaptation actions.

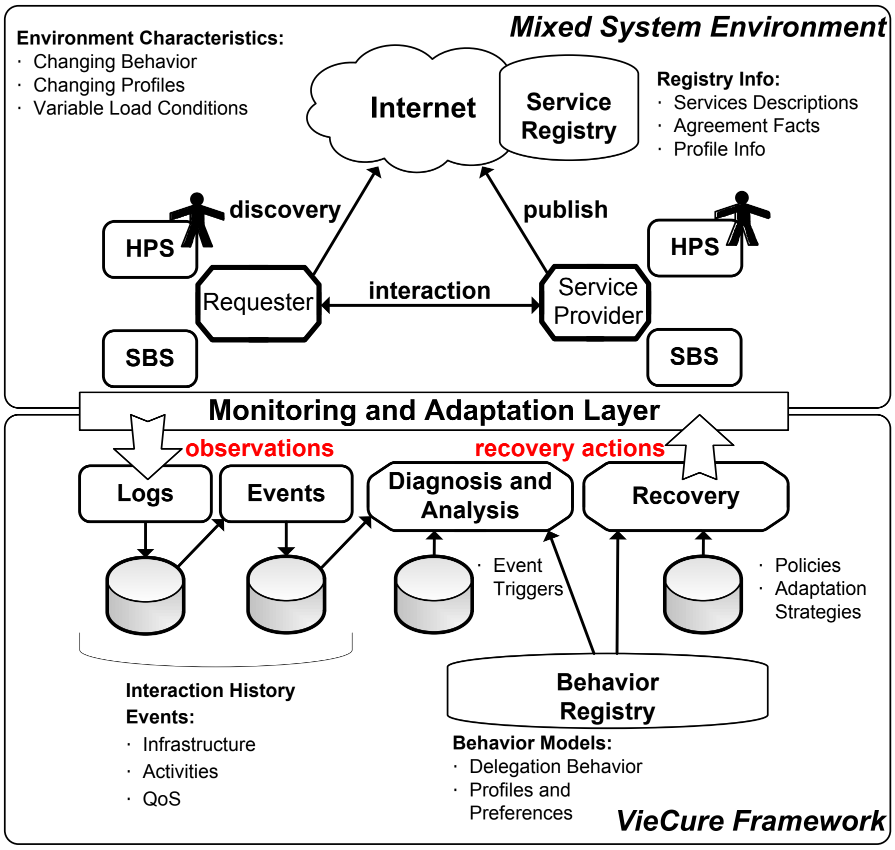


Figure 4.1: Environment overview and the *VieCure* framework.

Alignment with a Mixed System Environment

Many collaboration and composition scenarios involve interactions spanning human actors as well as software services. Traditional SOA architectures were designed to host SBSs without considering HPSs. The architectural model is extended by introducing:

- A service registry maintaining information related to human and software services.
- The definition of interaction patterns and interaction constraints using WS technology.
- Enhanced service-related information by describing human characteristics and capabilities.

The resulting *environment characteristics* are dynamic, because of changing behavior and skill profiles. There is a need for adaptation mechanisms due to variable load conditions (e.g., changing availability of human actors and changing amount of task that need to be processed).

Monitoring and Adaptation Layer

This layer maintains a self-adaptive extension of the aforementioned environment. It is ideally hosted, managed, and configured by the infrastructure provider, however, modular as depicted and open to potential external customization via interfaces, e.g., WS interfaces. Its main building block is a *feedback loop* enabling adaptation of Mixed Systems. The functions of a feedback loop are realized as a *MAPE-K* cycle (Monitor, Analyze, Plan, Execute, and K denoting the Knowledge) [41]. Therefore the architecture needs to integrate the functions of the loop by performing two essential steps:

Observations. Part of the knowledge base is provided by observations. Observations constitute most of the current knowledge of the system. Interaction data is gathered from the Mixed System environment and stored in the logging database (denoted as *Logs*).

Events. are registered and captured in the environment, stored in historical logs, and serve as input for triggers and the diagnosis.

Recovery Actions. By filtering, analyzing, and diagnosing events, an adaptation may need to be performed. Recovery actions are parts of a whole adaptation plan determined by diagnosis. Single recovery actions are deployed in correct order and applied to the environment by the *Recovery* module.

4.3 *VieCure* Framework

The building blocks of the *VieCure* framework are detailed in the following. Figure 4.2 shows the fundamental interplay of *VieCure*'s components. The *Monitoring and Adaptation Layer* is the interface to the controlled environment that is observed by the framework and influenced afterward through corrective actions. All monitored interactions, such as SOAP-based task delegations (see Listing 4.1), are stored for later analysis by *Interaction Logging Facilities*. Environment events, including adding/removing services or state changes of nodes, are stored by

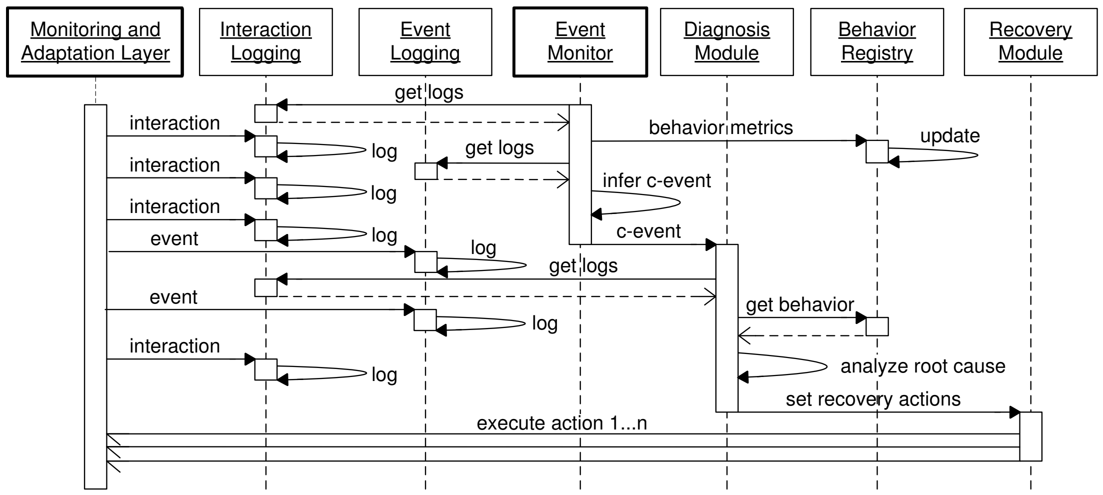


Figure 4.2: *VieCure*'s fundamental mode of operation.

similar *Event Logging Facilities*. Logs, events, and initial environment information represent the aggregated knowledge used by the *VieCure* framework to apply self-adaptation mechanisms. The effectiveness and accuracy of the adaptation techniques strongly depend on data accuracy.

The *Event Monitor* is periodically scheduled to collect recent interactions and events from the logging facilities. Upon this data, the monitor infers higher level composite events ($c-event$). Pre-configured triggers for such events, e.g. events reporting agreement violations, inform the *Diagnosis Module* about deviations from desired behavior. Furthermore, the actual interaction behavior of nodes is periodically updated and stored in the *Behavior Registry*. This mechanism assists the following diagnosis to correlate behavior changes and environment events. Furthermore, profiles in conjunction with the concept of HPSs allow categorizing these services and determine root causes.

Once a deviation indicating composite event triggered the *Diagnosis Module*, a root cause analysis is initiated. Previously captured and filtered interaction logs, as well as actual node behaviors, assist a sophisticated diagnosis and to recognize the mixed system's current state. On failures a set of corrective recovery actions is submitted to the *Recovery* module. A substantial part of recovery is the policy registry (underneath the *Recovery* block in Figure 4.1). It manages available adaptation methods. Adaptations and constraints applied by policies include, for example, boundaries and agreements imposed on the services defining the interaction paths and limiting recovery strategies. The recovery module executes the recovery actions and influences the mixed system environment through the *Monitoring and Adaptation Layer*.

Interaction Monitoring

Interactions between community members of Mixed Systems such as the expert network (c.f., chapter 3) are modeled as standardized SOAP messages with header extensions, as shown in Listing 4.1. A logging service is part of the monitoring layer to capture all interactions performed in the network. Header extensions include the context of interactions (i.e., the activity that is performed), delegation restrictions (e.g., the number of hops), identify the sender and receivers with WS-Addressing [118], and hold some meta-information about the activity type itself. For HPSs, SOAP messages are mapped to user interfaces by the HPS framework *Task Context* related information is also transported via header mechanisms. While activities depict what kind of information is exchanged between actors (type system) and how collaborations are structured, tasks control the status of interactions and constraints in processing certain activities.

Event Trigger, Diagnosis and Recovery Actions

The event monitor is an integral part of the monitoring layer. As previously described it constantly logs arriving events from the environment and composes log and event history to higher level events. Events from the environment are delivered by a reliable and asynchronous event bus provided by, e.g., the Java Message Service (JMS)[1].

The structure of an event as payload of a message or composed by the event monitor is provided by the XSD-based definition in Listing 4.2. The initial four fields identify the event

[1]http://java.sun.com/products/jms/

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:vietypes="http://www.infosys.tuwien.ac.at/Type"
 xmlns:hps="http://myhps.org/Type"
 xmlns:hpsht="http://myhps.org/HumanTask"
 <soap:Header>
  <wsa:MessageID>uuid:722B1240−...</wsa:MessageID>
  <wsa:ReplyTo>http://www.expertweb.org/Actor#Harald</wsa:ReplyTo>
  <wsa:From>http://www.expertweb.org/Actor#Harald</wsa:From>
  <wsa:To>http://www.expertweb.org/Actor#Florian</wsa:To>
  <wsa:Action>http://myhps.org/Action/Delegation</wsa:Action>
  <vietypes:activity url="http://www.expertweb.org/Activity#42"/>
  <vietypes:delegation hops="3"/>
  <vietypes:timestamp value="2010-01-29T15:13:21"/>
  <hpsht:taskContext>
   <hpsht:deadline="2010-01-30T12:00:00"/>
   <hpsht:priority>...</hpsht:priority>
  </hpsht:taskContext>
 </soap:Header>
 <soap:Body>
  <hps:prepReport>
   <prepReport:requ>Please create a report for experiment X</prepReport:requ>
   <prepReport:generalterms>algorithm</prepReport:generalterms>
   <prepReport:keywords>ranking, interactions, graph</prepReport:keywords>
   <prepReport:resource url="http://.../experimentX"/>
  </hps:prepReport>
 </soap:Body>
</soap:Envelope>
```

Listing 4.1: Simplified interaction example.

at the receiver (fist two) and sender (last two), if arriving from the environment. The tuple `sequence number` and `time` uniquely identify an event at both sides. This also supports examinations on the events actuality. The following fields `origin`, `type`, `extended type`, and `description` are mandatory. The `origin` indicates the source of the event. These in-

```
 <complexType name="Event"> <sequence>
  <element name="logSeqNumber" type="int"></element>
  <element name="logTime" type="time"></element>
  <element name="eventSeqNumber" type="int"></element>
  <element name="eventTime" type="time"></element>
  <element name="eventOrigin" type="string"></element>
  <element name="eventType" type="tns:EventType"></element>
  <element name="eventExtendedType" type="tns:ExtEventType">
  </element>
  <element name="eventDescription" type="string"><element>
  <element name="eventSeverity" type="tns:Severity"></element>
   ...
  </sequence>
</complexType>
```

Listing 4.2: Extract of event specification.

```
rule "TriggerOverDelegate"
    when
        node:Node(delegationRate > 50 && role == "worker")
        recoveryActionList:ArrayList()
    then
        Node neighbor = Utils.lookupNodeSimilarCapabilities(node)
        RecoveryAction ctlCapacity = new CtlCapacity(neighbor, node);
        recoveryActionList.add(ctlCapacity);
        ...
end
rule "TriggerUnderDelegate"
    when
        node:Node(numTasksQueued > 15 && delegationRate < 2)
    then ...
end
```

Listing 4.3: Triggering events and setting recovery actions.

clude environment or composed type. The `extended type` field tags the events nature. Tags reflect hardware and communication faults, human related workload and delegation problems, and QoS and agreements related issues. The `description` field contains a human readable description of the event. This is included for offline evaluation and or online test runs assisted by humans. The final required field of the schema is the event's severity. The `severity` defines the events queuing priority and processing urgency.

JBoss Drools[2] platform provides the rule engine to define policies that detect unexpected behavior of nodes. Multiple rules can be defined to trigger when behavior changes and leads to problems. In the following the delegation misbehavior models of the previous Chapter are referred for the example. Listing 4.3 shows an excerpt of rule definitions to detect suspiciously increasing or decreasing delegation behavior that indicate the misbehavior in the environment. In `TriggerOverDelegate` if the `delegationRate` exceeds a predefined value for a particular participants role the node is identified and its delegation capacity `RecoveryAction ctlCapacity` reduced. Furthermore, `recoveryActionList` provides an array that can be filled with additional actions and deployed in sequence. In `TriggerUnderDelegate` covers the case of decreasing delegation related to a considerably filled task queue (`numTasksQueued`). This two briefly introduced cases of misbehavior are discussed in detail in the next chapter.

The final step in the adaptation process is to execute recovery actions. Listing 4.4 shows an

```
<ControlAction xmlns="http://myhps.org/Action"
  xmlns:vietypes="http://www.infosys.tuwien.ac.at/Type"
  xsi:type="Coordination" URI="Coordination#10" Activity="Activity#42">
  <From>http://www.expertweb.org/Actor#Harald</From>
  <ActionType>http://myhps.org/Action/Delegation</ActionType>
  <To>http://www.expertweb.org/Actor#Florian</To>
  <vietypes:ctlCapacity capacity=.../>
</ControlAction>
```

Listing 4.4: Control action to recover from degraded system state.

[2]http://jboss.org/drools

example how such recovery actions can be performed in the system. As mentioned previously, an approach for recovering from degraded system state is regulation of delegation behavior between actors (integrated as HPSs). This is accomplished by sending the corresponding recovery action to an *Activity Management Service* (see [89] for details). In Listing 4.4, a `ControlAction` of type `Coordination` is depicted regulating the flow of delegations between two actors. Each `Coordination` action has a unique identifier and is applied in the context of an activity. The `ControlAction` also contains what kind of `ActionType` has to be regulated as a result of a recovery. In this example regulation applies to degenerated `Delegation` actions by changing the capacity of delegation channels.

4.4 Integration Model and Simulation Evaluation

Generally, one of the major concerns in extending systems with self-* properties for systems is an adequate adoption with existing and running systems (c.f., "Intrusiveness" in Section 2.5 in the state-of-art). Thus, a fundamental idea of the adaptation loop setup in this thesis is a decoupled double (online-offline) loop approach. Instead of aligning the observation/adaptation loop to the observed environment only (*online*) an additional verification loop (*offline*) is arranged in-between. The extension supports the analysis step with a simulation environment mirroring the situation of the real environment that is "adapted" first. The main advantage of such an integration of a simulated environment is an a-priori verification of the adaptation's side-effects without interfering with the real system.

Environment Integration: Offline - Online Adaptation Approach

Figure 4.3 details the concept. At the center the *VieCure* Framework with *Adaptation Strategies* and *Observations* is in place to collect the messages of the interactions in the running Mixed System (in the example SOAP-based). With this information the *Simulated Interaction Network* on the left of the framework is updated. The data runs also through the framework's analysis and possible necessary adaptations are planned. At this point, however, adaptation actions are

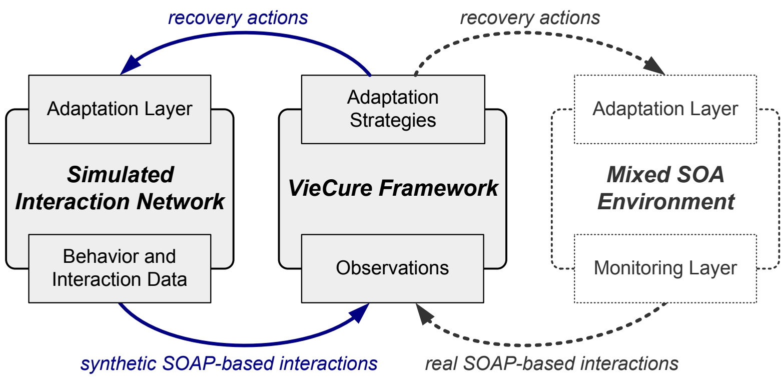


Figure 4.3: Integration Model.

deployed to the simulated network first. The effects can be evaluated by observing the synthetic interactions happening in the simulated network. Finally, if the actions' results are successful the adaptations can also be deployed to the real environment.

One might argue that this procedure takes a substantial additional amount of time and effort until the adaptive actions are applied to the real system. In large-scale systems, because of their vast number of participants a malfunction of one node (misbehavior in this context) in the majority of cases does not result in a total system stall. Instead, the system turns to an intermediate state identified as *Degraded State* [31]. A system in this state continues running and provides recovery algorithms time to deploy effective adaptations. As emphasized previously, Mixed Systems are large-scale systems and, i.e. they fit this concept.

Simulated Heterogeneous Service Environment

The simulation presented exploits the idea that data is derived from a Mixed SOA Environment and allows to reconstruct the main environment characteristics. The simulated interaction network comprises a node actor framework implemented in JAVA language. At bootstrapping the nodes receive a profile including different behavior models. Each node has a task list with limited capacity. Depending on the deployed behavior model a node tends either to delegate, or process tasks, or exposes a balanced behavior. New tasks are constantly provided to a quarter of the nodes via connected entry points. Tasks have an effort of three units. A global timer initiates the simulation rounds. Depending on the behavior model, in each round a node decides to process tasks or delegate one task. A node is able to process the effort of a whole task, or if delegating, only one effort unit. For the delegation activity a node holds a current neighbor list which is ordered according to the neighbors' task processing tendency. The delegating node prefers nodes with processing behavior and assigns the selected the longest remaining task. A receiving node with a task queue at its upper boundary refuses additional tasks. However, each task is limited by a ten round expiry. If a task is not processed entirely in this period it is considered a *failed task*.

VieCure Framework Integration

At bootstrapping the *VieCure* Framework's monitoring and adaptation layer is instantiated. The monitor has an overview over all nodes in the simulated environment. Thus, the monitor provides the *VieCure* Framework with a current node list together with their task queue levels. A trigger filters the queues' levels and reports to diagnosis if the lower threshold value is exceeded. Diagnosis estimates then the actual level and decides on the recorded history together with the current situation which recovery action to choose. For the purpose of the evaluation of the recovery actions, diagnosis is required to act predictable and decide according to the configuration which recovery action to select.

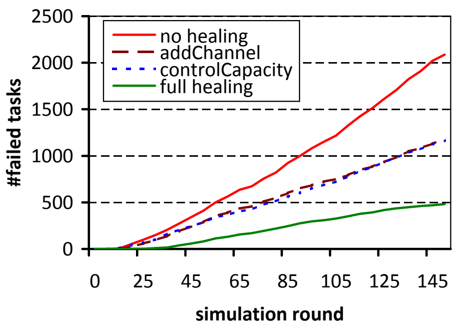
In the line with the work of the previous chapter and the behavior regulation algorithms (c.f., Section 3.5), for the evaluation two of the recovery actions were implemented. In *control capacity*, the delegation throughput to the affected node is adapted according to the current task queue level. In *add channel*, the filtered node is provided with a new channel to the node with the currently lowest task queue load factor. In order to evaluate the effects of the recovery actions, four different runs with the same setting are executed. At the end of each experiment the logging facilities of the framework provided all the information needed for analysis. The results are presented next.

Results and Discussion

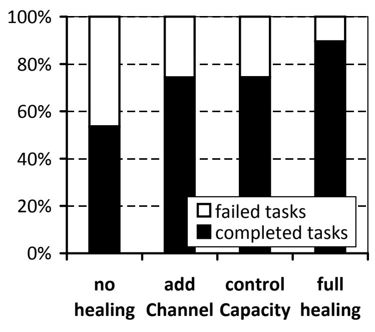
The experiments measure the efficiency of a recovery action by the amount of *failed tasks*. An experiment consists of a total number of 150 rounds and a simulation environment with 128 nodes. During an experiment 4736 tasks are assigned to the nodes' network. In order to prevent an initial overload of a single node as a result of too many neighbor relations, the amount of incoming delegations channels is limited to a maximum of 6 incoming connections at start-up. The resulting figures present on their left the total of failed tasks after a certain simulation round. The curves show the progress of different configurations of *VieCure's* diagnosis module. The figures on the right represent the ratio failed/processed tasks in percentages at the end of the experiments with an equal setting.

The setting for the results in Figure 4.4 consisted of an equal number of the three behavior models distributed among the nodes. Whilst the nodes on their own produce a total of 2083 failed tasks (top continuous curve) the two different recovery actions separately expose an almost equal progress and finish at almost half as much; 1171 for *add channel* action and 1164 for *control capacity* action, respectively. Combining both diminishes the failure rate to a quarter compared to no action, to 482 failed tasks (lower continuous curve). The results demonstrate that in an equilibrated environment the two recovery actions perform almost equal and complete each-other when combined.

In Figure 4.5 the setting configured a tenth of nodes with factory tendency and an equal distribution of the other two models across the remaining nodes. An immediate result of the dominance of task processing nodes is that less tasks fail generally. The failure rate for the experiment with no recovery falls to a total of 1693 (top continuous curve). The success of *add channel* (dashed curve) remains almost the same (1143). With this unbalanced setting the potential neighbors for a channel addition remain, however, the same as in the previous setting.

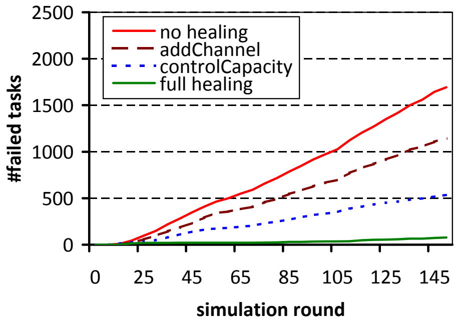


(a) Current failure rate.

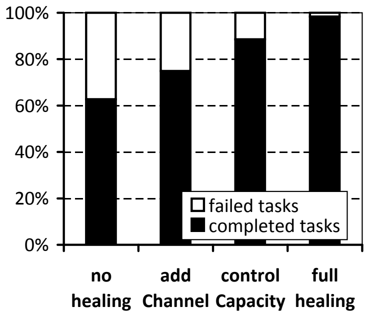


(b) Final overall success rate.

Figure 4.4: Equal distribution of behavior models.



(a) Current failure rate.

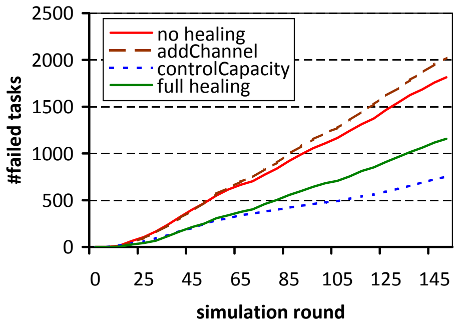


(b) Final overall success rate.

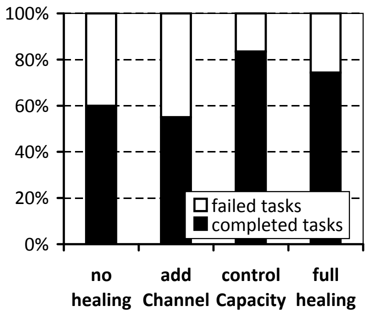
Figure 4.5: Distribution with a trend for 10% factory behavior.

In contrast, the success of *control capacity* (dotted curve, 535) relies on the fact that regulating channels assures that the number of tasks in a queue relates to the task processing capabilities given by a node's behavior. In strategy combination (lower continuous curve, 77), this balancing mechanism is supported by additional channels to eventually still failing nodes. The results are also reflected by the success rate figure.

In Figure 4.6 the setting was changed to a 10% of sink behavior trend. Without a recovery strategy the environment performs almost the same as in the previous setting (top continuous curve, 1815). The strategy of just adding channels to overloaded nodes fails. Instead of relieving nodes from the task load, tasks circle until they expire. Thus, a number of 2022 tasks fail for *add channel* (dashed curve). The figure further shows, that this problem has also impact on the combination of the two strategies (lower continuous curve, 1157). The best solution for the setting is to inhibit the dominating factory behavior by controlling the channels capacity (dotted curve, 753).



(a) Current failure rate.



(b) Final overall success rate.

Figure 4.6: Distribution with a trend for 10% sink behavior.

4.5 Conclusion

This chapter presented the *VieCure* Framework, a framework model that extends Mixed Systems with self-adaptation properties. The technical integration with the system is detailed by samples and examples of interaction protocols, event formats, and rule language. Apart from the essential modules of a self-adaptation framework a particular addition to the adaptation-loop is the behavior registry. This registry is essential for the adaptation of human misbehavior. It registers the actual interaction behavior of nodes and periodically updates and stores the information. It allows to recognize changes in behavior of the nodes and derive root causes for misbehavior.

The evaluations sections present initial results of the adaptation algorithm detailed in Chapter 3. The simulations assume that real interaction data is available from an interaction log collection of a running Mixed System. In an offline procedure a simulation can demonstrate the efficiency of the adaptation strategies. The evaluations show that the recovery actions compensate satisfactorily the misbehavior in a Mixed System (about 30% higher success rate with equal distribution of behavior models). The success rates of the recovery actions depend on the environment settings. In all but one of the cases, deploying recovery actions supports the overloaded nodes resulting in a higher task processing rate. Important to note, that the failure rate increases near linearly even if recovery actions adjust the nodes' network structure. The observation emphasizes the effectiveness of applying non-intrusive recovery strategies to Mixed Systems.

The main purpose of the evaluation was to demonstrate the applicability of the implemented adaptation algorithms. It is obvious that details of the integration have only been discussed at a modular and interface level. Also, the *VieCure* Framework presented in this thesis is considered a central system and combines all components necessary for adaptation. The question about scalability remains unanswered and must be considered in future work. Regarding the framework design, the next chapter introduces an approach with a partial distributed adaptation framework to emphasize the modularity of the framework.

Part II

SOA Integration and Adaptation Strategies

CHAPTER 5

Testbed Integration: Behavior Monitoring and Similarity Based Adaptation

5.1 Chapter Overview

After the preliminary results of adaptation with a simulated SOA environment presented in the previous chapter, this chapter dives into the particularities of the integration of *VieCure* with a real SOA testbed. The main presented contributions are as follows: The particularities of the framework integration with the testbed are outlined. This includes the monitoring of interaction data and the deployment of adaptation actions in the line with the testbed's programming model. Another integration work shows, how the traditional service model of the testbed can be extended to simulate arbitrary human behavior. Regarding the adaptation algorithm, while still considering factory and sink behavior as the sources of misbehavior, a new approach towards a resolution by service replacement is taken. By introducing the notion of profile similarity and dynamic trust the integration of an external similarity service is described to outline the extensibility of the *VieCure* Framework.

Chapter Outline. In Section 5.2 the Genesis2 testbed generator framework (G2) is introduced. Then the chapter's approach of integration is outlined. Section 5.4 provides the details. The following section gives insight into the applied trust metrics (Section 5.5). Next, behavior monitoring and self-adaptation for the misbehavior models defined in Chapter 3 are explained in Section 5.6. Before the evaluations in Section 5.7 a section explains G2 behavior simulation capabilities in Section 5.4.

5.2 Genesis WS Framework

In the field of runtime evaluation and simulation, there have been several approaches which could support testing of adaptation mechanisms. SOABench [10] and PUPPET [9], for instance, support the creation of mock-up services in order to test workflows. However, these prototypes are restricted to emulating non-functional properties (QoS) and cannot be enhanced with behavior simulations as required in the present studies. This issue had been partially solved in the first version of Genesis [46]. The key to this was to provide an extensible testbed with a plugins interface. With the current version Genesis2 [45] which allows extending testbeds with fine granular plugins diverse adaptation mechanisms can be implemented, applied, and tested. The purpose of the Genesis2 testbed generator framework (G2) is to provide all means to host an SOA-based environment including Mixed Systems.

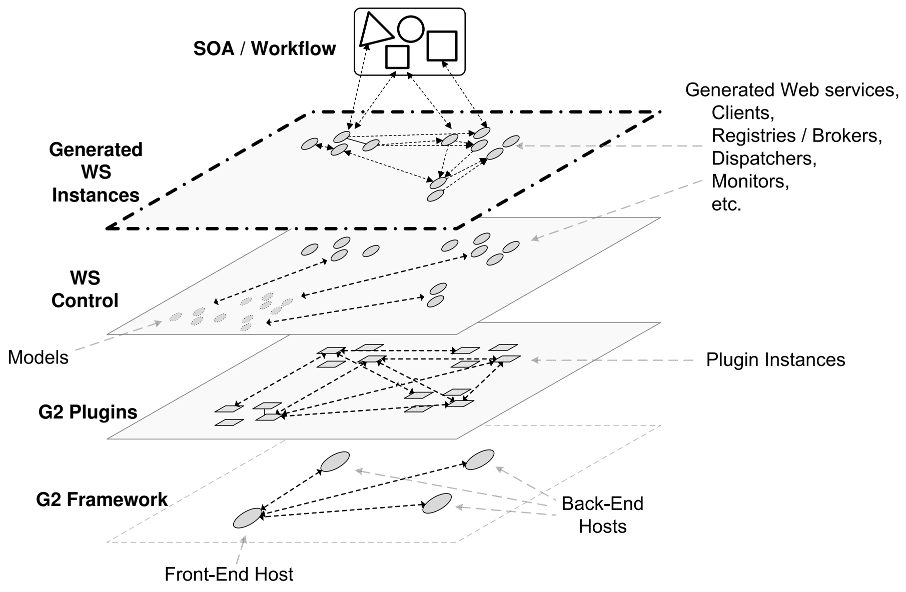


Figure 5.1: Layered G2 topology.

Figure 5.1 shows G2's layered topology consisting of various elements. This layered topology enables software engineers to generate and control SOA environments. The top layer displays the runtime with the *Generated WS Instances* including services, clients for service, registries, etc. The next layer enables control over the generated instances. The *WS Control* layer comprises a comprehensive model of the deployed environment. It allows steering the execution of the instances and propagating any model changes back to the environment. The following *G2 Plugins* layer allows dynamically contributing external extensions. Specific environment setups can be designed. In this context for example, plugin extensions allow modifying the ready to extend traditional G2 service definition, i.e. the SBSs, to create simulated HPSs. Plugins provide the means of integrating studied human behavior models in the service definition at design time and of control to these behavior models at runtime.

5.3 Approach Outline

Following the model of the twofold offline - online adaptation strategy describe in Section 4.4 in this section an integration of a real testbed represented by the *G2* with the model is presented. Figure 5.2 gives an overview of the integration approach.

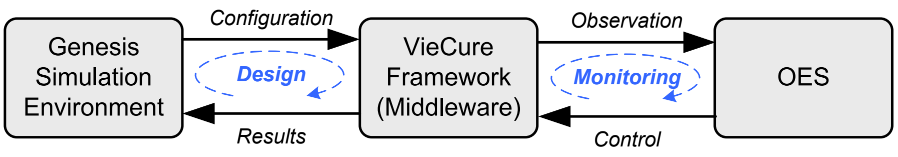

Genesis Simulation Environment
Configuration
Design
Results
VieCure Framework (Middleware)
Observation
Monitoring
Control
OES

Figure 5.2: Approach outline and contributions.

The fundamental parts involved are:

- *Open Enterprise System.* The collaborative community model introduced in Section 2.3 basing on a Mixed System that hosts both, SBSs and HPS interacting to perform joint activities.

- *VieCure as Middleware.* The framework as detailed in Section 4.3 is extended with a programming model that provides monitoring and adaptation mechanisms and can function as a middleware. These control the OES *and* the simulation environment. Service policies to regulate behavior (interaction dynamics) are based on observations and control.

  Adaptation strategies can be deployed in the G2 for testing purposes. Furthermore, logs, from either the simulation or the life OES can be analyzed to customize configurations and adaptation strategies.

- *G2 Simulation Environment.* The simulation environment allows investigating the effects of policies and adaptation strategies.

5.4 Adaptation Framework Integration

This section provides an overview of the components and services that allow simulations and tests of adaptation scenarios in collaborative service-oriented systems such as OES. The integration architecture (see Figure 5.3) consists of two main building blocks: the *testbed runtime environment* based on the Genesis2 framework [45] and the *VieCure* adaptation framework, adopted from the previous description in Chapter 4.

Testbed Generator Framework

As depicted in Figure 5.3, the G2 framework comprises a centralized front-end, from where testbeds are modeled and controlled, and a distributed back-end at which the models are transformed into real testbed instances. The front-end maintains a virtual view on the testbed, allows

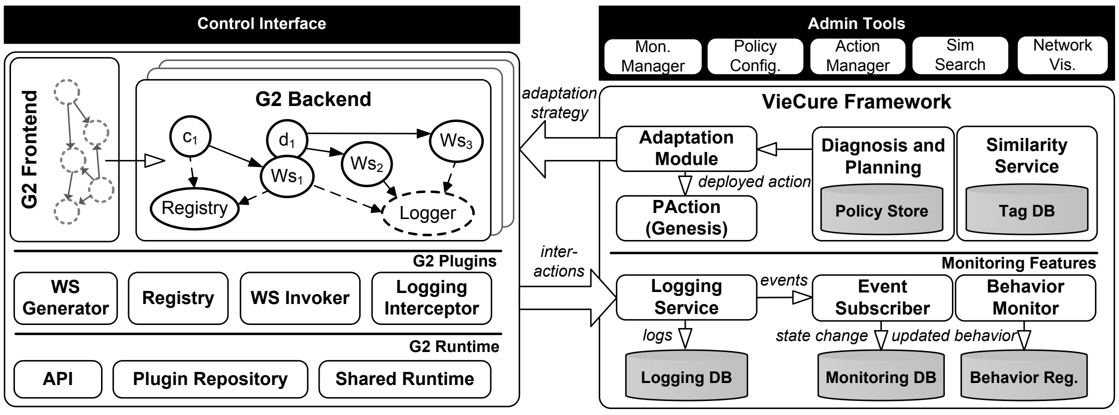


Figure 5.3: Architecture of the integration

engineers to manipulate it via Groovy[1] scripts, and propagates changes to the back-end in order to adapt the running testbed. To ensure extensibility, G2 follows a modular approach where a base runtime framework provides a functional grounding for composable plugins. These augment the testbed's functionality, making it possible to emulate diverse topologies, functional and non-functional properties, and behavior. Furthermore, each plugin registers itself at the shared runtime in order offer its functionality via the framework's script API.

The sample script in Listing 5.1 demonstrates a specification of a Web service which queries a registry plugin, applies a delegation strategy, and forwards the request message to a worker service. First, a call interceptor is created and customized with a Groovy closure which passes the SOAP message to the logger plugin. The model `datatype` allows importing an external complex data type with the method *create()*. The statement in line 7 shows the use of the `webservice` model to define an array of WS bodies with different properties and operations. The proxy service `Proxy` first attaches the created call interceptor to itself and defines an operation which delegates the request. This procedure is split into querying the registry for tagged Web services, applying the delegation strategy (`dStrat`) for determining the destination, and invoking the `Process` operation on it. For later adaptations, the delegation behavior itself is not hardcoded into the operation but outsourced as a service variable containing the delegation code. This makes it possible to update the deployed service's behavior at runtime by replacing the variable. Finally, Lines 24 and 25 demonstrate how a host is set-up with the `host` model and then deployed with the `webservice` model's *deployAt()* method. For a more detailed description of G2 and its model readers are referred to [45, 110].

Simulation of Human Behavior

The availability of rich and plentiful data on human interactions in *social networks* has closed an important loop [51], allowing to model social phenomena and to use these models in the design of new computing applications such as OESs. One of the key questions in social dynamics

[1]`http://groovy.codehaus.org/`

```
 1  li=callinterceptor.create() // logging interceptor
 2  li.hooks=[in:"RECEIVE", out :"PRE_STREAM"] // bind to phases
 3  li.code={ctx −> logger.logToDB(ctx.soapMsg) } // process msg
 4
 5  msgType=datatype.create("file.xsd","typeName") // xsd import
 6
 7  sList=webservice.build {
 8   // create web service
 9   Proxy(binding:"doc,lit", namespace="http://...") {
10    // attach logging interceptor
11    interceptors+=li
12    // create web service operation
13    Delegate(input:msgType, resonse:msgType) {
14     refs = registry.get{s−> "Worker" in s.tags} // by tag
15     r = dStrat(refs)
16     return r.Process(input).response
17    }
18    // delegation strategy as closure variable
19    dStrat={ refs −> return refs[0] } // default: take first
20   }
21  }
22
23  srv=sList[0] // only one service declared, take it
24  h=host.create("somehost:8181") // import back−end host
25  srv.deployAt(h) // deploy service at remote back−end host
26
27  srv.dStrat={ refs−> /∗...∗/ } // adapt strategy at runtime
```

Listing 5.1: Groovy script specifying a delegator service.

concerns the behavior of single individuals, namely how an individual chooses a convention, takes a decision, schedules his tasks and more generally decides to perform a given action. Most of these questions are obviously very difficult to address, due to the psychological and social factors involved [17]. These problems are also of particular interest in collaborative service environments, as their interaction patterns and usage dependent on human behavior. Instead of focusing on the individual's behavior in cooperative systems only (e.g., as discussed in [1]), the focus is on the *network effects* of human dynamics.

A unique feature of the presented G2 service hosting environment is its capability to not only be used as a hosting environment but also to run simulations of service environments as hinted in Figure 5.2 in the approach outline. Simulations can be used to evaluate the effectiveness of, e.g., the previously presented and defined policies, on varying environment conditions. As mentioned, there is a particular interest in service environments including human actors. In such simulations human behavior including making choices, taking decisions, working on tasks, or performing actions, need to be simulated with the help of previously observed and examined log data.

The G2 programming model introduced in Section 5.2 applies the concept of closures to equip services with individual behavior. Referring to lines 15, 19 and 27 in the sample script of Listing 5.1 it can be recognized that *dStrat* is defined as a global closure in the service model. While in the example the running service expects an executable algorithm, the *Proxy*'s strategy can be changed at any time during runtime by setting a new closure content to the global placeholder (line 27). Also, with this method the strategy can be set individually for the otherwise identical service definition. This makes the simulation more authentic.

Adaptation Framework

The adaptation framework in the line with the *VieCure* design, is located on the right side in Figure 5.3. The framework has monitoring features including logging, eventing, and a component for capturing actor behavior. Based on observations obtained from the testbed, adaptation actions are taken.

- The *Logging Service* is used by the logger plugin (see `PLogger` in Figure 5.3). Logged messages are persistently saved in a database for analysis. The logging service also implements a publish/subscribe mechanism to offer distributed event notification capabilities. Subscribers can specify filter using XPath statements which are evaluated against received logged messages.

  A short example is shown in Listing 5.2. Header extensions (Line 7 - 22) include the context of interactions (i.e., the activity that is performed), delegation restrictions, identify the sender and receivers using WS-Addressing [118] mechanisms, and hold some meta-information about the activity type itself. `MessageIDs` enable message correlation to correctly match requests and responses. `Timestamps` capture the actual creation of the message and are used for message ordering. For HPSs, SOAP messages are mapped to user interfaces by the HPS framework. `Task Context` related information is also transported via header mechanisms. While activities depict what kind of information is exchanged between actors (type system) and how collaborations are structured, tasks control the status of interactions and constraints in processing certain activities.

  Multiple instances of the logging service can be deployed to achieve scalability in large scale environments.

- *Event Subscribers* receive events based on filters that can be specified for different types of (inter-)actions, for example, to capture only delegation flows. Subscribers are used to capture the runtime state of nodes within the testbed environment such as current load of a node.

- The *Behavior Monitor* updates and stores periodically the actual interaction behavior of nodes as profiles in the *behavior registry*. This mechanism assists the following diagnosis to correlate environment events and behavior changes.

- *Diagnosis and Analysis* algorithms are initiated to evaluate the root cause of undesirable system states. Pre-configured triggers for such events, e.g., events reporting violations, inform the diagnosis module about deviations from desired behavior. Captured and filtered interaction logs as well as actual node behaviors assist in recognizing the system's health state.

- The *Similarity Service* uses the tag database to search for actors based on profile keywords (i.e., to replace an actor or to establish a new link to actors). Tags are obtained from logged interactions.

```
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 xmlns:xsi="www.w3.org/2001/XMLSchema-instance"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:hpsht="http://myhps.org/HumanTask"
 xmlns:vietypes="http://viete.infosys.tuwien.ac.at/Type"
 <soap:Header>
  <wsa:MessageID>052c5e11-5abd-4763-8725-86eaa48fb0fc</wsa:MessageID>
  <wsa:ReplyTo>http://www.expertnetwork.org/Actor#Harald</wsa:ReplyTo>
  <wsa:From>http://www.expertnetwork.org/Actor#Harald</wsa:From>
  <wsa:To>http://www.expertnetwork.org/Actor#Florian</wsa:To>
  <wsa:Action>http://myhps.org/Action/Delegation</wsa:Action>
  <vietypes:activity url="http://www.expertnetwork.org/Activity#42"/>
  <vietypes:timestamp value="2010-05-06T15:13:21"/>
  <hpsht:taskContext>
    <hpsht:deadline="2010-05-07T12:00:00"/>
    <hpsht:priority>
     <!-- task priority -->
    </hpsht:priority>
    <hpsht:keywords>WS, Adaptation, Trust</hpsht:keywords>
  </hpsht:taskContext>
 </soap:Header>
 <soap:Body>
   <hps:prepareReport>
    <!-- details omitted -->
   </hps:prepareReport>
 </soap:Body>
</soap:Envelope>
```

Listing 5.2: Simplified SOAP interaction example.

- The *Adaptation Module* deployes appropriate adaptation actions. An example for an adaptation action is to update a node's *delegation strategy* as indicated in Figure 5.3. For that purpose, the *PAction* plugin communicates with G2's control interface.

A set of *Admin Tools* offer graphical user interfaces for configuring and visualizing the properties of testbeds. User tools include, for example, policy design for adaptations or visualizations of monitored interactions.

5.5 Similarity Service: Profile Similarity and Dynamic Trust

This section gives a closer insight in the functionality of the `Similarity Service`. The integration of the service results from a collaboration with its authors and the section can only provide a quick introduction into the used functionality relevant for this scenario. The interested reader will find a more detailed description of all its concepts in [95]. OESs, as outlined in the previous chapters, are subject to trust studies. Over the last years, trust has been defined from several points of views [35]. However, until now, no agreed definition exists. Unlike a security view, in this context the trust focus is on the notion of dynamic trust from a social perspective [121]. The notion of trust [96] is based on the interpretation of collaboration behavior [96, 35] and dynamically adapting skills and interest similarities [33, 63]. The trust between community members is essential for successful collaborations.

Especially in collaborative environments, where users are exposed to higher risks than in common social network scenarios, and where business is at stake, considering trust is essential to effectively guide human interactions. The particular focus is on the establishment of trust through measuring interest similarities [96]:

- *Trust Mirroring* implies that actors with similar profiles (interests, skills, community membership) tend to trust each other more than completely unknown actors.
- *Trust Teleportation* rests on the similarity of human or service capabilities, and describes that trust in a member of a certain community can be teleported to other members. For instance, if an actor, belonging to a certain expert group, is trusted because of his distinguished knowledge, other members of the same group may benefit from this trust relation as well.

Interest Profile Creation

In contrast to common top-down approaches that apply taxonomies and ontologies to define certain skills and expertise areas, in this context a mining approach is followed that addresses inherent dynamics of flexible collaboration environments. In particular, skills and expertise as well as interests change over time, but are rarely updated if they are managed manually in a registry. Hence, they are determined and updated automatically through mining.

The creation of interest profiles without explicit user input has been studied in [96]. As discussed before, interactions, i.e., delegation requests, are tagged with keywords. As delegation receivers process tasks, the system is able to learn how well people cope with certain tagged tasks; and therefore, able to determine their centers of interests. It is assumed that tasks are aligned to keywords to create dynamically adapting interest profiles based on tags and manage them in a vector space model.

The utilized concepts are well-known from the area of information retrieval (see for instance [86]). However, while they are used to determine the similarities of given documents, the system creates these documents (that reflect user profiles) from used tags dynamically on the fly.

The profile vector $\mathbf{p_u}$ of actor u in Eq. (5.1) describes the frequencies f the tags $T = \{t_1, t_2, t_3 \ldots\}$ are used in delegated tasks accepted by actor u.

$$\mathbf{p_u} = \langle f(t_1), f(t_2), f(t_3) \ldots \rangle \tag{5.1}$$

The tag frequency matrix $\mathfrak{T}$ (5.2) in Eq. 5.2, built from profile vectors, describes the frequencies of used tags $T = \{t_1, t_2, t_3 \ldots\}$ by all actors $A = \{u, v, w \ldots\}$.

$$\mathfrak{T} = \langle \mathbf{p_u}, \mathbf{p_v}, \mathbf{p_w} \cdots \rangle_{|T| \times |A|} \tag{5.2}$$

The popular tf^*idf model [86] introduces tag weighting based on the relative distinctiveness of tags; see Eq. (5.3). Each entry in $\mathfrak{T}$ is weighted by the log of the total number of actors $|A|$, divided by the amount $n_t = |\{u \in A \mid tf(t, u) > 0\}|$ of actors who used tag t.

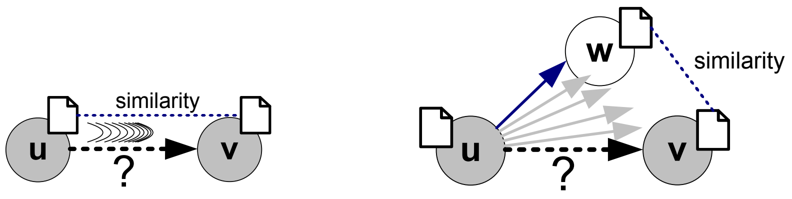
$$tf^*idf(t, u) = tf(t, u) \cdot \log \frac{|A|}{n_t} \tag{5.3}$$

Finally, the cosine similarity, a popular measure to determine the similarity of two vectors in a vector space model, is applied to determine the similarity of two actor profiles $\mathbf{p_u}$ and $\mathbf{p_v}$; see Eq. (5.4).

$$sim_{profile}(\mathbf{p_u}, \mathbf{p_v}) = \cos(\mathbf{p_u}, \mathbf{p_v}) = \frac{\mathbf{p_u} \cdot \mathbf{p_v}}{||\mathbf{p_u}|| \, ||\mathbf{p_v}||} \tag{5.4}$$

The Interplay of Interest Similarity and Trust

In the model, a trust relation $\tau(u, v)$ mainly relies on the interest and expertise similarities of actors. It is necessary to apply various concepts to facilitate the emergence of trust among network members.



(a) Trust Mirroring. (b) Trust Teleportation.

Figure 5.4: Concepts for the establishment of trust through interest similarities.

Trust Mirroring. Trust τ_{mir} (Figure 5.4a) is typically applied in environments where actors have the same roles (e.g., online social platforms). Depending on the environment, interest and competency similarities of people can be interpreted directly as an indicator for future trust (Eq. 5.5). There is strong evidence that actors 'similar minded' tend to trust each other more than any random actors [121,63]; e.g., movie recommendations of people with same interests are usually more trustworthy than the opinions of unknown persons. Mirrored trust relations are directed, iff $sim_{profile}(\mathbf{p_u}, \mathbf{p_v}) \neq sim_{profile}(\mathbf{p_u}, \mathbf{p_v})$. For instance an experienced actor v might have at least the same competencies as a novice u. Therefore, v covers mostly all competencies of u and $\tau_{mir}(u, v)$ is high, while this is not true for $\tau_{mir}(v, u)$.

$$\tau_{mir}(u, v) = sim_{profile}(\mathbf{p_u}, \mathbf{p_v}) \tag{5.5}$$

Trust Teleportation. Trust τ_{tele} is applied as depicted by Figure 5.4b. It is assumed that u has established a trust relationship to w in the past, for example, based on w's capabilities to assist u in work activities. Therefore, others having interests and capabilities similar to w may become similarly trusted by u in the future. In contrast to mirroring, trust teleportation may also be applied in environments comprising actors with different roles. For example, a manager might trust a software developer belonging to a certain group. Other members in the same group may benefit from the existing trust relationship by being recommended as trustworthy as well. The idea is to attempt to predict the amount of future trust from u to v by comparing w's and v's profiles P.

$$\tau_{tele}(u, v) = \frac{\sum_{w \in M'} \tau(u, w) \cdot (sim_{profile}(\mathbf{p_w}, \mathbf{p_v}))^2}{\sum_{w \in M'} sim_{profile}(\mathbf{p_w}, \mathbf{p_v})} \tag{5.6}$$

Equation 5.6 deals with a generalized case where several trust relations from u to members of a group M' are teleported to a still untrusted actor v. Teleported relations are weighted and attenuated by the similarity measurement results of actor profiles. The idea of exploiting the interest similarity for trust, not only for adaptations, but for social formation in evolving communities was continued in [99] and the following chapters.

5.6 Behavior Monitoring and Self-Adaptation

The design of the architecture presented in the Section 5.4 provides a variety of possibilities for self-adaptation strategies. Figure 5.3 shows that the adaptation framework is loosely coupled to the testbed. Furthermore, logging interactions is a very generic approach to monitor the environment. However, the focus remains on adaptation of service misbehavior. Misbehavior appears on any unexpected change of behavior of a testbed component with noticeable function degradation impacts to the whole or major parts of the testbed. The monitoring and adaptation strategies follow the principle of smooth integration with least interference. However, a loosely coupled design often results in delayed and unclear state information. This can cause a possibly delayed deployment and application of adaptations. On the other hand, the testbed remains more authentic and true to current real environments which lack direct monitoring and adaptation functionality.

Monitoring in this architecture relies on the accuracy and timeliness of the *Logging Service*. *Diagnosis and Analysis* get all required status updates with the help of the *Event Subscriber* mechanism. Filtered status information populates the network model held by *Diagnosis and Analysis* module. During start-up the first interaction information is used to build the initial structure of the model. During runtime this information synchronizes the model with actual status changes observed on the network. Especially the interaction data filtered by the *Behavior Monitor* module allows *Diagnosis and Analysis* drawing conclusions from interactions about possible misbehavior at the services.

Similar to the regulation of behavior algorithm in Section 3.5 in the previous chapter, the recovery approach is to reconfigure the network by adapting the interaction channels between the service nodes. Channels are opened to provide new interactions to alternative nodes and closed to hinder misbehaving nodes to further affect the surrounding nodes and degrade the environment's function. The challenge is not only to detect misbehaving nodes but also to find alternative interaction channels for those problem nodes. A feasible adaptation must temporarily decouple misbehaving nodes from the network and instantly find possible candidates for substitution. Potential candidates must expose similar properties as the misbehaving node, e.g., have similar capabilities, and additionally, have the least tendency to misbehavior, e.g., those with least current task load. In a real Mixed System environment nodes' capabilities will change and the initial registered profiles will diverge with time from the current. Therefore, the *VieCure* framework includes a *Similarity Service* that keeps track of the profile changes and provides alternatives to nodes according to their current snapshot profile.

The following shows how the misbehavior patterns, delegation sink and factory, can be detected and adapted with the tools of the adaptation framework. The reader is reminded and in the line with the previous chapter, a **sink behavior** is observed when a node persists in accepting tasks from other nodes however prefers to work on tasks of certain neighbors, or under-performs in task processing. This behavior is recognizable by a dense delegation of tasks to the sink possibly requiring different capabilities and a low task completion notification in the observed time span. In the notion of Groovy scripts introduced in Section 5.2, Listing 5.3 shows the procedure used to detect and adapt nodes with sink behavior in the testbed framework.

```
// in the monitoring loop
def sinkNode = env.triggerSink(4) // sink trigger with threshold 4 tasks
if (sinkNode) { // sink suspected
 if (env.analyzeTaskQueueBehavior(sinkNode)) { // analyze task history
  def simNodes = sim.getSimilar(sinkNode) // call similarity service
  altNodes = []
  simNodes.each { s ->
   if (env.loadTolerable(s))
   altNodes += s // find nodes with tolerable load
  }
  def neighborNodes = env.getNeighbors(sinkNode) // affected neighbors
  neighborNodes.each { n ->
   n.dStrat = { refs -> // overwrite dStrat from Listing 1.
    refs += altNodes // add alternatives channels
    refs -= sinkNode // remove channel to sink
    ... // selection strategy
   }
  }
 }
}
```

Listing 5.3: Code example for sink adaptation.

The script extract defines the task queue trigger's `triggerSink` threshold first. If the limit of four tasks is violated by a node analysis `analyzeTaskQueueBehavior` scans the affiliated task history and compares the latest delegation and task status reporting patterns of the node. If a sink is detected, the *Similarity Service* `sim` is called and returns a set `simNodes` of possible candidates for replacement. In the next loop the candidates' current task queue size is examined (`loadTolerable`). Only those with few tasks are added to the final alternative nodes `altNode` list. In the last step the delegation strategies of the neighbors of the sink node are updated. The alternatives are added to the possible delegation candidates and the `sinkNode` is avoided.

A moderate use of queue capacity in contrast to high and exceeding delegation rates despite available alternatives causes overload at single nodes. This identifies the **factory behavior**. Again interaction data uncovers the misbehavior expressed by a high fluctuation of tasks from the factory and a low task completion rate in the monitored interval. The Groovy script in Listing 5.4 presents the factory adaptation algorithm for the testbed framework.

The factory trigger's threshold `triggerFactory` fires diagnosis on task queue sizes below two tasks. If `analyzeDelegationBehavior` confirms a pattern with high delegation frequency a factory node is detected. The same as with a sink, a selection of alternative nodes for a factory node replacement is collected. From this list only those with minor load are further considered. Then the affected neighbors who are delegating nodes (`getDelegators`) are freed from the factory and provided with the alternative nodes. Finally, the delegation strategy of the delegating neighbors is adapted. In contrast to the sink in the last step all the factory's

```
// in the monitoring loop
def factoryNode = env.triggerFactory(2) // factory trigger with threshold 2 tasks
if (factoryNode) { //factory suspected
 if (env.analyzeDelegationBehavior(factoryNode)) {// analyze task history
   def simNodes = sim.getSimilar(factoryNode) // call similarity service
   altNodes = []
   simNodes.each { s −>
     if (env.loadTolerable(s))
       altNodes += s // find nodes with tolerable load
   }
   def neighborNodes = env.getDelegator(factoryNode) // affected delegators
   neighborNodes.each { n −>
     n.dStrat = { refs −> // overwrite dStrat from Listing 1.
       refs += altNodes //add alternatives channels
       refs −= factoryNode // remove channel to factory
       ... // selection strategy
     }
   }
   factoryNode.dStrat={ } // no delegations allowed
 }
}
```

Listing 5.4: Code example for factory adaptation.

delegation channels are closed temporarily.

5.7 Simulation and Evaluation

The experiments in this chapter evaluate the efficiency of similarity based adaptation in a virtual team of a crowd of task-based services. This team comprises a few hundreds of collaborators. The assumption is that some of the HPSs expose misbehavior with the progress of time. Misbehavior is caused by team members that for various reasons including, e.g., task assignment overload, change of interest, or preference for particular tasks, start to process assigned tasks irregularly. The observation strategy is to detect misbehavior by analyzing the task processing performance of the team. A degrading task processing rate indicates misbehavior. The main idea is to detect degradation, identify the misbehaving team members with a task history analysis, and, in time, provide a fitting replacement for the misbehaving member. This member match is provided by the *Similarity Service* that mines the capabilities and noted changes at the members. The main information source of the misbehavior analysis and detection is the data contained in the delegated tasks.

Scenario Overview

Following the concept of OESs a scenario is modeled showcasing the interaction dynamics of a specific sector comprised by a bunch of teams. Interested parties wish to outsource multiple tasks to a crowd. In order to get their tasks completed they refer to an *entry point* service that forwards tasks to multiple teams of the crowd. A team comprises two types of members. The first, the delegators, receive new tasks directly from the entry point. Instead of working on the tasks their concern is to redistribute the tasks to their neighbors. These neighbors are also called workers. A delegator picks its most capable and trusted workers that can process the assigned task. Each team is specialized on a particular type of task. Tasks carry keyword information in order to distinguish which team receives a particular task.

A task's life-cycle starts at the entry point that provides the team constantly with new tasks. The entry point acts as a proxy between team and the actual task owner and its main assignment is to decide which of the team members is suitable for processing. The question is how to find the appropriate worker for a task. All services are registered at start up by the registry including their capabilities. Though, the information of the registry remains static and becomes outdated over the course of time. Members' processing behaviors can change over time when tasks start to be delegated and processing loads vary. Thus, the entry point can refer to the environment's registry for candidates at the beginning and shortly after bootstrapping but once profiles start to change the lookup information becomes inaccurate. The solution is the *Similarity Service* which is aware of these changes. It tracks the interest shift by monitoring the delegation behavior between interacting neighbors. Therefore, the service provides the most accurate candidates for a delegation during runtime. However, at the contrary the *Similarity Service* cannot provide satisfying results from the beginning because of the lack of interaction data.

Once the appropriate candidate is selected by the entry point it delegates the task. Teams, as in the scenario are composed of a sub-community of HPSs that know and trust each other and, hence, keep references to each other in a neighbor-list. Delegations in the team are only issued between these trusted neighbors. Tasks are associated with a deadline to define the task's latest demanded completion time and a processing effort. Each worker has its individual task processing speed depending on the knowledge compared to the tasks requirements and the current work load. At the end of a task's life-cycle, a worker reports the task as complete, or if the deadline is missed, expired. The main focus of the misbehavior regulation is to avoid tasks to expire. The algorithm identifies failing services by observing the task throughput. It filters tasks that missed their deadline in a certain period. Such a misbehavior is then adapted with the help of the knowledge of the *Similarity Service* and the task history. First the most similar members to the misbehaving are selected and then with a task queue size analysis the least loaded chosen for an adaptation. Depending on the current trust-based adaptation strategy channels between working nodes are added or delegations shifted to competent but less busy workers.

Experiment Setup

In order to simulate described medium size teams of the aforementioned OES model, the following are the environment settings. The teams comprise a total of 200 collaborators represented by WSs created by G2 scripts deployed to one backend instance. 20% of these members expose a delegation behavior the rest works on assigned tasks. All services are equipped with a task queue. As in the real world the services are not synchronized and have their individual working slots. Usually a worker processes one entire task per slot. A worker starts to misbehave once its task queue is filled past the threshold of 6 tasks. It then reduces its working speed to one third. A total of 600 tasks are assigned to the environment. Adaptation is not enabled from start. At start there is a period of 200 tasks with no adaptation. Then in an adaptation cycle the workers task queue size is monitored by tracing the delegation flow among the nodes. The difference between acknowledged assignments and complete or expired reported tasks results in the current task queue size at a particular worker. Once this number exceeds the preset task queue threshold which varies for the different results of the experiments, the similarity service is invoked for a list of workers with similar capabilities. In a loop over this list sorted by best match the candidate is picked with the currently smallest task queue size. The applied adaptation action depends on the experiment's current adaptation strategy. In trust mirroring a channel between two similar workers is opened which allows the overloaded node additionally to delegate one task per slot over the new channel. In trust teleportation the overloaded worker is relieved from the most delegating neighbor and a new channel is opened from the delegator to a substitute worker.

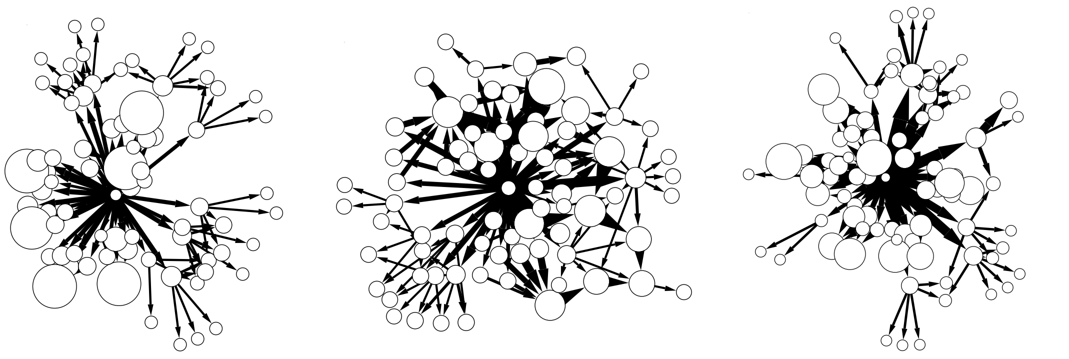

(a) No adaptation applied. (b) Adaptation through mirroring. (c) Adaptation through teleportation.

Figure 5.5: Evolving interaction networks based on adaptation actions.

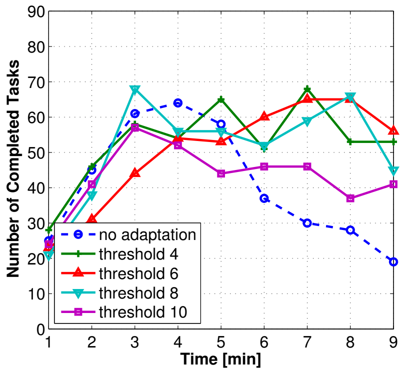
Figure 5.5 shows the temporal evolution of dynamic interactions under different adaptation actions. It demonstrates the changes in interactions for a threshold of 6 tasks in the three subfigures. A node's size represents the total number of incoming delegations. Larger edges indicate a high number of delegations across this channel with the arrow pointing in the delegation direction. Therefore, the node in the middle is easily identified as the entry point. It sheer provides tasks to all the connected delegators. Figure 5.5a shows that these delegators prefer selected workers to complete their tasks. In this figure six extremely overloaded workers are present after the first 200 tasks have left the entry point. Only a few others are sporadically called. Figure 5.5b represents the effects at the end of the experiment for the mirroring strategy. The effects of this strategy are clearly visible. The load between the workers is better distributed. A few, however more balanced worker nodes remain compared to no action because the delegators still prefer to assign tasks to their most trusted workers. However, a lager number of new workers is added at the outer leaves of the tree which release these nodes from their task load. Figure 5.5c highlights the situation with the trust teleportation strategy. The side-effects here show that the number of loaded nodes remains almost the same. However, the load peek at the preferred workers is kept below the predefined threshold. Once exceeded the worker is relieved from its delegator and a replacement found. With this strategy workers get loaded to their boundary and are then replaced with new workers.

The experiments test the effectiveness of adaptations with different task queue threshold triggers. The effectiveness is measured by the total task processing performance at the end of the experiment. Only completely processed and reported tasks went into the final result.

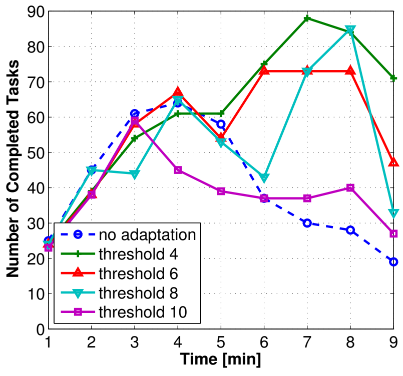
Result Description

Figure 5.6 presents the results of the simulation evaluations. Both diagrams provide the time-line in minutes on the x-axis and the number of completed tasks at the end of this period on the y-axis. In both cases there is a well noticeable incrementation of completed tasks until minute 4. This is when the first 200 tasks have been distributed to the workers. The task distribution is not linear over the measured period. This is due to the fact that at the beginning not so many tasks can be distributed because of bootstrapping delays in the G2 backend. This is also when the first adaptations are deployed. Whilst the task completion ratio decreases rapidly at this point if no adaptation actions are taken (demonstrated by the dashed line) the other lines represent the progress of the task completion when different thresholds triggers together with reconfigurations are applied. The diagrams in Figure 5.7 show again the time-line on the x-axis and the number of applied actions at the end of the period on the y-axis.

Figure 5.6a details the results of an adaptation strategy using trust mirroring. Generally all strategies perform better than when no action is taken. With a trigger threshold of 4 tasks and approximately 3 actions every minute the curve exposes an increment followed by a decrement between 70 and 50 completed task every minute. The pattern is similar to the curve representing a threshold of 8. Figure 5.7a shows that the adaptations are less and the altering of direction in Figure 5.6a is slower. The smoothest adaptations result from a trigger matching the real worker's threshold of 6 tasks. Comparing the figures, a smaller growth of success in task completion is noticed after the deployment of the 3 followed by 4 adaptations between minute 4 and 6. A threshold of 10 tasks decreases slower than an adaptation free environment but with only about 20 more successfully processed tasks. With the same adaptation effort as at threshold 8 this strategy exposes an overall inconvenient timing of the adaptations and can be considered impractical. The situation is different in Figure 5.6b. As Figure 5.7b shows, there are more

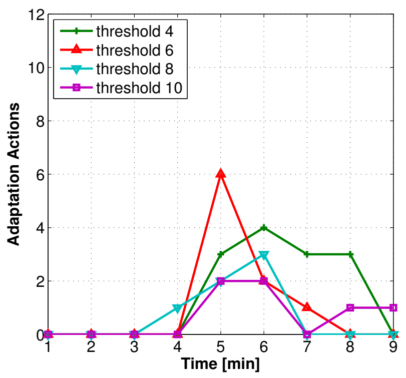


(a) Mirroring.

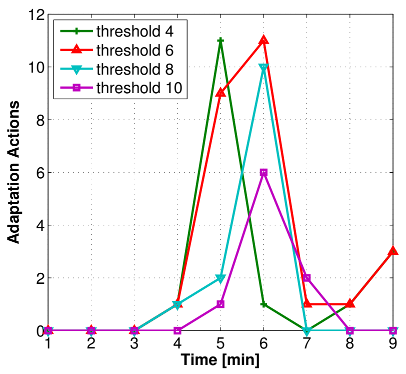


(b) Teleportation.

Figure 5.6: Adaptations using different thresholds for mirroring and teleportation.



(a) Actions applied in mirroring.



(b) Actions applied in teleportation.

Figure 5.7: Number of adaptation actions applied using different strategies.

adaptations deployed with this strategy. But not without leaving following side-effects. The curve of adaptations triggered at threshold 4 increases rapidly after minute 5 when a total of 11 new channels are provided to new workers in a time slot of 1 minute. Even if again with the smoothest progress among the successful strategies the curve representing actions at threshold 6 cannot reach the top performances of their neighbors (threshold 4 and 8). Instead the 20 new channels set between minute 4 and 6 let the system performance progress even. Finally the curve of threshold 10 has a noticeable regress between minute 3 and 4 caused by the dynamics of the system. In the following this type of strategy with only 9 adaptations in total is not able to recover and is even outperformed by the no adaptation run. The final results show that the precise timing of multiple adaptations in a short term is most convenient for environment adaptation actions. However this has a trend to highly altering task processing results (e.g., approximately 40 task for a threshold 8 in Figure 5.6b). Comparing both, a strategy where the trigger matches the environments actor's threshold of 6 is most practical in a balanced environment. Strategies with a threshold above 8 are infeasible for this setup. Generally the teleportation strategy performs better than mirroring, however requires twice as much and more adaptation actions.

5.8 Conclusion

Basing on the results of the previous chapter, the main objective of this chapter was to demonstrate the successful integration of two frameworks. On one side, the G2 SOA testbed provides the tools to design and deploy a Mixed System according to the studied open environments, and on the other, the extensible *VieCure* Framework for adaptation supports an integration of self-adaptive methodologies adopted for the observed environment. The two remain separate and independent frameworks and are only loosely coupled. As an adaptation loop extension a external service providing similarity ratings for the testbed's services was added. The results of the evaluation confirm that the deployed task processing team scenario and the monitoring and adaptation strategies supported by profile similarity interplay satisfactorily. A precise timing and a carefully aligned threshold for the actions is essential to reach high amounts of task completion rates. Further studies of integration of adaptation strategies for Mixed Systems into G2 can be found in [99].

For this thesis, this chapter concludes the studies on human misbehavior adaptation techniques in open environments basing on Mixed Systems. The gained observations base only on simulated human behavior. However, the final deployment of the behavior models to a real SOA testbed gives some insight into the dynamics and complexities to which such systems are exposed. Starting with the detachment of the similarity service as an external knowledge resource to the adaptation framework, future work could include studies on a distributed architecture of the adaptation framework in real large-scale Mixed System.

Part III

Sustainability and Maintenance in Mixed Systems

CHAPTER 6

Broker Query and Discovery Language: Connecting and Finding Communities

6.1 Chapter Overview

While the preceding chapters consider online adaptations to the infrastructure, i.e. communication channel regulation, as a methodology of self-healing poor or degenerated delegation activities, the chapters in this part consider offline approaches. Instead of directly interfering with the system and deploying changes, in the following chapters the interaction information is utilized to discover distinct features and set-ups in the human crowds. In particular, the idea of adaptation is related to self-organization. The platform provider tries to organize the system with the help of collected information and arranges the system according to expertise areas. Furthermore, instead of keeping strong references to all the members, well connected intermediates, i.e. brokers, are maintained. Brokers are then, for example, used as gateways to communities with a particular knowledge or skills, and also, to reduce the burden of management and organization of the large-scale system. Whilst the next chapter outlines the advantages in management of the broker approach this chapter peruses the task of how to find suitable brokers in open environments. The core presents a novel discovery language for such brokers supported by metrics for connectivity measurements.

Chapter Outline. Section 6.2 explains how to delegate some of the responsibilities in Mixed Systems to intermediates and how to apply social network principles to bridge segregated collaborative networks. Section 6.3 exemplifies discovery of brokers and broker behavior patterns with the scenario illustrated. Section 6.4 describes the model social trust used to identify communities and broker. The novel Broker Query and Discovery Language (BQDL) is defined in Section 6.5 followed by a discussion on the implementation and evaluation in Section 6.6.

6.2 Bridging Socially-Enhanced Virtual Communities

The studies in this chapter once again refer to the OESs discussed already in the preceding chapters. In such ecosystems, flexible interactions commonly take place in different organizational units. The challenge is that top-down composition models are difficult to apply in constantly changing and evolving service-oriented collaboration system. There are two major obstacles hampering the establishment of seamless communications and collaborations across organizational boundaries:

1. the *dynamic discovery* and *composition* of resources and services,
2. and *flexible* and *context-aware interactions* between people residing in different departments and companies.

Theories found in social network analysis are promising candidate techniques to assist in the formation process and to support flexible and evolving interaction patterns in cross-organizational environments. The theory of strategic formation in social networks and communities [111] and, furthermore, the one of structural holes developed by Burt [15] is based on the hypothesis that individuals can benefit from serving as intermediaries between others who are not directly connected. A formal approach to strategic formation based on advanced game-theoretic broker incentive techniques was presented in [52]. The approach presented in this chapter is based on interaction *mining and metrics* to dynamically discover brokers suitable for connecting communities in service-oriented collaborations.

In social networks, relations and interactions typically emerge freely and independently without restricted paths and boundaries. Research in social sciences has shown that the resulting social network structures allow for relatively short paths of information propagation (the *small-world phenomenon*, e.g., see [51]). While this is true for autonomously forming social networks, the boundaries of collaborative networks are typically restricted due to organizational units and fragmented areas of expertise. The proposition in this chapter is to apply social network principles to bridge segregated collaborative networks. The theory of structural holes is based on the idea that individuals can benefit from serving as intermediaries between others who are not directly connected [15]. Thus, such intermediaries can potentially *broker* information and aggregate ideas arising in different parts of a network [52].

In order to bridge socially-enhanced virtual communities the following concepts are presented in this chapter:

- The concept of *Brokers* is introduced to establish connections between independent subgroups in OESs. The presented approach enables the dynamic selection of brokers based on changing interest profiles.
- Selected metrics and their application to support the discovery and selection of brokers including *social trust* are presented.
- A novel query and discovery language is introduced. The *Broker Query and Discovery Language (BQDL)* can by applied to discover suitable brokers based on query preferences (discovery policies). The novelty of BQDL is the ability to query social network data

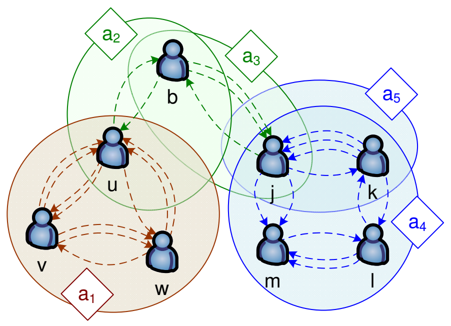
considering information obtained from mining results to fulfill the requirements for broker discovery in OESs.

6.3 Emerging Virtual Communities

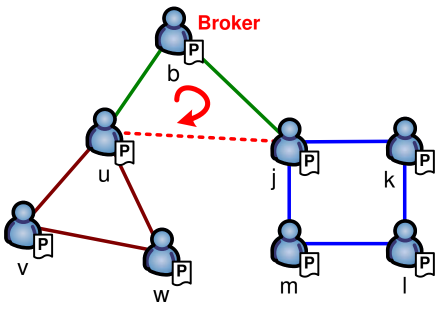
To better explain the idea let us discuss an actual collaboration scenario as depicted in Figure 6.1. Various member groups collaborate in the context of five different activities a_1, a_2, a_3, a_4 and a_5 (see Figure 6.1a). These groups intersect since members may participate in different activities at the same time. The color of the activity context determines the expertise areas an activity is related to. Such activities are, for instance, the creation of new specifications or the discussion of future technology standards. Note, that activities in flexible collaboration environments include the goal of ongoing tasks, involved actors, and utilized resources such as documents or services. They are either assigned from the outside of a community, e.g., belonging to a higher-level process, or emerge by identifying collaboration opportunities.

The members use HPS technologies to interact in the context of ongoing activities. Interactions are logged for analysis. Relations emerge from interactions as illustrated in Figure 6.1b, and are bound to particular scopes (*expertise areas*). The context in which interactions take place bases on tags applied to various artifacts exchanged between the partners. Tags are used to combine similar activities to create scopes (i.e., activity boundaries). In the scenario, a scope comprises relations between OES members regarding help and support activities in different expertise areas (reflected by tags of exchanged messages). Scopes are used for different purposes.

First, by analyzing the interaction context (i.e., using message tags), the users' centers of interest are determined. Frequently used keywords are stored in the actors' profiles (see symbol *P*) and later used to determine their interests and expertise areas. Second, interactions that occurred in a pre-defined scope are aggregated, metrics (numerical values describing prior interaction behavior) are calculated, and, thus, allow interpreting them as *social trust* that is based on reliability, dependability and success.



(a) Member interactions.



(b) Emerging relations.

Figure 6.1: Collaboration model for OESs: (a) interactions between OESs members are performed in the context of activities; (b) social relations and profile areas emerge based on interactions.

Brokering and Compositions

Consider a scenario in the given Scenario in Figure 6.1b. Suppose u wants to set up an activity that requires at least one additional expert from the *brown* $\{u, v, w\}$ and *blue* domain $\{j, k, l, m\}$. Since u personally knows v and w from previous collaborations, which is reflected by FOAF [14] `knows` relations, u is well-connected to the *brown* expertise area. However, u does not know any member from the *blue* domain. The broker concept helps to solve this problem. Actor u collaborated with b in the *green* domain, who is connected to j. Therefore, b could potentially act as a broker and forward requests or invitations to join u's current activity to j. It is of high importance to establish personal contacts in socially-oriented environments compared to the traditional SOA domain, where services are mostly composed based on their properties (i.e., features and QoS) only.

Assuming one is able to infer meaningful social relations between network members, such relations have major impact on future collaborations in different scenarios:

1. *Supporting the Formation of Expert Groups.* Successfully performed compositions of actors should not be dissolved but actively facilitated for future collaborations. Thus, tight trust relations can be dynamically converted to FOAF relations (i.e., *discovery of relevant social networks*).

2. *Controlling Interactions and Delegations.* Discovery and interactions between members can be based on FOAF relations. People tend to favor requests from well-known members compared to unknown parties.

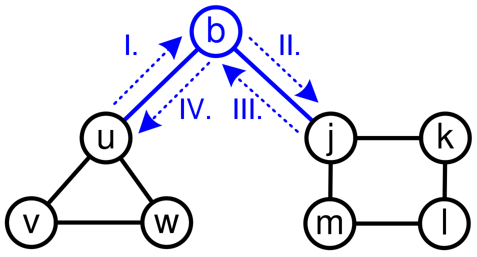
3. *Establishment of new Social Relations.* The emergence of new personal relations is actively facilitated through brokers. The introduction of new partners through brokers (e.g., b introduces u and j to each other) leads to future trustworthy compositions.

Broker Behavior Patterns

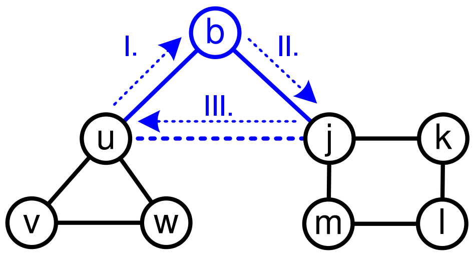
Brokers differ from other actors by their mediation capabilities. A broker acts as an intermediary node between two previously separated communities or collaboration teams. Thus, it is essential that it monitors frequently demanded contacts, updates and maintains its relations to increase and strengthen its popularity, and consequently, trust. If demand decreases, the broker must find and establish new relations. The discussed way to solve the problem is to provide the possibility of querying the social network for new contacts of interest. Of interest are, e.g., contacts to communities with high trust relations among the members and a distinct expertise.

In this chapter, different types of brokers are defined. Considering HPS-based interactions such as delegations of online help and support requests, brokers may exhibit different behavior patterns as illustrated by Figure 6.2:

(a) *Persistent Exogenous Interaction Pattern.* Any request and response is forwarded by the broker, thereby shielding the actually interacting nodes from each other. Thus, each network segment remains separated for the entire duration of collaboration.



(a) Persistent pattern.



(b) Triadic pattern.

Figure 6.2: Exogenous broker behavior patterns.

(b) *Triadic Exogenous Interaction Pattern.* The broker encourages receivers of requests to establish direct connections to the initiator, and therefore, actively facilitates the emergence of new social relations.

The observation is that both types of interaction patterns are applied in today's social and collaborative environments. A broker may favor one pattern over the other due to various reasons. For example, controlling the flow of interactions between personally unknown actors can strengthen a broker's reputation [52]. Establishing direct relations can significantly reduce a broker's workload. Another possible explanation for varying broker behavior patterns may be the similarity of expertise profiles.

For example, if a broker connects similar actors, it may apply the triadic pattern to support the establishment of new social relations. However, if actor profiles diverge significantly, the broker may need to mediate interactions persistently; for example, due to the lack of a common vocabulary or understanding between communities. The proposed query language (BQDL) supports both cases. However, the discussions in the following sections mainly demonstrate the application of BQDL for persistent exogenous broker behavior patterns without detailing the peculiarities of advanced triadic patterns.

6.4 Emergence and Evolution of Trust

In contrast to a widely used security perspective on trust, as in the previous chapter, here *social trust* is defined relying on the interpretation of previous collaboration behavior and also considering the similarity of dynamically changing interests [33, 96]. Especially in collaborative environments where users are exposed to higher risks as compared to common social network scenarios [28] and business is at stake, considering social trust is essential to effectively guide interactions [64]. In detail, the idea of trust in this chapter follows [35, 69, 7, 96]:

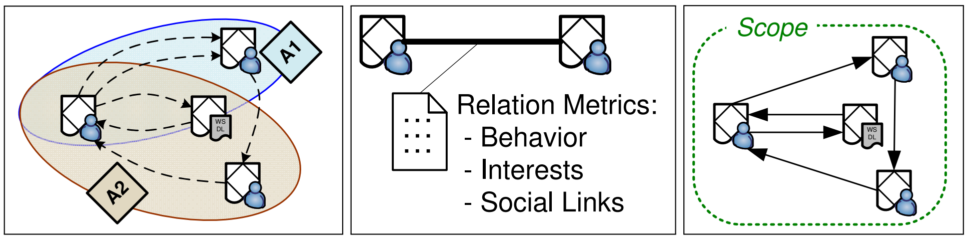
Trust reflects the expectation one actor has about another's future behavior to perform given activities dependably, securely, and reliably based on experiences collected from previous interactions.

The fundamental approach to automatic interaction-based trust inference is depicted in Figure 6.3. The necessary steps are the following:

1. *Interactions and Monitoring.* As mentioned in the scenario, people interact to perform their tasks. Work is modeled as activities, which describe the type and goal of work, temporal constraints, and used resources. As interactions take place in the context of activities (Figure 6.3a), they can be categorized and weighted. SOAP is the standard message format to support interactions between distributed software services. As already outlined in the previous chapter also human interactions can be supported in a service-oriented manner using technologies such as SOAP. This technology including extensions such addressing and correlation mechanisms is state-of-the-art in service-oriented environments and well supported by a wide variety of software frameworks (c.f., G2).

2. *Link Metrics.* Interaction logs are used to infer metrics that describe the relation of single actors (Figure 6.3b). Various metrics can be calculated by analyzing interaction logs such as behavior in terms of availability and reciprocity. A simple example of a metric is the *success rate* of delegated tasks between two members (successfully processed tasks divided by the total number of delegated tasks). Relation metrics describe the links between actors by accounting for (i) recent interaction behavior, (ii) profile similarities (e.g., interest or skill similarities), (iii) social and/or hierarchical structures (e.g., role models). However, the argument here is that social trust relations largely depend on personal interactions. A community of actors is modeled with their social relations as a directed graph, where the nodes denote network members, and edges reflect (social) relations between them. Since interaction behavior is usually not symmetric, actor relations are represented by *directed links*.

3. *Scoped Trust.* The approach considers the diversity of trust by enabling the flexible aggregation of various interaction metrics (e.g., *success rate* and *responsiveness*) that are determined by observing ongoing collaborations. Finally, available relation metrics are weighted, interpreted, and composed by a rule engine (the detailed mechanisms can be found in [96]). The result (i.e., a linguistic representation such as *high*, *medium*, or *low*) describes trust between the actors with respect to scopes (Figure 6.3c). For instance, trust



(a) Interactions. (b) Link metrics. (c) Scoped trust.

Figure 6.3: Trust emerging from interactions: (a) interaction patterns shape the behavior of actors in context of activities; (b) (semi-)automatic rewarding of behavior and calculation of interaction metrics; (c) inference in scopes by interpretation of metrics.

relations in a scope 'scientific dissemination' could be interpreted from interaction behavior of actors in a set of paper writing activities.

6.5 BQDL Specifications

Technically, the focus of BQDL is to provide an intuitive mechanism for *querying data from social networks*. These networks are established upon *mining and metrics*. Thereby, properties of such networks are under constant flux and changes. BQDL is not a generic graph query language such as SPARQL [115], which has been designed to query ontological data. Instead, BQDL addresses the specific requirements for the discovery of actors such as brokers by accounting for (weighted) paths and metrics obtained from mining results. There is also a query language for social networks in [81]. However, the language in [81] is closest to the BQDL approach (e.g., path functions). It lacks the support of the discovery of complex sub communities based on metrics and interaction mining techniques.

The key elements of BQDL are defined next. Table 6.1 lists important language elements to query interaction graphs. The language is inspired by an SQL-like syntax. It is important to note that BQDL operates on a graph defined as $G = (N, E)$ composed of a set of nodes N and edges E.

Element	Description
`satisfy`	Requires that a given condition is fulfilled by a set of nodes or edges.
`as`	Creates an alias for groupings of nodes, edges, or paths.
`<all>`	Retains all nodes/edges/subgraphs satisfying a given condition.
`[ ]`	An expression to satisfy conditions for exactly one `[1]`, one to m `[1..m]`, or one to many `[1..*]` nodes or edges.

Table 6.1: Important BQDL language elements.

A `Select` statement retrieves nodes and edges in G as well as aggregates of graph properties (for example, properties of a set of nodes). While traditional relational databases operate on tables, BQDL uses the `From` clause to perform queries on a graph G. A `Where` clause specifies filters and policies upon nodes, edges, and paths. To give intuitive examples, a set of BQDL queries along with their meaning considering a graph G and a set of subgraphs $G' \subseteq G$ is presented next. The structure discussions related to a BQDL query have four essential steps: **R** the basic requirements/goal of a query, **A** the approach that is taken, **O** the output of the query, **D** the detailed description of the query.

Connecting Predefined Communities

As a first simple example, in Figure 6.4, consider two initially disconnected communities residing in the graph G as a sets of nodes depicted as variables `var source` = $\{n_1, n_2, \ldots, n_i\}$ and `var target` = $\{n_j, n_{j+1}, \ldots, n_{j+m}\}$. **R1:** The goal is to find a broker connecting disjoint sets of nodes (i.e., not having any direct links between each other). **A1:** Two subgraphs `G1` and `G2` are created to determine brokers which connect the source community $\{u, v, w\}$ with

```
1   Input: Graph G, var source = $\{n_1, n_2, \ldots, n_i\}$,
2          var target = $\{n_j, n_{j+1}, \ldots, n_{j+m}\}$
3   Output: List of brokers
4
5   Select node From (
6                   ( Select distinct(node) From G
7                           Where
8                                   /* At least one in source 'knows' node */
9                             ( [1..*] n in source ) satisfy
10                        Path (n to node) as P1 With P1.length = 1 )
11                            as G1,
12          ( target ) as G2
13    )
14    Where
15          /* Retain all nodes that satisfy path filter */
16          ( <all> n in G1.nodes ) satisfy
17                  /* Path to any in G2.nodes */
18                  Path (n to [1..*] G2.nodes) as P2
19                        With P2.length = 1
20          and
21          /* Retain all edges that satisfy edge filter */
22          ( <all> e in G1.edges ) satisfy
23                  (e.relation = EPredicates.BIDIRECTIONAL) and
24                  (e.trust >= MTrust.MEDIUM)
25
26  Order by node
```

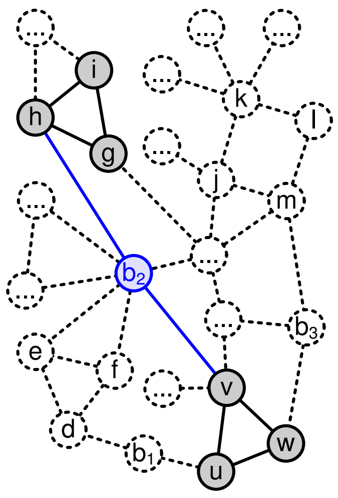


Figure 6.4: BQDL example 1: find broker to connect two predefined communities.

the target community $\{g, h, i\}$ (i.e., see `From` construct). **O1:** The output of the query is (the example shown in Figure 6.4) a list of brokers connecting $\{u, v, w\}$ and $\{g, h, i\}$. The lines 1-3 specify the input/output parameters of the query. **D1:** As a first step, a (sub)select is performed using the statement as shown by the lines 6-11. The statement `distinct(node)` means that a set of unique brokers shall be selected based on the condition denoted as the `Where` clause with a filter (lines 9-10). The term '`[1..*] n in source`', where `source` is the set of nodes passed to the query as input argument, means that at least one node $n \in G$ must satisfy the subsequent condition. Here the condition is that the node n has a link (i.e., through `knows` relations) to the source set of nodes. This is accomplished by using the `Path` function that checks whether a link between two nodes exists (the argument '`(n to node)'`). The path alias is used to specify additional constraints such as the maximum path length between nodes (here '`P1 With P1.length = 1`'). The second step is to create an alias `G2` for the target community $\{g, h, i\}$. By using the aliases `G1` (line 11) and `G2` (line 12) further filtering can be performed using the `Where` clause in line 14. The same syntax is used as previously in the sub-select statement (lines 9-10). The construct `<all>` retains nodes '`n in G1.nodes`' (`G1` holding the set of candidate brokers) that are connected to at least one node in the target community `G2` with direct links ('`P2 with P2.length = 1`'). Further filtering is performed by defining lines 22-24.

Here, brokers in `G1` and both the source $\{u, v, w\}$ the target community $\{g, h, i\}$ must have edges between each other that are bidirectional. In this graph representation, this means that each

```
 1  Input: Graph G, var search = $\{t_1, t_2, \ldots, t_n\}$
 2  Output: List of communities
 3
 4  Select load, nodes from (
 5                      ( Select distinct(nodes) as G' from G
 6                              Where
 7                                          ( <all> n in G'.nodes ) satisfy
 8                          Path (n to [1..∗] G'.nodes) as P1
 9                          With (
10                                  P1.length = 1 and P1.trust = MTrust.HIGH
11                  and ( [1..∗] tag in P1.tags ) satisfy
12                      (search contains tag)
13                          )
14              ) as SG1
15              Where
16                      ( <all> G'' in SG1 ) satisfy
17                              (G''.load <= GMLoad.MEDIUM)
18
19  Order by load asc
```

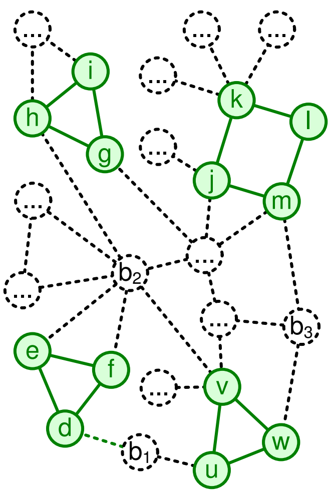


Figure 6.5: BQDL example 2: find ranked communities based on search criteria and metrics.

relation has to be interpreted as, for example, b_2 `knows` h and h `knows` b_2. A set of different metrics is established in the system. A specific type of metric (e.g., trust) is denoted by the namespace `MTrust`. In the specified query, each actor in the result set must share a minimum level of trust depicted as '`e.trust >= MTrust.MEDIUM`'. Trust metrics are associated to edges between actors. The term `MTrust.MEDIUM` is established based on mining data to obtain linguistic representations by mapping discrete values (metrics) into meaningful intervals of trust levels. The last statement '`Order by node`' in Figure 6.4 implies a ranking procedure of brokers. This can be accomplished by using eigenvector methods in social networks such as the PageRank algorithm to establish authority scores (the importance or social standing of a node in the network) or advanced game-theoretic techniques based on the concept of structural holes (see for example [52]). The detailed mechanisms of this procedure are not the focus of this work.

Finding Communities

The broker discovery example in the previous section (Figure 6.4) is straightforward. The target community is already specified and passed to the query as `var target` = $\{n_j, n_{j+1}, \ldots, n_{j+m}\}$. The next example query eliminates this assumption by showing an approach to find suitable communities based on search criteria (e.g., activity or skill tags). **R2:** The goal of the query as specified in Figure 6.5 is to find sub-communities (or subgraphs) in G that match search criteria. **A2:** Search is performed by using a set of distinct tags specified as input parameter `var search` = $\{t_1, t_2, \ldots, t_n\}$. **O2:** The output of the query is a list of communities. **D2:** The first step is to perform a (sub)select of distinct communities (see `distinct(nodes) as G'` in line 5) to obtain non-overlapping groups of community members specified by the lines 5-14. For example, Figure 6.5 shows four groups of nodes [$\{d, e, f\}$,$\{g, h, i\}$,$\{l, m, j, k\}$,$\{u, v, w\}$] each of them satisfying the constraints specified in the query. Each node in a specific community must be linked to at least one community member so that '`Path (n to [1..∗]`

```
1   Input: Graph G, var source = $\{n_1, n_2, \ldots, n_i\}$,
2                                var search = $\{t_1, t_2, \ldots, t_n\}$
3   Output: List of brokers and communities
4
5   Select node, nodes from (
6                   /* Select brokers */
7                   ( /* ... */ ) as G1,
8           /* Select communities */
9           ( /* ... */ ) as SG1
10    )
11    Where
12          ( <all> n in G1.nodes ) satisfy
13                  /* To one in SG1 */
14                  Path (n to [1] SG1) as P1 With P1.length = 1
15
16  Order by node
```

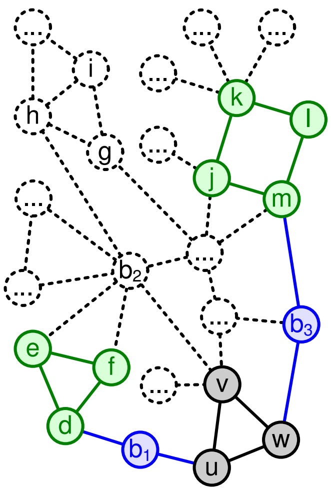
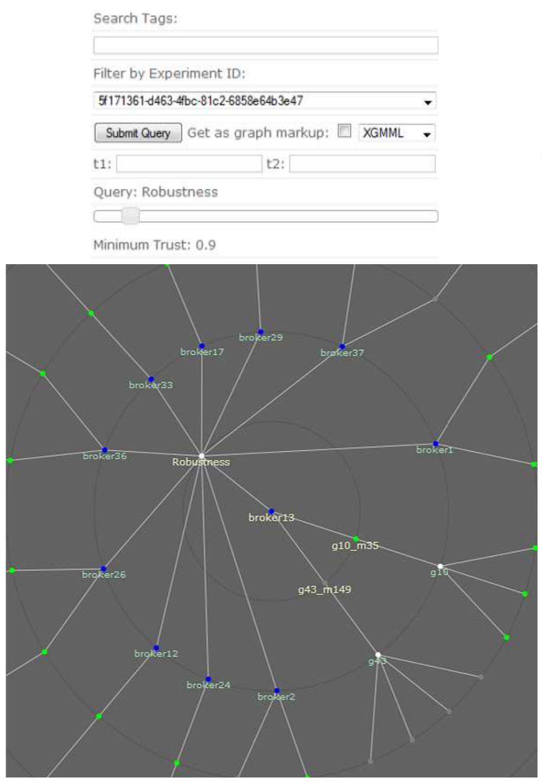


Figure 6.6: BQDL example 3: find exclusive brokers to connect two communities.

G′.nodes) as P1’. Also, at least one path between nodes with ‘length = 1′ satisfying trust requirements (trust level MTrust.HIGH) must exist in order to consider a node as a community member. Finally, a path must contain the tags specified by the search query (lines 11-12) to ensure that a member has interacted (collaborated) with other members in the context of certain activities. The alias SG1 provides access to each community. The Where clause applies filtering of communities based on load conditions measured by graph metrics (GMLoad). For example, load conditions G”.load are measured by the number of inbound requests and the number of pending tasks within the community.

Finding Exclusive Brokers

The final BQDL example is depicted by Figure 6.6 to combine previously introduced concepts for broker discovery. **R3:** The basic idea of this example is to find brokers that are connected to exactly one candidate (target) community. Again, the source community is $\{u, v, w\}$. **A3:** Communities are retrieved along with brokers. Filtering is applied based on paths to obtain exclusive brokers. **O3:** The output of the query are brokers along with communities they are connected to (e.g., b_1, $\{d, e, f\}$). **D3:** First, a set of candidate brokers is retrieved and made available via the alias G1 (line 7). This is the same procedure as introduced before (see Figure 6.4). Second, communities are retrieved and stored in SG1 (line 9). Again, this is based on the same principle as introduced previously in Figure 6.5. *Exclusive brokers* connect exactly one community. This is accomplished by the statements in 12-14 demanding for ‘n to [1] SG1’. The broker b_2 is a non-exclusive broker because it connects multiple communities $\{d, e, f\}$ and $\{g, h, i\}$, thereby making $\{g, h, i\}$ unreachable from the $\{u, v, w\}$ community perspective.


Search Tags:

Filter by Experiment ID: 5f171361-d463-4fbc-81c2-6858e64b3e47

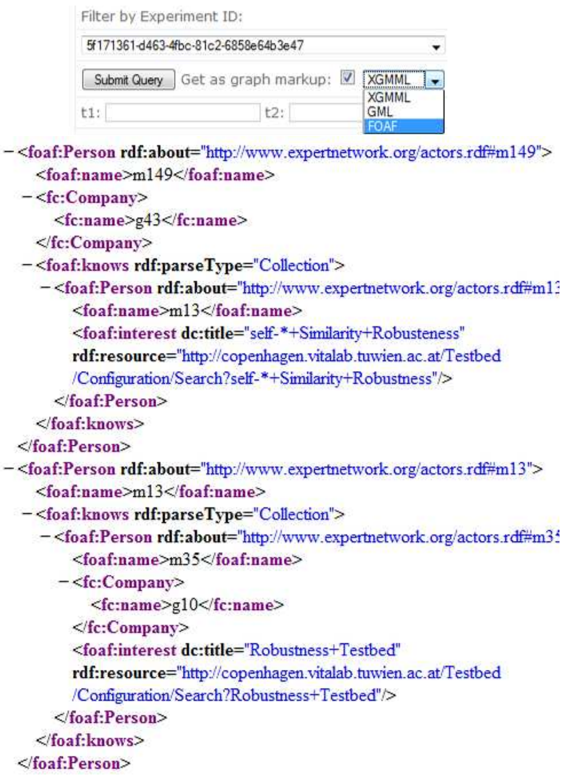
Submit Query Get as graph markup: ☐ XGMML

t1: t2:

Query: Robustness

Minimum Trust: 0.9

(a) Network visualization view.


Filter by Experiment ID: 5f171361-d463-4fbc-81c2-6858e64b3e47

Submit Query Get as graph markup: ☑ XGMML

t1: t2:

```
– <foaf:Person rdf:about="http://www.expertnetwork.org/actors.rdf#m149">
    <foaf:name>m149</foaf:name>
  – <fc:Company>
      <fc:name>g43</fc:name>
    </fc:Company>
  – <foaf:knows rdf:parseType="Collection">
    – <foaf:Person rdf:about="http://www.expertnetwork.org/actors.rdf#m1
        <foaf:name>m13</foaf:name>
        <foaf:interest dc:title="self-*+Similarity+Robusteness"
        rdf:resource="http://copenhagen.vitalab.tuwien.ac.at/Testbed
        /Configuration/Search?self-*+Similarity+Robustness"/>
      </foaf:Person>
    </foaf:knows>
  </foaf:Person>
– <foaf:Person rdf:about="http://www.expertnetwork.org/actors.rdf#m13">
    <foaf:name>m13</foaf:name>
  – <foaf:knows rdf:parseType="Collection">
    – <foaf:Person rdf:about="http://www.expertnetwork.org/actors.rdf#m3
        <foaf:name>m35</foaf:name>
      – <fc:Company>
          <fc:name>g10</fc:name>
        </fc:Company>
        <foaf:interest dc:title="Robustness+Testbed"
        rdf:resource="http://copenhagen.vitalab.tuwien.ac.at/Testbed
        /Configuration/Search?Robustness+Testbed"/>
      </foaf:Person>
    </foaf:knows>
  </foaf:Person>
```

(b) Example of FOAF profile.

Figure 6.7: Web-based broker discovery and network visualization tool.

6.6 Implementation and Discussion

The environment consists of the integration setup presented in Section 5.4 of Chapter 5. A WS-based simulation environment using the G2 [45] interfaces with the *VieCure* Framework middleware implementing user tools, logging, and eventing capabilities. The tools assist the users in discovering brokers based on visualized community structures.

Broker Discovery Application

The implemented prototype includes a Web-based broker discovery tool helping users in analyzing various BQDL queries and corresponding parameters. Figure 6.7 shows screenshots of the tool and an example FOAF profile that can be retrieved from the Web application. The users access information captured from the OES environment. The network view is obtained by mapping raw SOAP-interactions into a graph representation composed of nodes (services) and edges (interaction links). In the implementation, this is performed by selecting a particular set of logs which are associated with an `Experiment ID`. After issuing the corresponding (BQDL) query, a graph is visualized consisting of several brokers connecting communities. By default, the collaboration network is visualized as a graph view as depicted in Figure 6.7a. The user is able to select a trust threshold by moving a slider bar. A reduced (demanded) trust threshold results in more target communities being added to the visualization. *Color online*: target communities matching search criteria are depicted using a node that is labeled with the community identifier (white color) and a set of green colored nodes (labeled with the node's name) linked to the central community node (to indicate a node's membership to a community). Interactions can be retrieved as FOAF profiles (see Figure 6.7b) that include `<foaf:interest>` tags.

SOA Testbed Environment

The evaluations were gathered using the logging features of the Genesis2 framework [45]. Genesis2 has a management interface and a controllable runtime to deploy, simulate, and evaluate SOA designs and implementations. A collection of extensible elements for these environments are available such as models of services, clients, registries, and other SOA components. Each element can be set up individually with its own behavior, and steered during execution of a test case. For the experiments in this work, Genesis2 Backends were deployed to the *Amazon Elastic Compute Cloud*[1]. Depending on the amount of involved service instances, two or three *Community AMIs* of the type *High-Memory Extra Large Instance* (17.1GB of memory) running a Linux OS were launched. In the following, each instance provided the same Genesis2 Backend snapshot via mountable volumes from the *Elastic Block Store*. Finally, the deployed environment setup from a local Genesis2 Frontend was as follows. It included SOA-based OESs established by Genesis2 Web services equipped with simulated behavior and predefined relations to provide communication channels and instantiate communities. Services act like HPSs when delegating each other new tasks, processing tasks, re-delegating existing tasks, or reporting tasks' progress status. Tasks are not delegated arbitrarily but must match the receiver's capabilities. Therefore, they are tagged by three keywords one of which must match the picked receiver's capabilities. As an intermediate, a broker combines capabilities of the two communities it connects. The broker avoids task processing and only forwards tasks. The finally deployed environments are variable in number of services, number of participants per group (2-5 services) and consequently also in number of communities and required brokers that connect at least each community with another (see also [62] for *minimum spanning trees* in social networks). Task processing and delegation decisions happen individually and in random time intervals (1-8 seconds).

BQDL Performance Aspects

Several experiments were conducted to test the performance of the BQDL implementation under varying characteristics such as varying number of nodes and groups. The results are summarized in Figure 6.8. The simulated environments had different numbers of nodes and interactions to obtain insights in performance aspects. BQDL tools (Figure 6.7) and BQDL related graph libraries implemented in C# have been deployed on the local (lab-based) blade servers equipped with Intel Xeon 3.2GHz CPUs (quad core) and 10GB RAM hardware. Interaction logs are managed by MySQL 5.0 databases. A client request pool (RP, see Table 6.8a) is created on a separate machine (Intel Core2 Duo CPU 2.50 GHz, 4GB RAM) to perform parallel invocations

[1]Amazon EC2: `http://aws.amazon.com/ec2/`

Experiment	# Req.	MIN	AVG	MAX	Total
1 (RP=10)	50	3167	9083	10368	52543
	100	1669	9369	10576	101244
	200	1825	9211	10748	190647
1 (**RP=50**)	50	1606	15955	29952	50762
	100	1482	27440	48562	98685
	200	1638	36313	47689	188573
1 (RP=100)	50	1606	15955	29952	50762
	100	1544	28560	57501	105331
	200	1591	55185	100370	202394
2 (RP=50)	100	2308	37891	63258	123677
3 (RP=50)	100	2854	42041	67516	136266
4 (RP=50)	100	3276	55058	84739	167778

(a) BQDL processing time.

Applied Tags in Exp. 4 (n=1029 and groups=230)	Frequ.
self-*	295
Robustness	306
Testbed	311
DB	314
Healing	321
Trust	322
WS	327
Autonomic	335
Similarity	341
Logging	353

(b) Tag frequency.

Query ID	BQDL query keywords	# Brokers	AVG proc. time
Q1	Robustness Logging	105	3993
Q2	Robustness Logging DB Testbed	134	3666
Q3	Robustness Logging DB Testbed Similarity	146	3478

(c) BQDL queries in Exp. 4, number of discovered brokers and AVG processing time.

Figure 6.8: BQDL processing statistics in simulated environment (in milliseconds).

of the BQDL query Web service. Clients are connected with the server via a local 100MBit Ethernet.

The results of the first experiment are based on 198 nodes, 200 edges, and a total number of 10 distinct tags applied to interactions between nodes. The BQDL processing time for this environment is shown in Table 6.8a. The number of concurrent requests, denoted as RP, is varied by launching multiple threads. Given a size of **RP=50** and a total amount of # 100 requests to be processed, setting RP=100 does not speed up the processing time of requests (i.e., the total time needed to process a number of requests). The average processing time increases by comparing RP=100 and RP=50 due to the overhead when handling a larger amount of requests simultaneously. Thus, RP=50 was used for all further experiments. Also, by processing a larger amount of requests, say # 200, the total processing time linearly increases with the number of requests. The number of nodes and interactions was increased to understand the scalability of BQDL under different conditions: experiment 2 with 579 nodes, experiment 3 comprising 774 nodes, and experiment 4 with 1029 nodes in the testbed. HPSs in the testbed have been deployed equally on multiple hosts, e.g., 3 cloud hosts in experiment 4 to achieve scalability. In subsequent experiments detailed in Figure 6.8 (experiments 2-4) the focus is on a request pool with RP=50 and 100 requests to be processed by the BQDL service using different keywords (see Table 6.8c). To compare the experiments 1-4, the interaction graph was queried using the keywords `Robustness Logging`. Increasing the number of nodes by a factor ≈ 3 (see experiment 1 and 2), the processing time of BQDL raises by 30%. Comparing the experiments 2 and 3 (node addition of $\approx 30\%$), the processing time increases by 10%. By comparing the experiments 3 and 4 (node addition of $\approx 30\%$), the processing time increases by 20%. The experiments show that BQDL scales with larger testbed environments linearly. Furthermore, different BQDL query keywords as shown in Table 6.8c were used. The number of discovered brokers increases given multiple keywords (see Table 6.8b for the set of available tags). The average BQDL processing time is not significantly influenced by the number of used keywords.

6.7 Conclusion

This chapter outlines the notion of *brokers* in socially-enhanced service-oriented environments such as OESs. These large-scale systems pose various challenges on the system's management and organization. The concept of a broker as an intermediate between an OES community and the activity management is convenient for these environments. Brokers allow organizing the knowledge and skill resources in these environments because of their particular positions in the network, i.e. connectivity features. In particular, the content of the chapter experiments with broker queries that include selections according to collaboration experience related trust relations between the nodes or also distance.

The idea of the broker approach is derived from theories found in social sciences (structural holes). Brokers can be modeled as HPS to support the seamless integration of human capabilities in service-oriented infrastructures. The novelty of the presented approach is that brokers are not discovered based on static policies or static broker capabilities. Instead, the discovery of brokers bases on mining techniques and the automated computation of periodically updated metrics based on interaction logs. This not only helps to find suitable brokers but also relevant communities and social networks to which brokers are connected to. The central part of the chapter is the *Broker Query and Discovery Language (BQDL)* enabling the definition of discovery and interaction policies. BQDL operates on a graph structure that is maintained and updated through mining. In the evaluations, the implementation and performance aspects of BQDL are discussed.

CHAPTER 7

Crowdcomputing: Agreement Management and Preferences Scheduling

7.1 Chapter Overview

The previous chapter took the occasion to emphasize the necessity of intermediates, i.e. brokers in OESs. In particular, according to the observation of current crowdsourcing environments described in the beginning (c.f., Chapter 2), the *persistent exogenous interaction pattern* for brokers outlined in the previous Section 6.3 will gain momentum. Such an exclusive broker supports the observation of the mediated community including all the interaction traffic passing through. This makes the broker an ideal node for interaction logging and behavior observation. In some of the crowdsourcing marketplace environments, e.g. [21, 19], the crowd platform provider itself takes over the role of such a particular broker. The distinguished advantage is that by exclusively mediating the workforce to the requesters and observing the main part of the interactions, a platform provider can learn the behavior of its crowd and improve the management.

This exclusive broker idea motivates the work in this chapter and encouraged discussions on designing a crowdsourcing marketplace environment based on Mixed System technology, named Crowdcomputing in this context. In particular, three broker types have been identified in Crowdcomputing. The described crowd brokers mediate the crowd's workforce, settle agreements, organize activities, schedule tasks, and monitor behavior. At the center of the studies is a redesign and an alignment of the *VieCure* Framework of adaptation presented in Chapter 5. This time its purpose is to self-organize a task processing crowd by monitoring the interactions. At the core, a task scheduling algorithm uses metrics applicable for the interaction data to rank the crowd workers' assignments. The main contribution of this chapter is a hard/soft constraints preferences scheduling algorithm that integrates existing agreement models for SOA systems with Crowdcomputing environments.

Chapter Outline. Section 7.2 provides an insight into the factors that cause unreliability on a crowdsourcing platform. Section 7.3 outlines the details of the broker integration into a crowd scenario with the help of Crowdcomputing. An architectural overview of a Crowdcomputing environment is available in Section 7.4. Section 7.5 discusses the layout of agreements and use of quality metrics for adapting crowd scenarios in order to meet agreements. One promising adaptation approach is dynamic re-scheduling of tasks in crowds which is highlighted in Section 7.6. Section 7.7 shows evaluation results and Section 7.8 concludes the chapter.

7.2 Resource and Agreement Management

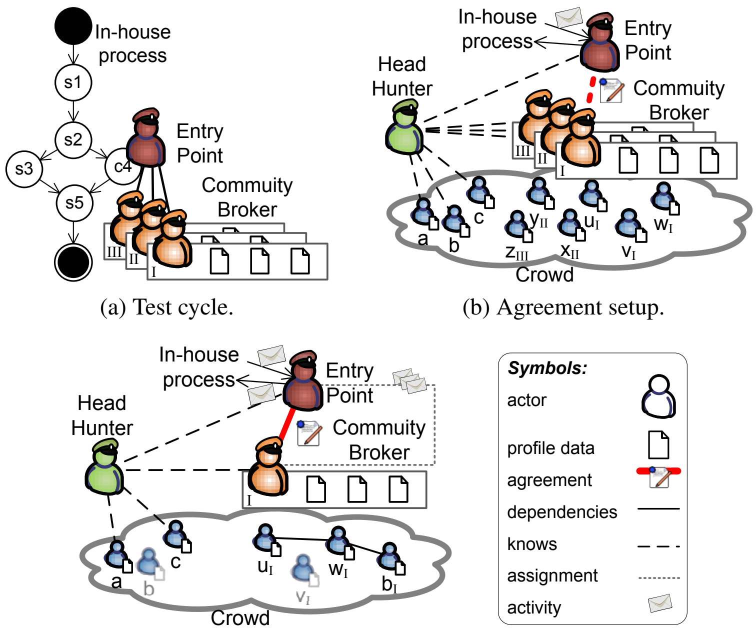
While conventional enterprise systems rely on well established policies, crowdsourcing has a more loosely coupled, dynamic, and flexible structure and depends especially on the preferences and behavior of the individual crowd members. Even if an advantage of a crowd platform is the possibility to choose from a larger number of skilled members, the selection must consider, e.g., the distinguished working hours of the members possibly contradicting with the current requirements. The members availability will mostly depend on their context. Their working hours, for example, also depend on their location and the related timezone. Context does not only influence task assignment strategies but certain changes can also cause unpredictable interruptions of services. This leads to an incomplete and unsatisfactory task state at deadline. As a consequence, meeting promised service contracts is challenging and demands for sophisticated management techniques for a crowd platform.

One of the key issues of the management investigated in this work is to find an appropriate scheduling for task assignments that matches tasks to skills and to the availability of the members, and above all, also meets the agreed contract. This adaptive scheduling strategy must constantly update on changes, and ensure, that shifts in interest, skills, and behavior of members including task-related misbehavior, such as degrading worker performance, refusal of tasks, or missing feedback that affects successful task completion is detected, avoided, and balanced with alternative workforce.

This chapter describes a framework which integrates agreements into SOA-based crowdsourcing platforms. The prerequisite is a monitoring infrastructure that updates a crowd-based resource model. In the previous chapters, *VieCure* Framework (Chapter 5) a monitoring and behavior adaptation architecture serving as the basis for the presented agreement management was presented. Here, the focus is on an agreement model combined with an adaptation approach for reliable task execution. An accurate resource model supports sophisticated scheduling of crowd activities. A self-adaptive mechanism must anticipate misbehavior and update the crowd's task scheduling to enforce behavior rules also dictated by the agreement.

The following are the challenges discussed:

- *Human Behavior Characteristics.* The crowd is a transient network where humans can join and leave the platform at any time. Furthermore, in contrast to software services, human behavior is subject to numerous contextual constraints.
- *Feedback Loop.* Since the crowd's resources, i.e., available members and their capabilities, are in a constant flux and change, models for monitoring the environment and ac-



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

Figure 7.1: Crowdsourcing software tests.

counting for given constraints need to be applied.

- *Quality Guarantees.* It is challenging to provide any guarantees regarding execution time and reliability of actors in highly dynamic environments. The flexible agreement management models tackle that problem by allowing management of negotiated agreements.

7.3 Crowdcomputing Environment

This section outlines a Crowdcomputing environment. With reference to existing testing marketplaces, c.f. [113], a software testing scenario is discussed with the different roles of crowd members involved. Note, the work considers in particular that Crowdcomputing environments base on standard SOA. A prerequisite is that the interaction traffic between and document exchange between the roles presented next bases on WS technology.

Roles in Crowdcomputing

The crowd *Entry Point (EP)* is a mediator that connects customers from outside the crowd with the required crowd members (see Figure 7.1). Generally, crowd customers look for a certain knowledge or capability which their company environments lack. Thus, the EP maintains regular contacts to required crowd members. It acts as representative of a company's outsourced assignments to the crowd and must assure all implications to the dependencies with the company's internals. The *Community Broker (CB)* is a proxy for a certain crowd segment or platform. It maintains and represents a group of registered members with similar capabilities and offers the joint knowledge to interested parties, e.g., EPs. In the example of Figure 7.1, the representative of platform I is the CB of the crowd member group u, v and w with a I subscript each. A CB is in charge of a fair distribution of the incoming assignments and settles agreements. The *Head Hunter (HH)* acts as a kind of registry for a crowd environment. It monitors the tendencies of the required capabilities in the crowd environment, and discovers new knowledge sources (single members or groups). Additionally, it offers an interface that allows new members to register, and further, to be discovered. EPs and CBs can find required partners and members using the HH's lookup service. In the example in Figure 7.1, the HH provides, e.g., for the new crowd members a, b, and c a first point of reference to enter the crowd business. The main concept is that, between entities of these roles, relations based on agreements also need to be established. Agreements state rules that organize and regulate the assignments of tasks in the crowd. They help to assure dependencies and regulate the rather unpredictable behavior of an unorganized crowd.

Use Case Scenario

Figure 7.1 outlines the various phases of a typical crowdsourcing software test scenario. Beginning with Figure 7.1a the in-house quality assurance (QA) process is split into five repetitive and automated steps (*s1*, *s2*, *s3*, the crowdsourcing step *c4*, and *s5*). In *s1* all modules that need to be tested are collected for the next upcoming QA cycle. In *s2* a sorting process differs between automated tests that run in the company's own test environment and those that need to be crowdsourced by, e.g., monitoring a flag on the test units provided by the QA team. Step three (*s3*) represents the test period run in-house. In parallel, in step *c4* a company's representative, EP, is in charge of a smooth flow of the crowdsourced test activities. In order to get into this position, s/he needs to have some previously established relations to available crowd platforms, e.g., *I*, *II*, and *III*, and their representative CBs. Having numerous alternatives for outsourcing guarantees a reliable management of this test period. The final step *s5* collects the testing results for an evaluation and merge.

Next, in Figure 7.1b, the EP has to decide which crowd community can handle the currently pending test activities. The assignment depends on the requirements of the tasks and resources of current platforms, in particular, the members' capacities and capabilities. Next, the EP must balance the effort, expected quality, and costs of the available CBs' offers. If none of the known CBs fits the requirements, the EP can invoke the HH lookup service to find a new CB. Thus a HH mediates CBs to an EP on request. In the example, however, this is not the case and EP contacts the chosen platform I's representative. An agreement for the following assignment is negotiated.

Figure 7.1c illustrates the scenario at runtime. Testing activity segments, i.e. tasks, are delegated to the appropriate crowd members. As mentioned in the introduction, the crowd's structure is transient and its members' processing attitude is context dependent and individual, thus, at times unpredictable. In the given example, nodes u, v, and w process dependent tasks. However, v is not able to process the tasks as scheduled. It is now the challenge of the crowd management to find a solution. A first solution would be to reschedule the task. Unfortunately, in the presented scenario no member can replace v. The CB must call the assistance of the HH and request a fitting, though previously unknown, crowd member. As depicted, member b is mediated to CB of platform I and takes over the duties of node v. Finally, once all tasks have been completed, the results are collected and merged by the EP. Thereafter, the final result is provided to the evaluation step (*s5*).

Agreement Management

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

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

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

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

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

7.4 Architecture

The following section takes advantage of the already detailed *VieCure* Framework for adaptation in Chapter 5. Basing on the fundamental observation/deployment loop structure, in this chapter the modular structure has been rearranged to enable agreement related assignment of tasks and monitoring thereof in Crowdcomputing.

Architectural Overview

Figure 7.2 outlines the three-layer infrastructure of the framework extension. The top layer comprises the *Agreement Manager*. This is a tool-set to monitor, track, and analyze the crowd structure. Additionally, by hiding the particularities of the scheduling technique, it allows extending

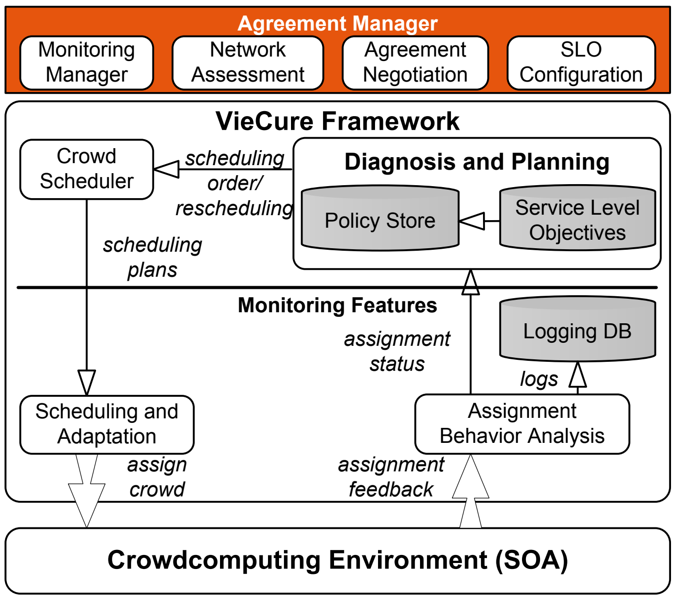


Figure 7.2: Agreement management framework.

and changing the framework's *SLOs* and *Policy Store* entries, thus, adapt the environment to new agreements. The layer in the middle represents the framework itself. It is organized according to an **automated adaptation loop** and its main purpose is to adapt the *Crowd Scheduler*'s assignment strategy. The strategy depends on the current crowd's acceptance behavior and capacities, as well as, on policies representing system and agreement constraints. Therefore, interfacing with the environment, the *Assignment Behavior Analysis* collects feedback to a log database and forwards the current status to the *Diagnosis and Planning* module. Considering the valid policies and the fresh assignment status from analysis this module adapts the scheduling order, and/or, on an assignment reject, issues a rescheduling directive with new ordering rules to the *Crowd Scheduler*. Depending on the current situation the *Crowd Scheduler* uses its algorithm to assign a batch of tasks, or on request of *Diagnosis and Planning* module, reschedules an unsuccessful assignment. Finally, the scheduling result is transmitted to the *Scheduling and Adaptation* module. This deploys the assignments and scheduling changes to the crowd.

Further Building Blocks

The bottom layer of the architecture in Figure 7.2 bases on SOA. In the previous chapter, the possibilities to embed flexible interactions, metrics for crowds, and monitoring of collaboration environments with an SOA infrastructure have been studied. Next, a quick summary for a more complete picture of the architecture is provided.

SOA-based Interactions in Crowds. Dynamic discovery of services, flexible interactions and compositions at run-time are only some properties that make SOAs an intuitive and convenient technical grounding for large-scale crowdsourcing environments. However, not only service interactions, but also human interactions may be performed using SOAP (see HPSs) for collaborative environments and BPEL4People [3] for human interactions in business processes), which is the state-of-the-art technology to exchange XML-based messages in service-oriented environments, and well supported by a wide variety of software frameworks.

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

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

Agreement Life-Cycle

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

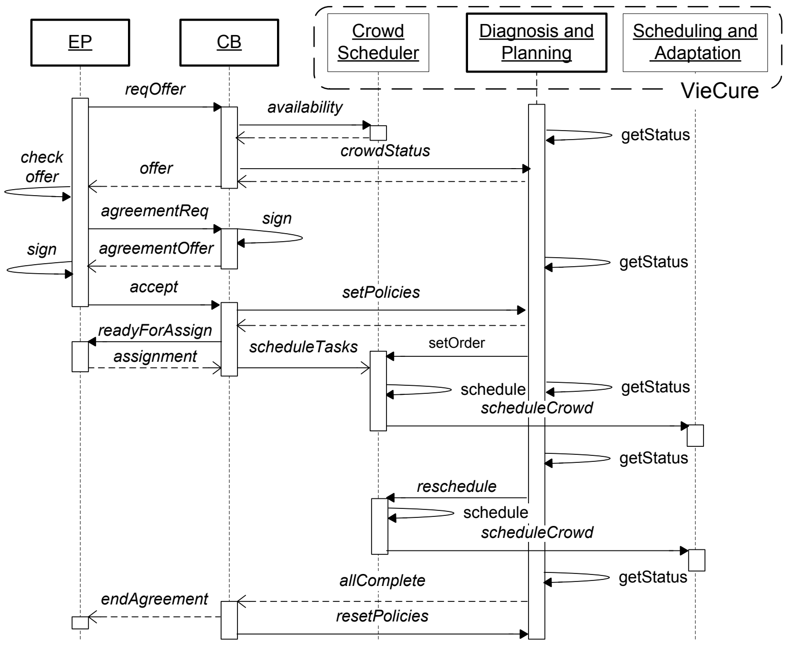


Figure 7.3: Agreement management life-cycle.

The sequence in Figure 7.3 presents an agreement's life-cycle. It details the interactions between the involved parties, EP and CB presented in Section 7.3, and the automated *VieCure* Framework crowd assignment management.

Negotiation. While an EP browses for interesting bids, the CB uses the adopted *VieCure* Framework to create offers within the limits of the current crowd's capacities. On a request an individual offer can be created and provided. The negotiable items of a later assignment and their boundaries depend on the available resources and their current scheduling. Both can be gathered by checking availability at the *Crowd Scheduler* and the current crowd member status at the *Diagnosis and Planning* module. The provided offer is revised by the EP comparing it against the in-house requirements. If not pleased, negotiation starts over again or a different CB is considered as service provider. Finally, a satisfying offer including the agreement details in the objectives is signed by both parties and enacted with the according policies for assignment management.

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

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

In the line with the requirements emerging from the agreement's life-cycle, the next section details a structure for the agreements and discusses examples of fundamental quality attributes applicable for Crowdcomputing.

7.5 Agreements and Quality

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

Agreement Structure

There has been substantial research on translations of **service level agreements** to a Web service applicable standard. Two of the main contributions are the specification from the Grid Resource Allocation Agreement Protocol (GRAAP) Working Group [6] and from IBM [59]. Both present similar XML-based model for an SLA, however, differ in the details. When creating their specification the GRAAP Working Group in particular focuses on the setup, negotiation, and renegotiation phases of the agreement, thus, presents a rather flexible structure [60]. IBM's WSLA focus was on defining agreement objectives, their constraints and combination. For this purpose parameters can be linked to SLOs together with thresholds. The used agreement model is inspired by, both, the work of IBM and the GRAAP Working Group.

The overall structure includes header, agreement items, and terms. The header comprises the agreement's parties details and contact information. In the contractual items the agreement's subjects are listed. These include the service content (i.e., in Web service environments WSDL location, endpoint, and operation) along with scheduling information, metrics and their measuring method. Finally, the terms provide the objectives, SLOs respectively, and their validity period. Threshold values express the desired relation between objectives and metrics defined in the items. For a dynamic environment such as a Crowdcomputing the parameter scheme is reused, however, extended to define applicable quality attributes.

For this reason the next section, provides a number of quality metrics that can be applied and measured in SOA based Crowdcomputing environments, and thus, aligned to the described structure.

Quality Parameters

Analyzing the requirements of the scenario in Section 7.3, Table 7.1 represents a plausible list of quality attributes for negotiation in crowdcomputing. The crowd broker manages the attributes for registered crowd members and constantly updates their value.

Table 7.1: Negotiated quality attributes.

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

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

7.6 Task Scheduling in the Crowd

Next, an example of an agreement in extended Web Service Level Agreement (WSLA)[1] notation is provided. The format is XML-based, thus, processable for the phases of negotiation, and extraction of agreement items and objectives. Further, this helps to fit the extracted hard- and soft-constraints into a self-adaptive scheduling algorithm that is formalized in Algorithm 7.1.

Agreement Setup

The XML examples in Listing 7.1 and Listing 7.2 detail a sample WSLA agreement applicable to Crowdcomputing environments. Only the important parts are fully listed. Additionally, the structure has been extended to fit the Crowdcomputing particulars.

After the contract parties' details, *ServiceProvider* and *ServiceConsumer* listed in lines 6 to 13, Listing 7.1 states the items from line 14 to 37. These are a collection of *ServiceObjectType* items including scheduling, operation description, and configuration, and also, an *SLAParameter* (`FeedbackExpected`) arranging periodic feedback. Important and a new contribution to this part is the *SLAActivity* (lines 25 to 36). It extends WSLA's *ServiceObjectType* and states what kind of member capabilities must be involved in the testing activity (line 27), at which URI the final testing reports are expected (line 30), and at the end, the *SLAParameter*s for the assigned activity.

These include for example, all the quality attributes as presented previously in Table 7.1. The parameter is identified by a name, a type, a unit, and related to a metric for estimation. Listing 7.2 shows the terms as *Obligations* of the contract including all SLOs. An SLO consists of an obliged party, a validity period, and *Expression*s that can be combined with a logic expression (e.g., *And*). The content of the expressions connects the pool of *SLAParameters* of the items to a predicate (e.g, `Equal`) and threshold value (*Value*). The final tag *QualifiedAction* defines the consequence of an SLO violation. In the example case, if a threshold of SLO `sloAct` is violated an action *Notification* is called.

Scheduling Algorithm

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

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

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

```xml
<wsla:SLA
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsla="http://www.ibm.com/wsla"
xmlns:hps="http://www.infosys.tuwien.ac.at/hps/"
name="SwTestCrowdSLA5312">
 <wsla:Parties>
  <wsla:ServiceProvider name="CommunityBroker">
   <!-- ... -->
  </wsla:ServiceProvider>
  <wsla:ServiceConsumer name="EntryPoint">
   <!-- ... -->
  </wsla:ServiceConsumer>
 </wsla:Parties>
 <wsla:ServiceDefinition name="TestService">
  <wsla:Operation
    xsi:type="wsla:WSDLSOAPOperationDescriptionType"
    name="AddActivity">
   <!-- schedule period -->
   <wsla:SLAParameter name="FeedbackExpected"
     type="int" unit="Days">
    <wsla:Metric>CountDays</wsla:Metric>
   </wsla:SLAParameter>
   <!-- config details: name, wsdl-location, binding -->
  </wsla:Operation>
  <hps:SLAActivity name="Testing">
   <hps:SLAInvolvedProfiles>
    <hps:SLAProfileTyp>UI Test</hps:SLAProfileTyp>
    <!-- ... functional, performance, security -->
   </hps:SLAInvolvedProfiles>
   <hps:SLAReportURI>http://.../reports</hps:SLAReportURI>
   <wsla:SLAParameter name="TasksCost"
     type="float" unit="Euro">
    <wsla:Metric>MaxCost</wsla:Metric>
   </wsla:SLAParameter>
       <!-- reliability, load, overlap -->
  </hps:SLAActivity>
 </wsla:ServiceDefinition>
<!-- wsla Obligations -->
</wsla:SLA>
```

Listing 7.1: Involved parties and body.

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

In the following the steps of the crowd scheduling algorithm are detailed. Let us define the set of crowd members $U = \{u_1, u_2, \ldots\}$ and the set of tasks $T = \{t_1, t_2, \ldots\}$ to be processed by the crowd. The goal of the algorithm is to assign each task to one individual crowd member. Notice, no assumption is made about the role of the broker. In fact, the broker may be implemented as a software service, thus the procedure is fully automated, or the procedure may be performed under human supervision.

Given the loop in Algorithm 7.1 (lines 3 to 26), three essential steps are performed:

Matching. A set of members whose profiles satisfy a task's required capabilities (see line

Algorithm 7.1: Task scheduling in the crowd.

```
Require: $T \neq \emptyset \wedge U \neq \emptyset$
begin
    $AT \leftarrow \emptyset$                               /* Set of assigned tasks */
    $FT \leftarrow \emptyset$                      /* Set of failed (to assign) tasks */
    foreach Task $t \in T$ do
                      /* Retrieve matching members with $U' \subseteq U$  */
        $U' \leftarrow$ getAllMembersMatchingTask ($t$);
        foreach Member $u \in U'$ do
                                 /* Check additional constraints */
            if (meetsDeadline ($u$, $t$) == false) $\vee$ ;
            (meetsResponseTime ($u$, $t$) == false) then
                continue             /* Do not consider as candidate */
            end
            $score_{(u,t)} \leftarrow$ calculateScore ($u$,$t$) ;
            $U'' \leftarrow$ insertByScore ($score_{(u,t)}$,$U''$)
        end
        $CM[t] \leftarrow U''$                                  /* Save $CM$ */
        foreach Member $u \in CM[t]$ do
            ;
            if assignTask ($u$,$t$) == true then
                ;
                $AT \leftarrow AT \cup t$ ;
                break ;
            end
            else
                $CM[t] \leftarrow CM[t] \backslash u$                /* Remove $u$ from $CM$ */
            end
        end
        if $t \notin AT$ then
            $FT \leftarrow FT \cup t$                           /* Add $t$ to $FT$  */
        end
    end
end
```

```
<wsla:Obligations>
<wsla:ServiceLevelObjective name="sloSrv"
     serviceObject="AddActivity">
 <wsla:Obliged>CommunityBroker</wsla:Obliged>
 <!-- Validity -->
  <wsla:Expression>
   <wsla:Predicate xsi:type="wsla:Equal">
    <wsla:SLAParameter>FeedbackExpected</wsla:SLAParameter>
    <wsla:Value>7</wsla:Value>
   </wsla:Predicate>
  </wsla:Expression> <!-- evaluation weekly -->
</wsla:ServiceLevelObjective>
<wsla:ServiceLevelObjective name="sloAct"
     serviceObject="Testing">
 <wsla:Obliged>CommunityBroker</wsla:Obliged>
 <!-- Validity -->
 <wsla:And>
 <wsla:Expression>
  <wsla:Predicate xsi:type="wsla:LessEqual">
   <wsla:SLAParameter>TasksCost</wsla:SLAParameter>
   <wsla:Value>50.0</wsla:Value>
  </wsla:Predicate>
 </wsla:Expression>
 <!-- expressions for reliability, load, overlap -->
 </wsla:And>
 <wsla:EvaluationEvent>TaskAssignment</wsla:EvaluationEvent>
</wsla:ServiceLevelObjective>
<wsla:QualifiedAction>
 <wsla:Party>CommunityBroker</wsla:Party>
 <wsla:Action actionName="violation" xsi:type="Notification">
   <wsla:NotificationType>Violation</wsla:NotificationType>
   <wsla:CausingGuarantee>sloAct</wsla:CausingGuarantee>
   <wsla:SLAParameter>MaxCost</wsla:SLAParameter>
   <!-- expressions for reliability, load, overlap -->
 </wsla:Action>
</wsla:QualifiedAction>
</wsla:Obligations>
```

Listing 7.2: Obligations and SLOs.

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

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

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

$$score_{(u,t)} = \left[\sum_{m \in M'} |w_m| \times score(u,m)^p \right]^{1/p} \tag{7.1}$$

The detailed parameter description can be found in Table 7.2. This approach is based on a model for *simultaneity* and *replaceability* of preference parameters known as Logic-Scoring of Preferences (LSP), e.g. [26].

The parameter p can be assigned manually based on the desired scoring behavior [26] or calculated automatically. Here a simple pattern is used to calculate p based on the homogeneity (or diversity) of the preference weights $w_m \in W$ where W is the set holding preference weights for each metric m: if $\texttt{max}(W) - \texttt{min}(W) > \texttt{avg}(W)$ use $p = 1.5$, if $\texttt{max}(W) - \texttt{min}(W) = \texttt{avg}(W)$ use $p = 1$, otherwise use $p = 0.5$. This means that replaceability should be preferred over simultaneity if the weight values (preferences) vary highly expressed by the relationship between `max` and `min` values compared to the average (`avg`) weight.

7.7 Simulation and Evaluation

The main idea of the evaluations is to identify the boundaries of the scheduling algorithm presented in Section 7.6 in Crowdcomputing environments to support reasonable negotiations of quality attributes in agreements. Monitored with the metrics defined by common crowdsourcing quality parameters presented in Section 7.5. The setup includes a simulated Crowdcomputing environment with properties related to the environment outlined in the scenario. Only a subgroup of a larger and more complex Crowdcomputing network is considered because in contrast to existing environments with the broker approach resources can be organized for specific skills. In particular, the challenges and effort of scheduling, and also of rescheduling, tasks for differently behaving crowd members are studied.

Table 7.2: Description of ranking procedure.

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

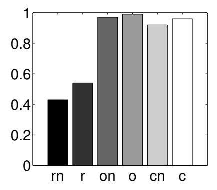
Environment Setup

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

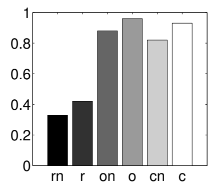
Each **crowd member** has its own capability profile. Additionally, the member exposes a predefined behavior in task assignment and task processing. The acceptance behavior in task assignment depends on the current task queue size. Whilst on an empty queue the member is eager to get task assignments,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

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

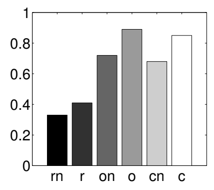
At bootstrap the framework instantiates broker and members and fills the registry. The runtime is split in two phases including **scheduling and rescheduling** activities. Both phases apply



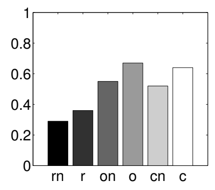
(a) Processed - 256 batch.



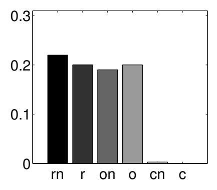
(b) Processed - 512 batch.



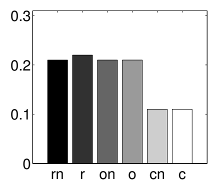
(c) Processed - 768 batch.



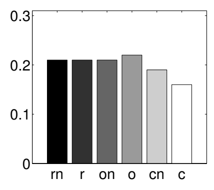
(d) Processed - 1024 batch.



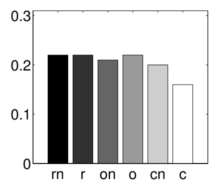
(e) Over cost - 256 batch.



(f) Over cost - 512 batch.



(g) Over cost - 768 batch.



(h) Over cost - 1024 batch.

Figure 7.4: Results of the experiments for various scheduling strategies with different task batch size. Strategies include *random* scheduling with *no* rescheduling phase *(rn)* and with rescheduling *(r)*, according to metrics *ordered* scheduling with *no* rescheduling (on) and with rescheduling (o), and finally, metric-ordered scheduling including *costs* with *no* rescheduling *(cn)* and with rescheduling *(c)*.

Algorithm 7.1 to assign a batch of tasks. Members have different behavior in accepting and processing tasks. Thus, the goal of the two phases is to minimize the task failure rate, and in the later experiments, to minimize the costs of processing by choosing the currently less expensive crowd member. Two different scheduling **strategies** have been implemented and evaluated. The first one, a random strategy, picks from the set of available and capable crowd members randomly the next candidate for the assignment. The second one, the metrics assisted strategy, uses the previously presented metrics in Table 7.1 to select the next candidate from the set. During an independent task assignment the metrics weight factor is equally distributed. Nevertheless, this strategy is also aware of task dependencies. Once a parent task is assigned, an extra weight (60%) is set to the *overlap* metric to move the most overlapping members to the begin of the selection queue.

Experiment Results

The main goal of the experiments is to illustrate the effectiveness of the metrics enhanced scheduling algorithm outlined in Algorithm 7.1; also compared to random scheduling. In the first set of experiments Figures 7.4a, 7.4b, 7.4c, and 7.4d, the percentage of *completed tasks* with respect to the total number of assigned tasks is an indicator for the effectiveness. In the next set of figures (7.4e, 7.4f, 7.4g, and 7.4h), the *exceeding costs* are considered. This second type of experiments illustrates the percentage of completed tasks that exceed the SLA cost objective. As aforementioned, this is caused by members that exceed the cost with their fee. The results of the two sets are presented by the rows in Figure 7.4. Furthermore, to explore lower, and in particular, upper bounds of the algorithm different batch sizes of tasks are scheduled. A batch size defines the number of tasks that are received from the EP and need to be scheduled in the next period. In the simulations, once all tasks of a batch assignment are past their deadline, a new similar size batch is scheduled until a number of 10000 tasks total are assigned. Note, the fixed size crowd has a maximum capacity of 768 tasks per period. The columns of the results in Figure 7.4 represent the different batch sizes.

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

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

The scheduling effort is another cost source of task assignments. Referring to the experiments in Figure 7.5, the assignment failure rate (*afr*) for the different batch sizes and varying

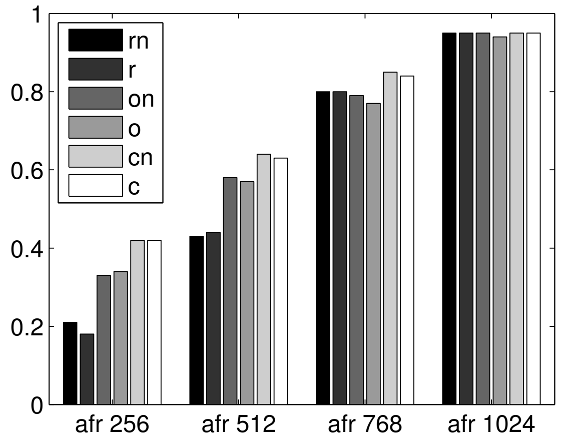


Figure 7.5: Assignment failure rate (afr) for different batch sizes.

strategies is listed. The results show the percentage of scheduling requests that are rejected on the first request. The results highlight how the increasing batch size reduces the impact of the *afr* for the metric-assisted strategies ((**on**),(**o**), (**cn**), and (**c**)). These strategies force similar member selections for comparable tasks. Thus, for small batches and cost on focus, the low fee members reject on full queues. For size 256 and 512 the effort difference between (**rn**), (**r**) and (**cn**), (**c**) is around 20% and decreases rapidly for larger sizes.

7.8 Conclusion

In this chapter an alignment variant of the Mixed System model with crowdsourcing marketplace environments was presented. In this distinct collaboration environment, named Crowdcomputing, two types of exclusive broker roles are identified and closely investigated. The first derives from the studies in the previous chapter and is a well connected node that can represent a community of workers. The second broker type mediates between these crowd community brokers and a company wishing to outsource to the matching crowd. The presented methodologies to identify and submit a task to the best matching community assume that actors in Crowdcomputing interact on an SOA infrastructure. This not only allows monitoring and analyzing the interaction traffic as outlined in previous chapters and OESs but also to takes advantage of the already existing models for agreements (WSLA and WS-Agreement). An important part of the chapter focuses on an agreement design feasible for contracts between the two brokers.

As the results emphasize, the combined information from agreements and crowd interaction logs support the crowd brokers in an objective-aware metric ordered strategy for task assignments. These objectives can then be used as soft- or hard-constraints for a weighted multi-objective ranking of the workers. The results of the experiments highlight the advantages of an objective-aware metric ordered strategy in contrast to plain random scheduling while task loads remain in between the boundaries. Nevertheless, the results show, the effort for ordering the assignment lists induces a higher effort in scheduling. The studies on tasks with different skill requirements and quality criteria related scheduling have been continued in the work in [50].

CHAPTER 8

Skill Evolution and Auctioning

8.1 Chapter Overview

A major problem in crowdsourcing is to guarantee a high-quality processing of tasks. So far the approach in this part has been to consider a mediator, i.e. a broker, as an emerging instance of the community. Its particular function in the network can be extended to represent the community and take over part of the responsibility. As a further step, the solutions in this chapter focus on the individual crowd workers. Form the perspective of a platform provider in the role of an aforementioned crowd broker, the challenges are how to improve the workers' skills and how to maintain a regularly available crowd vigilant for new tasks.

In the solution, a novel crowdsourcing marketplace is presented that matches tasks to suitable workers based on their skill profile. The key to ensure high quality tasks is provided by both, the ability to estimate more precisely the members' skills and announce the tasks in auctions. By observing the workers' performance their real skills can be analyzed. Updating the crowd's profiles also allows for an automated ranking between candidates and tasks. Furthermore, only capable workers are allowed to participate in the auctioning process that guarantees the best price for the offered work. Training tasks further assist the platform in gaining confidence in their workers' skills and increasing its popularity. Tandem assignments support the estimation of new crowds members in comparison to established. It further helps the crowd members to improve and to learn new skills. Overall, the platform is capable to retain a motivated, loyal crowd of skilled workers.

Chapter Outline. In Section 8.2 the motivation for skill evolution in crowdsourcing is outlined. Section 8.3 describes the design of the crowdsourcing system, including its actors and their interaction. Then, Section 8.4 details the adaptive auction mechanisms and Section 8.5 presents the conducted experiments and discusses their results. A conclusion reviews the chapter.

8.2 Introducing Skill Evolution to Crowdsourcing

Managing and adapting the crowd's skills and resources in an automated manner in crowdsourcing remains challenging. Crowd customers prefer fully automated deployment of their tasks to a crowd, just as in common business process models. A preliminary solution would base crowdsourcing on the SOA paradigm. SOAs are an ideal grounding for distributed environments. With their notion of the participants as services and registries, resources can be easily and even automatically discovered for composing whole business processes. A plethora of standards supports seamless integration and registration of new services, and provides protocols for communication, interaction and control of the components. Altogether, it seems that SOAs provide an intuitive and convenient technical grounding to automate large-scale crowdsourcing environments.

The main challenges addressed in this chapter relate to building and managing an automated crowd platform. It is not only of importance to find suitable workers for a task and to provide the customer with satisfying quality, but also, to maintain a motivated base of crowd members and provide stimulus for learning required skills. Only a recurring, satisfied crowd staff is able to ensure high quality and high output. As any crowd, fluctuations must be compensated and a *skill evolution* model must support new and existing crowd workers in developing their capabilities and knowledge. Finally, the standard processes on such a platform should be automated and free from intervention to handle the vast amount of tasks and to make it compatible with an SOA approach. Atop, the model should increase the benefit of all participants.

In detail, the presented crowdsourcing marketplace extension considers the following:

- *Automated matching and auctions.* For providing a beneficial distribution of the tasks to the available resources auctions are organized according to novel mechanisms.
- *Stimulating skill evolution.* In order to bootstrap new skills and unexperienced workers skill evolution is embedded into the auction model by integrating assessment tasks.
- *Extension Verification.* Experiments quantify the advantages of a skill evolution based approach in comparison to traditional auctions.

8.3 Design of Marketplaces in Crowdsourcing

The core activity in task-based crowd environments is members providing their labor by processing tasks. This section explains the idea of task-based crowdsourcing on a market-oriented platform. The aim is to organize and manage the platform to the satisfaction and benefit of all participants; crowd members and platform provider. Next, the basic design of the proposed crowdsourcing environment will be introduced.

Skill-based Task Markets

In task markets different stakeholders can be identified. Generally, there is the requesters and workers representing the registered members of a crowd marketplace. The task of the third stakeholder, the crowd platform in between, is to manage the crowd task auctions. To satisfy any of the stakeholders the platform must assure that the requesters obtain a result of high quality in a timely manner. On the other hand, the workers would like to have tasks available whenever they are motivated to work and are interested in a high reward for processing a task. The platform itself works towards a long-term profit. To bootstrap the skill-based system, each member interested in offering of processing tasks is required to create a profile containing information about her/his interests and skills. The basic interactions and an external view on the proposed crowdsourcing environment are depicted in Figure 8.1.

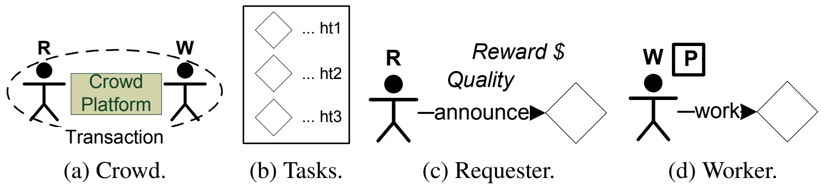


Figure 8.1: Crowd environment building blocks and interaction of stakeholders.

The crowdsourcing environment consists of members who can participate in transactions (see Figure 8.1a). Within a particular transaction a member can either adopt the role of a requester **R**, who initiates a transaction by announcing tasks (see Figure 8.1b), or the role of a worker **W**, who processes a task. The proposed crowdsourcing marketplace handles transactions transparently for its members; requesters and workers do not communicate directly, but only with the crowdsourcing marketplace in between. The argument is that this standardized style of interaction is less prone for misconceptions and more efficient because it allows members getting used to the system. Tasks (Figure 8.1b) are created by requesters based on their current needs. Requesters initiate a transaction by submitting a task to the marketplace, with additional information about the amount of money he is willing to pay for the processing of the task and additional requirements (Figure 8.1c). It is the responsibility of the marketplace platform to find a suitable worker, to submit the task to the worker, to collect the result, and to transmit it to the requester.

The interaction of a worker with the market platform is initiated by the latter by asking a member whether s/he is interested in processing a task (Figure 8.1d). This interest can be expressed by bidding for the task. Workers have skill profiles denoted by the symbol **P**. These profiles are not statically defined, but are updated based on the level of delivered task quality. This procedure ensures an up-to-date view on workers' capabilities and skills. Based on the bids and background information about the bidders, the system selects one or multiple workers, who are then asked to process the task.

Towards Auction-based crowdsourcing

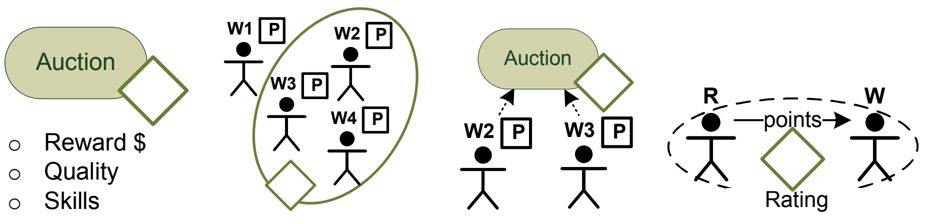
Auctions are a very old idea already used by the Babylonians but still an active area of research. The rise of e-commerce has drastically increased the number and diversity of goods traded via auctions. Many recently installed markets, such as energy or pollution permit markets, are based on auctions [53]. There are many different flavors of auctions differing in the number of items considered (single/multi item), the number of buyers and sellers (demand/supply/double auction), the bidding procedure (open/closed bids and ascending/descending), and how the price is determined (e.g., first/second price); however, four standard types are widely used [53]. They all assume a single seller and multiple buyers (demand auction) and, in their simplest forms, a single item to sell (single-item auction). The so-called English auction is an ascending open-bid auction; the Dutch auction is a descending open-bid auction. The other two standard auction types are closed-bid auctions, i.e., each bidder submits a single bid which is hidden for other buyers. The auction presented in this chapter uses an adapted version of a closed-bid auction; a single auction deals with the matching of one task to one or many crowd workers (single-item demand auction). The details of the mechanism are introduced in the following sections.

8.4 Auction-based Task Assignment

The following section discusses the steps involved in transaction processing from task announcement to worker rating. Then the novel idea of skill evolution on the basis of auctioning for crowdsourcing is presented.

Processing of Transactions

Figure 8.2 illustrates the steps involved in the internal processing of a transaction. In the qualification step the marketplace identifies all members capable of processing the task (Figure 8.2a), based on the task description and the members' profiles. The preselection chooses a limited number of the most suitable workers (Figure 8.2b) to have a reasonable amount of participants for the auction. The preselection step helps to avoid a flooding of auction announcements. In this way members are only exposed to tasks/auctions for which they actually have a realistic chance of being accepted as worker. Due to the voluntary, open nature of crowdsourcing environments, not all preselected workers may decide to follow the invitation to compete in an auction.



(a) Announcement. (b) Preselection. (c) Biddings. (d) Feedback.

Figure 8.2: Internal processing of a transaction.

This fact is depicted by the transition phase between Figure 8.2b and Figure 8.2c where only a subset of preselected workers decides to participate. The auction phase (Figure 8.2c) allows each participant to submit an offer and finally, a winner is determined who is supposed to process the task. In the case of a successful processing the marketplace returns the final result to the requester, handles the payment, and allows the requester to give feedback about the transaction in the form of a rating (Figure 8.2d).

As mentioned before, tasks come with a description, a maximum amount of money the requester is willing to pay and further requirements, i.e., time requirements and quality requirements. The former is typically given in the form of a deadline, the latter could range from a simple categorization (e.g., low, high) to sophisticated quality requirement models. Each worker has a self-provided profile describing her/his skills and, additionally, the marketplace platform is keeping track of the actual performance. A further idea is to maintain a performance value per user and skill, encoded as tuple consisting of the observed performance and the confidence in that value. The input used to generate these performance values comes from the ratings of the requesters and a check whether the deadline was met, which can be performed by the system without feedback from the requester. The qualification phase is based on a matching of the task description to the skills of the members considering their performance and confidence values. Higher requirements impose higher standards on the performance of the member. The result of this matching is a boolean value indicating whether a member is meeting the minimum requirements. In the next step, the preselection, the qualified members are ranked based on skill, performance, and the confidence in the performance; only the top-k members are chosen to participate in the auction. This helps to reduce the number of auction requests to members in order to avoid spamming members and to spare members the frustration caused by not winning the auction. The marketplace platform as the auctioneer hides parts of the task's data. Workers only see the task description and the time and quality requirements, but not the associated price determined by the requester. The auction is performed as a closed bid auction, whereas each participant is only allowed one bid. At the end of the auction a winner is determined based on the amounts of the bids and the performance-confidence combination of the bidders' skills. If all bids are higher than the amount the requester is willing to pay the auctioneer would typically reject all bids and inform the requester that the task cannot be assigned under the current conditions. In this case the requester could change the task by increasing the earnings, lowering the quality requirements or extending the deadline and resubmit the task. With a selection strategy outlined in more detail in the next section the marketplace assigns the task to the worker for processing. After the processing of the task by the worker and the receipt of rating information, the performance of the worker is adjusted and the confidence value is increased. Technically, an aptitude function estimates how well workers are suited for handling a task. It is used as basis for qualification and preselection and can be formally defined as

$$aptitude : W \times T \rightarrow [0, 1], \tag{8.1}$$

where W is the set of workers and T represents tasks. $aptitude(w, t) = 1$ would mean that worker $w \in W$ is perfectly qualified for handling task $t \in T$. A mapping to zero would represent a total inaptness. Similarly, a ranking function is used to rank workers' bids:

$$rank : W \times T \times B \rightarrow [0, 1], \tag{8.2}$$

where B is the set of bids. In addition to the aptitude, the $rank$ function also takes monetary aspects, contained in bid $b \in B$, into account. A property of a sound ranking function is that if two workers have the same aptitude for a task then the one with the lower bid will have a higher rank. The $aptitude$ function is used for performing qualification and preselection. As auction admittance strategy one can either admit all workers with an aptitude higher than a certain threshold, the top-k workers according to aptitude, or a combination of the two strategies. The ranking function is used to determine the winner of an auction: the highest ranked worker.

Skill Evolution

The concepts discussed so far provide the fundamentals for automated matching of tasks to workers. As outlined, a major challenge hampering the establishment of a new service-oriented computing paradigm spanning enterprise and open crowdsourcing environments are quality issues. In the presented scenario this is strongly connected to correctly estimating the skills of workers. One approach for increasing the confidence in worker skills are qualification tasks, with the shortcoming that these tasks would need to be created (manually) by the requesters who have the necessary knowledge. This implies a huge overhead for the testing requester; s/he is also the only one who benefits from the gathered insights. Here, a different approach is taken by integrating the capability of confidence management into the crowdsourcing platform itself. Instead of having point-to-point tests, workers are automatically assessed to unburden requesters in inspecting workers' skills. The advantage of this approach is the great potential for the (semi-)automatic inclusion of crowd capabilities in business environments. The first challenge addressed is to cope with the "hostile" environment in which computing is performed. Workers may cheat on results (e.g., copy and paste of existing results available in the platform), spam the platform with unusable task results, or even provide false information. A well-known principle in open, Web-based communities is the notion of *authoritative* sources that act as points of references. For example, this principle has been applied on the Web to propagate trust based on *good seeds*. The presented idea of skill evolution is in a manner similar. It proposes the *automatic assessment* of workers where confidence values are low. For example, newcomers who recently signed up may be high or low performers. To unveil the tendency of a worker, it creates a hidden 'tandem' assignment comprising a worker whose skills are known (high performer) with a high confidence and a worker where the crowdsourcing platform has limited knowledge about its skills (i.e., low confidence). The next step is that both workers process the *same* task in the context of a requester's (real) task. However, only the result of the high confidence worker is returned to the requester, whereas the result of the low confidence worker is compared against the delivered reference. Skill evolution through tandem assignments provides an elegant solution to avoid training tasks (assessments are created automatically and managed by the platform) and also implicitly stimulates a learning effect. Of course, the crowdsourcing platform cannot charge the requester for tandem task assignments since it mainly helps the platform to better understand the true skill (confidence) of a worker. Thus, the platform must pay for worker assessments. However, the later evaluation will show, performing assessments provides the positive effect that the overall quality of provided results and thus requester satisfaction increases due to a better understanding of worker skills. Embedding skill evolution in the crowdsourcing platform works as follows. After the winner of an auction has been determined it is evaluated whether an assessment task is issued to further workers. The function $assess$ outputs 1 if an assessment task is to be assigned to a worker and 0 otherwise.

$$assess : W \times T \times B \times W \rightarrow \{0, 1\} \tag{8.3}$$

An input tuple (w, t, b, w_r) checks whether tasks $t \in T$ is to be assigned to $w \in W$ who offered bid $b \in B$. Worker $w_r \in W$ is the reference worker, in this case the worker who has won the corresponding auction and who will thus process the same task.

8.5 Simulation and Evaluation

This section details the aspects of the implemented simulation environment. The experiments design is explained and the results of the experiment runs discussed.

Simulation Environment

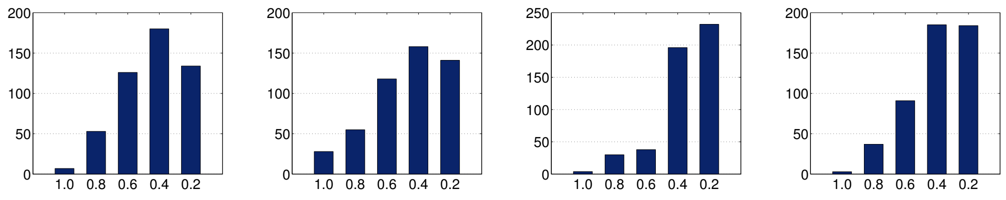
The simulation framework is Java-based and supports all previously introduced concepts and interactions between requesters, the platform, and workers. All of the above introduced functions (8.1)-(8.3) have been implemented in the framework. The interested reader can find details regarding the prototype as well as a Web-based demo online[1].

Experiment Design and Results

An evaluation scenario consists of a set of workers W and a set of requesters R. In every round of the simulation each requester usually announces a task. An auction is conducted for each announced task t, which consists of a description of the skills needed for its processing, an expected duration, a deadline, and the expected quality. High quality requirements indicate highly sophisticated and demanding tasks. For each worker w and skill s the platform maintains a performance value $pfmc(w, s)$ and a confidence in that value $cnfd(w, s)$. This observed performance value is derived from requester ratings; if it is based on many ratings the confidence is close to one, if there are only a few ratings available the confidence is close to zero. Based on task t's skill requirements and a worker w's performance/confidence values for these skills it is possible to calculate the expected performance $pfmc(w, t)$ and confidence $cnfd(w, t)$ for that task. For the evaluation it is assumed that each worker w has a certain performance $pfmc_{real}(w, s)$ for a skill s which is hidden but affects the quality of the results. Requesters rate the workers based on the results which in turn is the basis for the observed performance and confidence values. Further, the assumption is that the processing of tasks demanding a certain skill causes a training effect of that skill, i.e., $pfmc_{real}(w, s)$ increases. For the sake of simplicity the evaluations consider only one skill is required. This does not change fundamental system properties and one skill allows extrapolating the behavior of multiple skills. Whether a single or multiple skills are considered indeed affects qualification, preselection, bidding, and rating but all the mechanisms are naturally extensible from one to multiple skills by performing the same computations for each skill and a final combination step. Hence, $pfmc : W \times S \to [0, 1]$ and $pfmc : W \times T \to [0, 1]$ are reduced to $pfmc : W \to [0, 1]$. The same holds for $cnfd$ and $pfmc_{real}$.

For the simulation 500 workers with random values for $pfmc_{real}(w)$ and $cnfd(w)$ according to a normal distribution $\mathcal{N}(\mu, \sigma^2)$ have been created. The initial performance value

[1] `http://www.infosys.tuwien.ac.at/prototyp/Crowds/Markets_index.html`



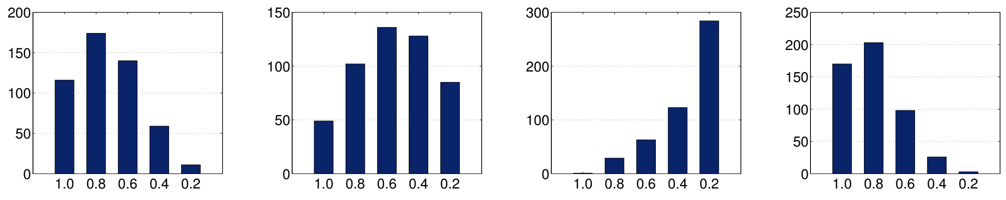
(a) real pfmc (b) obs pfmc (c) diff real/obs (d) cnfd

Figure 8.3: Generated worker population according to Scenario 1.

$pfmc(w)$ is set according to the formula $pfmc_{real}(w) + \mathcal{N}(0,\, 1 - cnfd(w))$ which ensures that for high confidence the expected deviation of $pfmc(w)$ from $pfmc_{real}(w)$ is small and for low confidence values it is high, respectively. All values are restricted to the range of $[0, 1]$. The following figures illustrate the simulation setup in detail.

Scenario 1 assumes that there are three requesters (i.e., typically three tasks are issued in every round). It is an environment in which skilled workers are rare and the confidence in the workers' performance is relatively low, i.e., there are many workers who have few ratings. The real performance $pfmc_{real}(w)$ for a worker w is drawn according to $\mathcal{N}(0.3,\, 0.25)$, the confidence value $cnfd(w)$ is randomly generated by $\mathcal{N}(0.2,\, 0.25)$. Given the two generated values $pfmc_{real}(w)$ and $cnfd(w)$ the observed performance is randomly drawn according to $\mathcal{N}(pfmc_{real}(w),\, 1 - cnfd(w))$. Hence, a low confidence in the performance leads to highly distorted values for $pfmc(w)$, higher confidence values decrease the variance. Figure 8.3 gives a detailed view on the statistical distributions of the workers' performance and confidence in the experiments.

The histograms count the number of workers in the buckets $[0, 0.2)$, $[0.2, 0.4)$, $[0.4, 0.6)$, $[0.6, 0.8)$, and $[0.8, 1]$ according to real performance $pfmc_{real}$ in Figure 8.3a and according to the observed performance $pfmc$ in Figure 8.3b. Figure 8.3c represents the difference of real to observed performance; the confidence $confd$ values is shown in Figure 8.3d.



(a) real pfmc (b) obs pfmc (c) diff obs/real (d) cnfd

Figure 8.4: Generated worker population according to Scenario 2.

Scenario 2 contains 20 requesters which results in a much more "loaded" system. The workers are relatively skilled; their real performance $pfmc_{real}(w)$ is generated according to $\mathcal{N}(0.7,\, 0.25)$, i.e., the mean performance value is 0.7 compared to 0.3 as in the previous scenario. Here the assumption is that there is a higher amount of ratings already available and $cnfd(w)$ is generated according to $\mathcal{N}(0.8,\, 0.25)$. Performance values are again generated as $\mathcal{N}(pfmc_{real}(w),\, 1 - cnfd(w))$. The figures of Figure 8.4 illustrate the generated worker population for Scenario 2 in the same way as for the previous one.

In both scenarios tasks are issued in the first 500 rounds and the simulation ends when all workers have finished all accepted tasks.

Tasks are generated randomly with the following methodology. The real hidden result of a task is set to a uniformly distributed random value $\mathcal{U}(0, 1)$ from the range $[0, 1]$. An expected duration is randomly drawn from the set $\{1, 2, \ldots, 24\}$. A randomly assigned quality requirement $\mathcal{U}(0.4, 1)$ states whether the task poses high requirements to its processing. This means that requesters never state that the quality of their tasks' results is allowed to be lower than 0.4. Last but not least a random deadline is generated for which is guaranteed that it is after the expected duration.

Requester Behavior. In every round of the simulation each requester is asked to submit a task. After processing requesters receive the result for the task. If they receive the result after the deadline they rate the transaction with a value of zero and suspend for 20 rounds, i.e., they would refuse to issue a new task due to the negative experience. If the result is transmitted on time requesters rate the quality of the received result. Computationally this is done by comparing the task's real result with the received result. It is assumed that task requesters are able to estimate whether the received result is close to what was expected. The best possible rating is one, zero is the worst result, all values in between are possible. Ratings for a worker w, be it negative or positive, increase the confidence $cnfd(w)$ and update the observed performance $pfmc(w)$. If the rating is below a threshold of 0.3 the worker suspends for ten rounds, similar to a deadline violation. Hence, requesters with negative experiences tend to make less usage of the crowdsourcing marketplace. In addition to the pure task description requesters announce the maximum price they are willing to pay for the processing of the task. Prices are also represented by random values within the range $[0, 1]$. Tasks with high quality and high expected duration are more likely to have costs close to the maximum value.

Worker Behavior. When asked for a bid during an auction a worker first checks whether it is realistic to finish the task before the deadline considering all tasks the worker is working on. Each task has an expected duration t and each worker would only submit a bid if s/he has at least $1.5 \cdot t$ of time to work on the task before the end of the deadline considering already accepted tasks. The actual processing time is set to a random value within $[t, 2t]$. For workers with high real performance the processing time is more likely to be close to t. This value is set by the simulation environment and the workers do not know about the exact processing time in advance. The evaluation considers workers with two different bidding behaviors: conservative and aggressive. Conservative workers determine the price according to a linear combination of a tasks effort (i.e., a normalized value of the expected duration), the workers real performance, and her/his workload. The rationale is that workers want more money for work-intensive tasks; workers with a high real performance are aware of their capabilities which influence the price as well. Finally, the higher a worker's current workload the more payment is conceived.

$$bid(w, t)_{conservative} = 0.4 \cdot effort(t) + 0.4 \cdot pfmc_{real}(w) + 0.2 \cdot load(w)$$

Aggressive workers, in contrast to conservative workers, do not increase the bid's price based on the workload but are more strongly driven by their own real performance.

$$bid(w,t)_{aggressive} = 0.4 \cdot effort(t) + 0.6 \cdot pfmc_{real}(w)$$

Whether a worker is conservative or aggressive is chosen randomly with the same probability. The processing of a task has a positive influence on the real performance of the worker w, i.e, $pfmc(w)_{real,new} = pfmc_{real}(w) + 0.1 \cdot (1 - pfmc_{real}(w))$. This modeling of a training effect results in a high learning rate for workers with low real performance and a slowed down learning effect for workers who are already very good.

Auction Processing. Auctions are conducted for the purpose of matching a task to a worker. As described in Section 8.4 there is a qualification and preselection stage before the actual auction in order to avoid spamming a huge worker base with auction request for which many workers may not have the necessary skills. Here only one skill is considered and a with a limited number of 500 workers it is reasonable to admit all of them to the auctions. To achieve that the aptitude function, see Eq. 8.1, is set as follows:

$$aptitude : w \mapsto 1$$

After receiving the workers' bids they are ranked by a ranking function as defined in Eq. 8.2. Since there is only one skill the function is slightly adjusted:

$$rank : (w,b) \mapsto 0.6 \cdot pfmc(w) + 0.3 \cdot cnfd(w) + 0.1 \cdot (1 - price(b)).$$

Workers may either return a bid or refuse to submit a bid. From the received bids all values are removed whose price is higher than the price the requester is willing to pay. The remaining valid bids are ranked such that a high observed performance, high confidence, and a low price of the bid positively influence the rank. The emphasis at that stage clearly is on the performance and not on the price. It may happen that there is no valid bid; in that case the requester is informed that the task could not be processed.

Skill Evolution. In this chapter investigations on how crowdsourcing can benefit from skill evolution are conducted, which is achieved by assigning assessment tasks to workers. This is especially useful for workers with a low confidence value. For these workers only few or no ratings are available. An assessment task is a task that is assigned to a worker although another worker has won the auction and was assigned to the task as well. The workers are not aware of the fact that there are other workers processing the very same task; requesters are not either. The crowdsourcing provider is responsible for paying for the training tasks. As usual, the result of the highest ranked worker is returned to the requester but it is additionally used as a reference for the training task. This enables the marketplace to generate a rating for the assessed worker by comparing her/his result to the reference. A further positive effect is the training of the assessed worker. The assignment of training tasks is based on the received list of valid bids. For controlling the skill evolution Equation 8.3 needs to be set accordingly.

The following definition of the assessment function, which results in disabling skill evolution and leads to purely profit driven auction decisions, maps each combination of workers, bids, and reference workers to 0.

$$assess_{profit} : (w, b, w_r) \mapsto 0.$$

The following setting is used in the evaluation for the skill evolution enabled auctions.

$$assess_{skill} : (w, b, w_r) \mapsto \begin{cases} 0, & \text{if working queue not empty} \\ & \text{or } pfmc(w_r) < 0.8 \\ & \text{or } cnfd(w_r) < 0.8 \\ select(w, b), & \text{otherwise} \end{cases}$$

The function $assess_{skill}$ guarantees that only workers with empty working queue are assessed and that reference workers have high performance and high confidence. This is crucial because the worker w is rated according to the result of the reference worker w_r. If workers with a performance lower than 0.8 or confidence lower than 0.8 win an auction a training task assignment is prohibited. If all prerequisites are met the $select$ function determines the workers who are assigned a training task. It is possible that multiple training tasks are assigned.

$$select : (w, b) \mapsto \begin{cases} 1, & \text{with probability } (1 - cnfd(w)) \cdot urg \\ 0, & \text{otherwise} \end{cases}$$

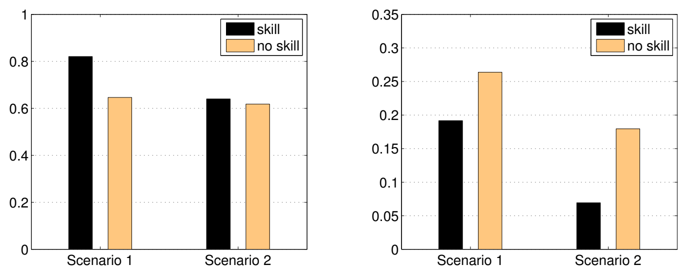
The $select$ function assigns a training task based on the confidence of the considered worker. A low confidence increases the likelihood for a training task. The constant urg can be used to finetune the training task assignment procedure. In the experiments it is set to a value of 0.01. A high value raises the probability of assessment tasks.

Discussions. Based on the introduced scenarios and simulation parameters, the benefit is estimated by comparing the skill evolution (*skill*) to regular auction processing (*no skill*). In the simulations, requesters issue a number of tasks to be processed by the crowd. However, requesters suspend their activity if the task quality is low (observed by low ratings) or task deadlines are violated. The hypothesis is that a higher quality of task processing, and thus received ratings, also has positive effects on the profit of workers and the crowdsourcing platform. Table 8.1 gives an overview of the task statistics in each scenario. The number of *issued* tasks is influenced by the requesters' satisfaction. *No bid* means that a task could not be assigned to any matching worker. The column *timeout* counts the number of tasks that were not delivered on time. Finally, the number of *training* tasks is depicted in the last column. All entries have the form $skill/no\ skill$.

Tasks	Issued	No bid	Timeout	Training
Scenario 1	527/315	2/5	57/73	757/-
Scenario 2	1687/1641	19/26	556/579	1310/-

Table 8.1: Tasks in Scenario 1 and Scenario 2.

In Figure 8.5, a comparison of the results of Scenario 1 and 2 on the basis of total rating scores given by requesters and the average difference between the evolved real performance of the workers and the observed performance is provided. For the *rating* (Figure 8.5a) results in Scenario 1, the difference is more significant than in Scenario 2; 19% difference compared to 2%. In Scenario 1, whilst with no skill evolution support only an average rating score of slightly



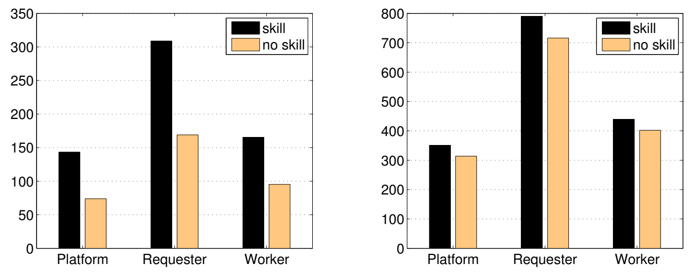
(a) Rating. (b) Misjudgement.

Figure 8.5: Rating and skill misjudgement.

above 60% is given by the requesters, the advantage of skill evolution support is most apparent. In this scenario with low load, the system can optimally exploit the better accuracy of worker skill estimation, resulting in an average rating of 80%. Interestingly, the ratings are far lower in Scenario 2, although the average true performance of the workers is higher. The reason is that the high number of requesters causes a heavily loaded system and increases the probability of deadline violations. Also, low performing workers are more likely to win an auction. The reaction of requesters to deadline violation and low quality is to give bad ratings, in this case an average rating of 60%. Due to the heavy load the benefit of skill evolution is small because for determining an auction's winning bid, performance and confidence become less decisive but free working capacity is more important.

For the *misjudgement* of the workers in Figure 8.5b (lower values are better), the results indicate clear benefits of skill evolution. Misjudgement is based on the average difference between the real and the observed performance. With more tasks, and in particular assessment-tasks being issued, worker capabilities can be estimated more correctly. For Scenario 1, the difference is below 20% with and below 30% without skill evolution support. For Scenario 2 with more load, the results considering misjudgement are evidently better. With more transactions, the average performance values' difference in the skill evolution support model is around 7% and around 19% otherwise. Thus, assessments provide remarkable good results for reducing misjudgements.

In the simulation requester try to minimize expenses; workers and crowdsourcing marketplace try to maximize earnings. Currently, a simple model is used in which the marketplace collects for each transaction the difference between the maximum expenses, as specified by the requester, and the minimum salary, as specified by the worker bid. In a real setting, the crowdsourcing platform could decide to charge less for its service. The quality of the results, and thus, their satisfaction directly influences the task offering tendency of the requesters. Again, with skill evolution applied, more tasks are processed since good ratings encourage requesters in offering tasks at a constant rate (see Figure 8.6a). Altogether, the requesters spend almost twice as much money with skill evolution. This is only true for Scenario 1 and similar to the previous results, the difference is far smaller considering overload situations. With more than six times as many requesters, their expenses remain way below the sixfold amount as spent in Scenario 1



(a) Scenario 1. (b) Scenario 2.

Figure 8.6: Payments from requesters to workers and crowdsourcing platform.

with skill evolution support. In total, the expenses in Scenario 2 are almost the same with and without skill evolution. The ratios are similar for the benefits of the workers and the platform. As a summary, skill evolution generally performs better, however, is not antagonistic to overload scenarios. While with moderate task offering frequencies the model performs much better in all measurements, the differences become even when load increases and assessment-task further overload the platform. The results show that it is the responsibility of the platform to balance the task load and trade only with a fair amount of requesters.

8.6 Conclusion

This chapter concludes the last presented non-intrusive adaptation strategy for collaborative networks basing partially on Mixed Systems and SOA infrastructures. The idea in this chapter was to focus on the individual crowd worker and how to support the individual in gaining or improving new skills to establish her/himself in the crowd. This is expected to benefit an approach of a platform with loyal returning workers. By observing this skill evolution and comparing the assignment history of an individual to the rating provided by the requesters, the worker's profile can be kept up-to-date and the platform gains confidence in their real skills. Knowing the skills of the crowd more precisely enables the platform also (i) to filter the most capable/interested workers for a received task and (ii) to choose the best fitting from the interested. Nevertheless, the taken approach does not dispense with the crowdsourcing principle of choosing the most convenient offer and in an auction the interested candidates are once again ranked, also according to their bid.

As the previous results, also the results in this chapter are of experimental nature with simulations only. Even if the results seem promising it is clear that future work must consider an alignment to a real crowd to confirm the results. Nevertheless, quality-oriented crowd management is challenging. This proves the fact that currently prominent platform providers (e.g., MTurk [5]) consider their crowd as a cloud of redundant workers. Their focus is on providing as many tasks and workers as possible without concerns on the quality of the result. However, as mentioned in the previous chapter there is already platforms [21, 19] atop of MTurk [5] that organize the crowd similarly to the concepts provided in this chapter.

CHAPTER 9

Variation and Configuration Management

9.1 Chapter Overview

Until now, the content of this work presented adaptation strategies for Mixed System environments. This last chapter considers configuration management for parts of these systems to prevent the necessity for adaptations and norm the ranges of the system. The problem is that for such large-scale systems the ranges differ usually from one environment variant to the other. This makes the management of variations a challenging problem based on the versatility of variants. This is most evident with software, but also, SOA and its notion of service composition is affected when enabling almost endless variants of services and their compositions, driven by different customer and market needs. The taken approach tries to identify the variation scope of existing configurations and derive a normative model of admissible configurations. Application examples for Mixed Systems are discussed.

Chapter Outline. Section 9.2 provides an introduction to the chapter. In Section 9.3 related approaches motivate the idea of variation management in SOA. Then in Section 9.4 the construction of a normative model is outlined. Section 9.5 outlines an implementation of a variation management with the example of variation management for network structures. Finally, Section 9.6 concludes the chapter.

9.2 Introduction

Today, companies recognize the complexity of variation managements for their products. Most spend major effort for a customer specific configuration management and a standardization of their manufacturing processes. The main goal is to find sound, quick, and also, economic adaptation or adoption methodologies for a required new product variant. One trend, for example, is to design product manufacturing processes that compose unified and variable product segments to anticipate all possible customer requirements. The manufacturing process can then be copied and deployed to different facilities worldwide (e.g, product lines in the automotive industry). The same concept is followed by service providers that offer their various service compositions as implementations of different business processes. Already, a multitude of different standards, helps them to uniformly provide services and manage their processes' and services' life cycles. Most of the standards relate to the functional and non-functional properties of a service. Functional properties include the description of the services capabilities, pre- and post-conditions, and expected inputs and outputs, e.g., the format. Non-functional properties include also properties of a service at higher levels of abstraction and cover the whole service life cycle. Categories include temporal and timing aspects (e.g. availability, responsiveness), monetary aspects (e.g, price, payment, discounts, and penalties), agreements (e.g., limits, obligations, and rights), quality, security, and trust [27]. Related standards for Web-services include WSDL, the semantic extension WSDL-S, WSLA, WS-Policy, UDDI, WSIL. The different properties result in many dimensions of flexibility and customization. Even if the original concept of SOA bases on highly flexible compositions and advocates interoperability some of the aforementioned properties contradict to this intention. For example, consider agreements or security concerns that inhibit or prohibit particular adaptations to compositions, i.e. service variants.

The variation management presented in this chapter follows two common objectives. The first is to ensure the manageability of the product's life cycle. An automated identification of the common components and points of variations support the management. The aggregated data can be explored to better understand the usability and benefits of product variants by drawing the experience from features of existing or decommissioned products. Currently developed and deployed product variants benefit from these classifications. The information derived from past variants supports maintenance decisions for the current. Furthermore, the knowledge about past and present variants supports product fitness for future variants. It allows to better estimate the viability of the product in regard to demand, market observations, and novel customer requirements. Also, it can guide the process of product innovation. The second objective is cost management. Companies want to determine the approximate price of a product beforehand and, more importantly, be able to anticipate the costs-effectiveness of a new product variant. Thus, it is essential to build up knowledge about past and present product variants, analyze future requirements and identify static cost factors, i.e. common components, and flexible cost factors, i.e. points of variation.

In detail, the contributions of this chapter are:

- *Version Management Classification:* A bottom-up/top-down approach classification is provided; variation and version management is explained.
- *Variation Management Model:* A novel model for variation management is presented including the notion of a normative model that leverages the knowledge from existing variants to provide guidelines for future product variant design.
- *Variation Management Prototype:* The prototype of a variation management system is outlined in the field of network infrastructure management. Architecture and implementation details are outlined. An application sample is explained.

The chapter is structured as follows: In Section 9.3 related chapter on variation management and SOA is outlined. Then in Section 9.4 a novel model of a normative model is presented. Section 9.5 outlines an implementation of a variation management with the example of variation management for network structures. Finally, Section 9.6 concludes the chapter.

9.3 Motivation and Background

Several approaches of variation management have been described by other authors. Here the most influential works are briefly listed and described. From the perspective of a provider in the work of [65] the authors describe variability modeling to support companies to manage the variability of their provided applications and their requirements. Variability is required to cover the requirements of as many tenants as possible. Their solution is to use explicit variability models to systematically derive customization and deployment information for the individual tenants. The main challenge described is to balance the offered variability with an economy of scale regarding the required resources. The definition and exploitation of explicit variability models is proposed that relate the two trade-offs. A different approach is considered by the authors of [103]. In this SAP research the variation modeling is passed from the provider to the tenant. The main idea is a model-driven approach for service customization by variability management. A meta-model describes the variable aspects, usage conditions, and constrains of the service model and the final service. The goal is to configure and integrate the service to properly fit the customer's requirements. The described 3-step engineering process for service customization includes the step of design and development by the provider. The next step integrates technical knowledge and business expertise by domain experts into a pre-configured service. In the final step of customization and personalization the service is adopted by the client. A quality model framework for business processes is provided in [66]. In this work quality evaluation is supported by variation management. Of particular interest is the provided feature model for business processes. A feature reflects a customer's requirements, in detail, an increment in product functionality that offers a configuration option. The model displays commonality and variability of features. The work in [34] provides a case study of the same process model with different execution styles. The variations are identified and integrated into a single configurable process model. The overview in [23] provides a similarity matching approach for business process models. The idea is to better maintain large business process repositories by categorize those according to their similarities. The proposed similarities for a set of models include label matching similarity including a syntactic, semantic, and contextual analysis of the labels describing the processes. Next there is structural similarity. As an example the graph-edit distance is mentioned. Finally, there is also behavioral similarity as a third metric which includes execution traces in the structural similarity to include considerations about the sequence of the models steps.

The approach taken in this chapter relates to all the related works concepts. However, in this chapter a more holistic concept is provided that integrates all the concepts into a model in a semi-automatic fashion. In particular, the concept of aggregating the properties of variants and generate the according normative model is extended.

9.4 Variation Management Model

This section provides the steps that lead to the creation of the novel variation management basing on a normative model. At the beginning the differences between variation and version management are outlined to clarify the meaning of both terms in the scope of this chapter. Then the *Top-Down – Bottom-Up* perspective on variations is explained by an example. The last section illustrates the creation cycle for a *Normative Model* and its integration in a variation management.

Variation vs. Version Management

In particular with a better manageable software development version releases and sophisticated versioning systems have become popular. Usually a new version relates to a change or improvement of a part of the product, e.g., during regular maintenance and update cycles. Improvements include removing bugs, testing new features, or creating prototypes. Important to note, versions of a product always relate to a common product base. In contrast, product variations usually specify characteristic properties of a product (e.g., particular configurations as required by different tenancies). Observations of the market, new requirements, and necessary adoptions can lead to the introduction of one or more variations. Also, former and current variations developed and adapted to cover all the requirements of a customers can be used to backtrack commonalities and differences, i.e. to derive common product components and points of variation. Furthermore, variations might not always derive from the same base. Those with the same base are usually variations of a product line.

From the viewpoint of a time line, a new version is a progress in time from an older version. In contrast, different variants of a product develop in parallel, also, with individual timing of their versions. However, prototypes of a version (e.g., branches and splits in software development) can also lead to new variations of a product. Thus, new versions can end up in a new variant of a product.

Top-Down and Bottom-Up Perspective

The approaches in the related work provide two distinct perspectives on variation management. Either the management is bound to a specific product domain, knows the points of variation, and imposes rules on the creation of new variants, or, within a given product domain, an assorted collection of product variants are aggregated according to predefined similarity metrics to pinpoint their commonalities and differences.

Domain example. Figure 9.1 illustrates the two perspectives approach by an example in the network domain. The *Router* at the center presents the point of variation. From a *Top-Down* perspective variation management is aware of the point of variation a-priori and the challenge is to decide for the ideal router variant from the collection by respecting the requirements. From the requirements usually a final set of possible candidates results as in the example comprised by the *Variations*. In the *Bottom-Up* approach the situation is different. In this case the variants *Variant1-3* are given and the challenge is to find their similarities and differences, respectively. Usually the field of application (i.e., the *Domain*) is also known to the analysis. In the example

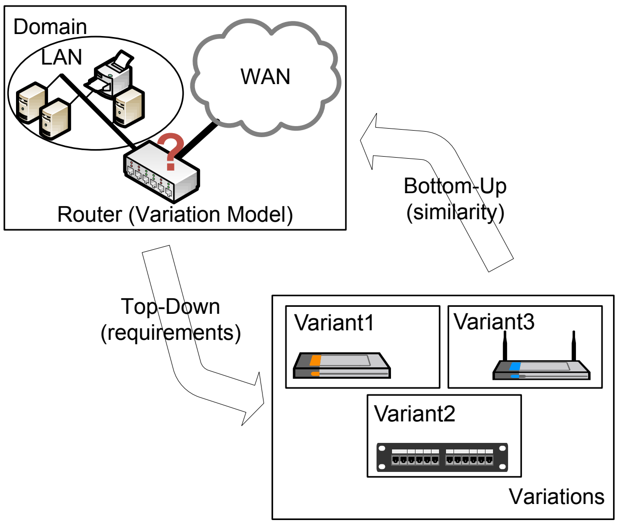


Figure 9.1: Top-Down and Bottom-Up Analysis of a router scenario.

the resulting information could include the router's different features, e.g. WLAN capabilities or port ranges and hints about the applicability to a certain routing scenario.

The investigations towards finding the scope of variants, and ultimately, to find some sort of norm for a given variant are explained in the next section.

Normative Model

Figure 9.2 represents a semi-automated *Bottom-Up* cycle approach that results in a *Normative Model*. It assists to constrain the product's scope by analyzing existing variants and together with the role of the *product manager* a norm for future variants can be defined.

Variation Model. In *step 1*, a pre-selected set of variants belonging to the same application domain is scanned by similarity metrics. For example, this step could include the product specific query of a database, or in the SOA case, the query of a service registry for service candidates for a particular business process. At the end of this step the *Variation Model* comprises the aggregated variant information. In an analysis of the structure the common and variable components can be identified. Thus, the *Variant Model* allows to distinguish between the common components, that are the same to each *Variant* and the variable components, points of variation, respectively. It also provides collective overview of the properties and their applied settings of the considered variants. For example, in the business process domain with similar composition the static and variable steps of the process can be identified. Moreover, a collection of the valid functional and non-functional properties of the involved services can be analyzed.

Normative Model. In a second step (*step 2*) the aggregated model's properties' settings are analyzed. By extracting the values of the settings, ranges can be aggregated. The result of this step is an automatically created scope of the considered set of variants an initial raw version of the *Normative Model*. Returning to the previous business process example, in this step the currently applied ranges for functional and non-functional properties are extracted which, further, allow to identify, e.g., viable service replacements for the variable steps in the process. After this fully automated steps a *product manager* takes charge of the *Variation Management*. Once s/he has analyzed the structure, and considered the currently applied settings and the particular in-house business rules, the generated ranges of the last step can be fine-tuned to provide the rules for future product variants. The role of such a *product manager* could be taken by a domain expert, or a sales person, etc. The last step (*step 3*) comes into play once a new product variant is created and released. With the *Normative Model* now in place, designers of new variants can rely on the norm and check the product variant against it. This improves the creation process in many aspects. Admissible configuration values derive from the experience with deployed and running systems. The cost range of a new product variant can be better estimated by considering the collected knowledge about possible or necessary settings and their intervals.

9.5 Crowdsourcing and Configuration Management

This section discusses some potential applications of the presented variation and configuration management in crowdsourcing. As outlined in the previous chapters crowdsourcing platform administration is a challenging task and must relay on interaction observations. The idea in the context of this chapter would be the establishment of standardized procedures and normed ranges for the management and decision processes. In the following a few related starting points are discussed.

Work Force Management. The aggregation of the information from task processing and experience from the skill evolution process outlined in the previous chapters could be used to derive best practices for different situations in work force management. The accurate knowledge

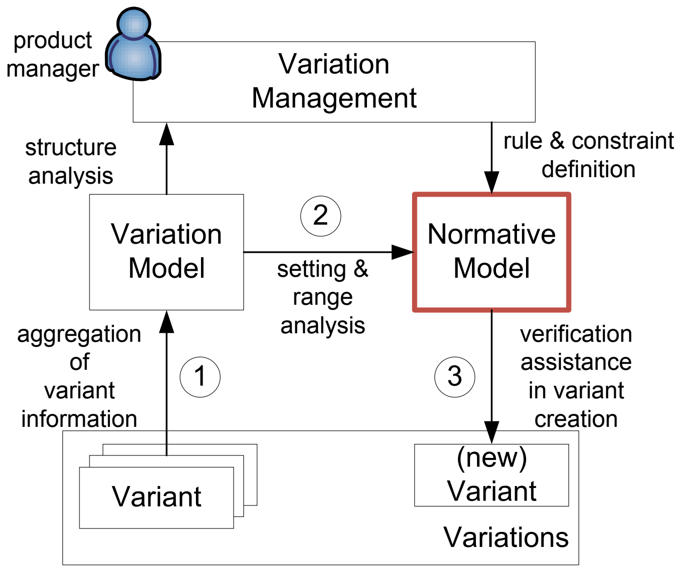


Figure 9.2: Normative Model.

about the currently available skills and in particular the current trend or future requirements lead to better organized broker searching, hiring, and formation routines. Regarding the broker and worker enlisting, boundaries could specify the upper and lower limits of required skill types in the crowd depending on different situations. Auctions could automatically be modified to include also plain training tasks in order to attract workers with the required skill type. Furthermore, and independently some different normed rules could define the learning task complexity for an individual worker depending on her/his education and experience.

Agreement Ranges. The application of agreements to brokers in crowdsourcing was detailed in Chapter 7. In this context standardization for platform management would mean that from past experience and variations of agreements a platform provider can better relate a broker's profile to the values of hard- and soft-constraints of objectives in agreements. The extracted ranges assist to better estimate offerable objective values and related costs for new customers. Also, based on the knowledge from this relation the discovery process can be extended to find brokers and potential workers that match particular objective constraints in an agreement.

9.6 Conclusion

In this chapter a general approach to variation and configuration management was sketched. The core idea of the bottom-up approach is to generate a model norm for the prevalent configurations that can be modified and adapted by the business management. In the last part a possible alignment with two examples from crowdsourcing are presented.

CHAPTER 10

Conclusion and Future Research

The investigations of this thesis focus on the area of *collaborative networks*. These platforms, while catching the interest of the Internet users and thus getting more and more crowded, gained a major presence in everyday's life. With a substantial boost by current advertising industry strategies, today the platforms' logos and emblems are commonplace in many of the TV-commercials and promotional poster (c.f., Facebook and Twitter).

With their loose coupling, openness, and voluntary-like participation these large-scale environments burden management with various challenges. The particular focus in this thesis is on collaborative networks with the goal of providing solutions to tasks on time and of high quality. The research volume of this thesis focuses on the unpredictable side-effects deriving from the openness which allows humans, among other, to join and leave unexpectedly and adopt an unpredictable working behavior. The core assumption is that this fact leads to misbehavior in collaborations. In particular, collaborative networks which accept outsourced steps of top-down planed, tightly scheduled process steps tend to fail, because they have no means cope with the *unpredictable working behavior* of humans.

The content of this thesis provides self-adaptive/self-organization management approaches for the loosely coupled human resources in this environment. As a prerequisite, this thesis considers an alignment between the concept of collaborative networks and the SOA model. This has already been studied in previous work, i.e. Human-provided Services and Mixed Systems, and their results and advantages regarding, e.g. interaction logging, were foundations and references for the investigations of this work. For their main parts, the contributions study *misbehavior related to HPS*, and propose appropriate adaptation strategies. In the view of this work, the required openness of the environments is due to the human participants. Misbehavior, e.g. failures, of SBS is out of scope because traditional services in these environments are considered either under control of the environment infrastructure provider or the HPSs. It is their responsibility to maintain those services to avoid them from forming potential additional factors causing humans to misbehave. Whereas the non-intrusive approaches presented, focuses on adaptation of human resources that generally are out of direct control and of unpredictable presence. Thus, appropriate strategies need to be different from the conventional applicable for, e.g., traditional services. Intuitively there is no way to "redesign" or "update" humans similar to conventional services, but rather, the solutions are to segregate disturbing sources, learn the characteristics of the human behavior and their real skills, offer humans means to develop their skills, and assist group formation.

This understanding leads to approaches that can be summarized by two distinct categories: In the *online approaches*, human interaction logs allow detecting degradation caused by delegation misbehavior. Adaptive actions try to transparently confine the misbehavior by regulating interaction channels and redirecting collaborations. In the *offline approaches*, the history of interaction data is considered. A novel query language allows extracting groups and their connectivity. Brokers can be identified and assigned with the responsibility to coordinate groups. Reviews of the behavior information let better estimate the real skill of the participants. This is the basis for an efficient task scheduling. Furthermore, training tasks and auctions aim at stimulation for contributions and loyalty to the platform. In return, these increase the confidence in the workers and the quality of their work.

For future extensions, intensive studies of the relations between the approaches, their results, and their applicability in Mixed Systems could initiate a proper configuration management with norms for collaborative networks. Moreover, a change from a simulated collaboration network to a real would confirm the results or provide valuable hints to correct the approaches. Furthermore, future extensions of the approaches must also consider SBSs as sources of misbehavior.

APPENDIX A

Curriculum Vitae – Harald Psaier

Vienna University of Technology
Distributed System Group
Argentinierstrasse 8
A-1040 Wien, Austria

Phone:	+43 (1) 588.01-584.18
Email:	hpsaier@infosys.tuwien.ac.at
Homepage:	http://www.infosys.tuwien.ac.at/staff/hpsaier
Born:	November 26, 1978.
	Italian Citizen (Südtirol).

Education

ongoing	PhD student in Computer Science, Vienna University of Technology.
2005	M.S. (Dipl. Ing.) Computer Science, Vienna University of Technology.

Employment

2009 - ongoing	Project assistant at Vienna University of Technology.
2011	Internship at IBM Research Almaden (3 month).
2006-2008	Software engineer for ERP/CRM software products.

Publications

Journal and Magazine Articles

- Dustdar S., Schall D., Skopik F., Juszczyk L., **Psaier H**.: *Socially Enhanced Services Computing, Modern Models and Algorithms for Distributed Systems*, Springer, 140 p., 2011
- Skopik F., Schall D., **Psaier H.**, Treiber M., Dustdar S.: *Towards Social Crowd Environments using Service-oriented Architectures*. it - Information Technology : Special Issue on Knowledge Processes and Services, 2011
- **Psaier H.**, Dustdar S.: *A survey on self-healing systems: approaches and systems*. Computing, Springer Wien, 2010.

Conference Papers

- Khazankin R., **Psaier H.**, Schall D., Dustdar S.: *QoS-based Task Scheduling in Crowdsourcing Environments*. 9th International Conference on Service Oriented Computing (ICSOC), Springer, 2011
- Satzger B., **Psaier H.**, Schall D., Dustdar S.: *Stimulating Skill Evolution in Market-Based Crowdsourcing*. 9th International Conference on Business Process Management (BPM), pp. 66–82, Springer, 2011
- **Psaier H.**, Skopik F., Schall D., Dustdar S.: *Resource and Agreement Management in Dynamic Crowdcomputing Environments*. 15th IEEE International EDOC Conference (EDOC), pp. 193–202, IEEE, 2011
- Schall D., Skopik F., **Psaier H.**, Dustdar S.: *Bridging Socially-Enhanced Virtual Communities*. 26th ACM Symposium On Applied Computing (SAC), pp. 792–799, ACM, 2011
- Skopik F., Schall D., **Psaier H.**, Dustdar S.: *Adaptive Provisioning of Human Expertise in Service-oriented Systems*, 26th ACM Symposium On Applied Computing (SAC), pp. 1568–1575, ACM, 2011
- Skopik F., Schall D., **Psaier H.**, Dustdar S.: *Social Formation and Interactions in Evolving Service-oriented Communities*, 8th European Conference on Web Services (ECOWS), pp. 27–34, IEEE, 2010
- **Psaier H.**, Skopik F., Schall D., Juszczyk L., Treiber M., Dustdar S.: *A Programming Model for Self-Adaptive Open Enterprise Systems*. 5th Workshop of the 11th International Middleware Conference (MW4SOC), pp. 27–32, ACM, 2010
- **Psaier H.**, Juszczyk L., Skopik F., Schall D., Dustdar S.: *Runtime Behavior Monitoring and Self-Adaptation in Service-Oriented Systems*. 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), pp.164–173, IEEE, 2010

- **Psaier H.**, Skopik F., Schall D., Dustdar S.: *Behavior Monitoring in Self-healing Service-oriented Systems*. 34th Annual IEEE Computer Software and Applications Conference (COMPSAC), pp. 357–366, IEEE, 2010
- Juszczyk L., **Psaier H.**, Manzoor A., Dustdar S.: *Adaptive Query Routing on Distributed Context - The COSINE Framework*. International Workshop on the Role of Services, Ontologies, and Context in Mobile Environments (ROSOC-M), pp. 588–593, IEEE, 2009

Bibliography

[1] Wil M. Aalst. Process-aware information systems: Lessons to be learned from process mining. *Transactions on Petri Nets and Other Models of Concurrency II*, pages 1–26, 2009.

[2] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne. Finding high-quality content in social media. In *WSDM '08*, pages 183–194. ACM, 2008.

[3] A. Agrawal et al. WS-BPEL Extension for People (BPEL4People), Version 1.0., 2007.

[4] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. Web services: Concepts, architectures and applications. *Springer Publishing Company, Incorporated*, 2010.

[5] Amazon Mechnical Turk. http://www.mturk.com, last access Jan 2012.

[6] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (ws-agreement). In *Global Grid Forum*, pages 1–47. The Global Grid Forum (GGF), 2004.

[7] D. Artz and Y. Gil. A survey of trust in computer science and the semantic web. *J. Web Sem.*, 5(2):58–71, 2007.

[8] L. Baresi, G. Guinea, and L. Pasquale. Self-healing bpel processes with dynamo and the jboss rule engine. In *ESSPE'07*, pages 11–20. ACM, 2007.

[9] A. Bertolino, G. De Angelis, L. Frantzen, and A. Polini. Model-based generation of testbeds for web services. In *TestCom/FATES*, volume 5047 of *Lecture Notes in Computer Science*, pages 266–282. Springer, 2008.

[10] D. Bianculli, W. Binder, and M.L. Drago. Automated performance assessment for service-oriented middleware. Technical Report 2009/07, Faculty of Informatics - University of Lugano, November 2009.

[11] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, Iii W. N. Mills, and Y. Diao. Able: A toolkit for building multiagent autonomic systems. *IBM Systems Journal*, 41(3):350–371, 2002.

[12] G. S. Blair, G. Coulson, L. Blair, H. Duran-Limon, P. Grace, R. Moreira, and N. Parlavantzas. Reflection, self-awareness and self-healing in openorb. In *WOSS'02*, pages 9–14. ACM, 2002.

[13] D.C. Brabham. Crowdsourcing as a model for problem solving: An introduction and cases. *Convergence*, 14(1):75, 2008.

[14] D. Brickley and L. Miller. Foaf vocabulary specification 0.98, 2010. http://xmlns.com/foaf/spec/.

[15] R. S. Burt. Structural holes and good ideas. *American Journal of Sociology*, 110(2):349–399, Sept. 2004.

[16] L.M. Camarinha-Matos and H. Afsarmanesh. Collaborative networks. In *PROLAMAT*, pages 26–40, 2006.

[17] C. Castellano, S. Fortunato, and V. Loreto. Statistical physics of social dynamics. *Reviews of Modern Physics*, 81(2):591–646, May 2009.

[18] S-W. Cheng, D. Garlan, B. R. Schmerl, J. P. Sousa, B. Spitnagel, and P. Steenkiste. Using architectural style as a basis for system self-repair. In *WICSA'02*, pages 45–59, 2002.

[19] Crowd Control. http://www.crowdcontrolsoftware.com/, last access Jan 2012.

[20] S. Corsava and V. Getov. Intelligent architecture for automatic resource allocation in computer clusters. In *IPDPS'03*, page 201.1, 2003.

[21] CrowdFlower. http://crowdflower.com/, last access Jan 2012.

[22] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. Towards architecture-based self-healing systems. In *WOSS'02*, pages 21–26, 2002.

[23] R. Dijkman, M. Dumas, B. Van Dongen, R. Käärik, and J. Mendling. Similarity of business process models: Metrics and evaluation. *IS*, 36(2):498–516, 2011.

[24] A.H. Doan, R. Ramakrishnan, and A.Y. Halevy. Mass collaboration systems on the World-Wide Web. *Comm. ACM*, 2010.

[25] M. Dontcheva, E. Gerber, and S. Lewis. Crowdsourcing and creativity. In *CHI'11*. ACM, 2011.

[26] J.J. Dujmovic and H.L. Larsen. Generalized conjunction/disjunction. *Int. J. Approx. Reasoning*, 46:423–446, December 2007.

[27] M. Dumas, J.J. O'Sullivan, M. Heravizadeh, D. Edmond, and A.H.M. ter Hofstede. Towards a semantic framework for service description. *IFIP*, pages 277–291, 2001.

[28] C. Dwyer, S.R. Hiltz, and K. Passerini. Trust and privacy concern within social networking sites. In *AMCIS*, 2007.

[29] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World.* Cambridge University Press, 2010.

[30] Facebook. http://www.facebook.com, last access Jan 2012.

[31] D. Ghosh, R. Sharman, H. Raghav Rao, and S. Upadhyaya. Self-healing systems - survey and synthesis. *Decis. Support Syst.*, 42(4):2164–2185, 2007.

[32] M. Glass, M. Lukasiewycz, F. Reimann, C. Haubelt, and J. Teich. Symbolic reliability analysis of self-healing networked embedded systems. In *SAFECOMP'08*, pages 139–152. Springer, 2008.

[33] J. Golbeck. Trust and nuanced profile similarity in online social networks. *ACM Trans. on the Web*, 3(4):1–33, 2009.

[34] F. Gottschalk, T. Wagemakers, M. Jansen-Vullers, W. van der Aalst, and M. La Rosa. Configurable process models: Experiences from a municipality case study. In *CAiSE*, pages 486–500. Springer, 2009.

[35] T. Grandison and M. Sloman. A survey of trust in internet applications. *IEEE Communications Surveys and Tutorials, 2000,*, 3(4), 2000.

[36] R.B. Halima, K. Drira, and M. Jmaiel. A QoS-Oriented Reconfigurable Middleware for Self-Healing Web Services. In *ICWS'08*, pages 104–111. IEEE, 2008.

[37] R.B. Halima, K. Guennoun, K. Drira, and M. Jmaiel. Non-intrusive QoS Monitoring and Analysis for Self-Healing Web Services. In *ICADIWT'08*, pages 549–554. IEEE, 2008.

[38] M. Hirth, T. Hoßfeld, and P. Tran-Gia. Anatomy of a crowdsourcing platform-using the example of microworkers.com. In *IMIS'11*, pages 322–329. IEEE, 2011.

[39] J. Howe. Crowdsourcing: A definition. `http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html`, last accessed Jan. 2012.

[40] M. C. Huebscher and J. A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Computing Survey*, 40(3):1–28, 2008.

[41] IBM. *An architectural blueprint for autonomic computing*. IBM White Paper, 2005.

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

[43] P. G. Ipeirotis. Demographics of mechanical turk. *Center for Digital Economy Research, NYU Stern School of Business, Working paper*, 2010.

[44] A. Iriberri and G. Leroy. A life-cycle perspective on online community success. *ACM Comput. Surv.*, 41:11:1–11:29, February 2009.

[45] L Juszczyk and S Dustdar. Script-based generation of dynamic testbeds for soa. In *ICWS'10*. IEEE, 2010.

[46] L. Juszczyk, H.-L. Truong, and S. Dustdar. Genesis - a framework for automatic generation and steering of testbeds of complexweb services. In *ICECCS'08*, pages 131–140, 2008.

[47] J. O. Kephart. Research challenges of autonomic computing. In *ICSE'05*, pages 15–22. IEEE, 2005.

[48] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[49] J.O. Kephart and W.E. Walsh. An artificial intelligence perspective on autonomic computing policies. *POLICY'04*, pages 3–12, June 2004.

[50] R. Khazankin, H. Psaier, D. Schall, and S. Dustdar. QoS-based Task Scheduling in Crowdsourcing Environments. In *ICSOC*, pages 297–311. Springer, 2011.

[51] J. Kleinberg. The convergence of social and technological networks. *Commun. ACM*, 51(11):66–72, 2008.

[52] J. Kleinberg, S. Suri, É. Tardos, and T. Wexler. Strategic network formation with structural holes. In *EC'08*, pages 284–293. ACM, 2008.

[53] P. Klemperer. *Auctions: Theory and Practice*. Princeton University Press, March 2004.

[54] R. Kumar, Y. Lifshits, and A. Tomkins. Evolution of two-sided markets. In *WSDM '10*, pages 311–320. ACM, 2010.

[55] T. Ledoux. Opencorba: A reflektive open broker. In *Reflection*, pages 197–214, 1999.

[56] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. End-to-end versioning support for web services. In *SCC'08*, volume 1, pages 59–66. IEEE, 2008.

[57] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. Monitoring, prediction and prevention of sla violations in composite services. In *ICWS'10*, pages 369–376. IEEE, 2010.

[58] F. Leymann. Workflow-based coordination and cooperation in a service world. In *CoopIS, DOA, GADA, and ODBASE*, pages 2–16, 2006.

[59] H. Ludwig, A. Keller, A. Dan, R.P. King, and R. Franck. Web service level agreement (wsla) language specification. *IBM Corporation*, pages 815–824, 2003.

[60] H. Ludwig, T. Nakata, O. Wäldrich, P. Wieder, and W. Ziegler. Reliable orchestration of resources using WS-Agreement. *HPCC*, pages 753–762, 2006.

[61] M. Amend et al. Web Services Human Task (WS-HumanTask), Version 1.0, 2007.

[62] P. J. Macdonald, E. Almaas, and A.-L. Barabási. Minimum spanning trees of weighted scale-free networks. *Europhys. Lett.*, 72(2):308–314, 2005.

[63] Y. Matsuo and H. Yamamoto. Community gravity: Measuring bidirectional effects by trust and rating on online social networks. In *WWW*, pages 751–760, 2009.

[64] M. J. Metzger. Privacy, trust, and disclosure: Exploring barriers to electronic commerce. *J. Computer-Mediated Communication*, 9(4), 2004.

[65] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl. Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In *ICSE/Pesos*, pages 18–25. IEEE, 2009.

[66] B. Mohabbati, D. Gaševic, M. Hatala, M. Asadi, E. Bagheri, and M. Boškovic. A quality aggregation model for service-oriented software product lines based on variability and composition patterns. In *ICSOC/ServiceWave*, pages 436–452. Springer, 2011.

[67] T. P. Moran, A. Cozzi, and S. P. Farrell. Unified activity management: Supporting people in e-business. *Com. of the ACM*, 48(12):67–70, 2005.

[68] O. Moser, F. Rosenberg, and S. Dustdar. VieDAME - flexible and robust BPEL processes through monitoring and adaptation. In *ICSE'08*, pages 917–918. ACM, 2008.

[69] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational model of trust and reputation for e-businesses. In *HICSS*, page 188, 2002.

[70] Horn P. *autonomic computing: IBM's Perspective on the State of Information Technology*. IBM, 2001.

[71] M.P. Papazoglou and D. Georgakopoulos. Service-oriented computing. *Communications of the ACM*, 46(10):25–28, 2003.

[72] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: a research roadmap. *IJCIS*, 17(02), 2008.

[73] C. Petrie. Plenty of room outside the firm. *IEEE Internet Computing*, 14, 2010.

[74] H. Psaier, L. Juszczyk, F. Skopik, D. Schall, and S. Dustdar. Runtime behavior monitoring and self-adaptation in service-oriented systems. In *Self-Adaptive and Self-Organizing Systems (SASO), 2010 4th IEEE International Conference on*, pages 164–173. IEEE, 2010.

[75] H. Psaier, H. Ludwig, L. Anderson, and B. Shaw. Identifying and managing variation scope in service management. Technical report, IBM Research – Almaden, San Jose, California and Distributed Systems Group, Vienna University of Technology, Jan 2012.

[76] H. Psaier, F. Skopik, D. Schall, and S. Dustdar. Behavior monitoring in self-healing service-oriented systems. In *Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual*, pages 357–366. IEEE, 2010.

[77] 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. IEEE, 2011.

[78] H. Psaier, F. Skopik, D. Schall, L. Juszczyk, M. Treiber, and S. Dustdar. A programming model for self-adaptive open enterprise systems. In *Proceedings of the 5th International Workshop on Middleware for Service Oriented Computing*, pages 27–32. ACM, 2010.

[79] M. Reid, C. Gray, and C. Honick. Online social networks, virtual communities, enterprises, and information professionals. *Searcher*, 16:28–40, 2008.

[80] H. Rheingold. *The Virtual Community*. Addison-Wesley, December 1993.

[81] R. Ronen and O. Shmueli. Soql: A language for querying and creating data in social networks. In *ICDE*, pages 1595–1602, 2009.

[82] F. Rosenberg, P. Celikovic, A. Michlmayr, P. Leitner, and S. Dustdar. An end-to-end approach for qos-aware service composition. In *EDOC'09*, pages 151–160. IEEE, 2009.

[83] J. Ross, L. Irani, M. Silberman, A. Zaldivar, and B. Tomlinson. Who are the crowdworkers?: shifting demographics in mechanical turk. In *CHI'10*, pages 2863–2872. ACM, 2010.

[84] M. Salehie and L. Tahvildari. Autonomic computing: emerging trends and open problems. *SIGSOFT*, 30(4):1–7, July 2005.

[85] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14, 2009.

[86] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[87] B. Satzger, H. Psaier, D. Schall, and S. Dustdar. Stimulating skill evolution in market-based crowdsourcing. In *9th International Conference on Business Process Management (BPM 2011)*, pages 66–82, 2011.

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

[89] D. Schall, C. Dorn, S. Dustdar, and I. Dadduzio. Viecar - enabling self-adaptive collaboration services. In *SEAA '08*, pages 285–292. IEEE, 2008.

[90] D. Schall, F. Skopik, H. Psaier, and S. Dustdar. Bridging socially-enhanced virtual communities. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 792–799. ACM, 2011.

[91] D. Schall, H-L. Truong, and S. Dustdar. Unifying Human and Software Services in Web-Scale Collaborations. *IEEE Internet*, 12(3):62–68, 2008.

[92] D. Schall, H.L. Truong, and S. Dustdar. The human-provided services framework. In *CEC/EEE*, pages 149–156. IEEE, 2008.

[93] N. Schuster, C. Zirpins, and S. Tai. A document-centric approach to open collaboration processes. *Current Trends in Web Engineering*, pages 538–544, 2010.

[94] M.W. Shapiro. Self-healing in modern operating systems. *ACM Queue*, 2(9):66–75, 2005.

[95] F. Skopik. Dynamic Trust in Mixed Service-oriented Systems - Models, Algorithms, and Applications. In *PhD thesis*. Vienna University of Technology, 2010.

[96] F. Skopik, D. Schall, and S. Dustdar. Modeling and mining of dynamic trust in complex service-oriented systems. *Information Systems*, 35(7):735–757, 11 2010.

[97] F. Skopik, D. Schall, and S. Dustdar. Trust-based adaptation in complex service-oriented systems. In *ICECCS*, pages 31–40. IEEE, 2010.

[98] F. Skopik, D. Schall, and S. Dustdar. Trustworthy interaction balancing in mixed service-oriented systems. In *SAC*, pages 799–806. ACM, 2010.

[99] F. Skopik, D. Schall, H. Psaier, and S. Dustdar. Adaptive provisioning of human expertise in service-oriented systems. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 1568–1575. ACM, 2011.

[100] M.A. Smith and P. Kollock. *Communities in cyberspace*. Psychology Press, 1999.

[101] L. Sproull and S. Kiesler. *CONNECTIONS New Ways of Working in the Networked Organization*. MIT Press, 1991.

[102] R. Sterritt. Autonomic computing. *Innovations in Systems and Software Engineering*, 1(1):79–88, April 2005.

[103] M. Stollberg and M. Muth. Service customization by variability modeling. In *ICSOC/ServiceWave*, pages 425–434. Springer, 2009.

[104] Q. Su, D. Pavlov, J.H. Chow, and W.C. Baker. Internet-scale collection of human-reviewed data. In *WWW'07*, pages 231–240. ACM, 2007.

[105] J. Surowiecki. *The wisdom of crowds: why the many are smarter than the few and how collective wisdom shapes business, economies, societies, and nations*. Doubleday, 2004.

[106] S. Tai, R. Khalaf, and T. Mikalsen. Composition of coordinated web services. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 294–310. Springer, 2004.

[107] A.S. Tanenbaum, J.N. Herder, and H. Bos. Can we make operating systems reliable and secure? *Computer*, 39(5):44–51, 2006.

[108] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White. A multi-agent systems approach to autonomic computing. In *AAMAS*, pages 464–471. ACM, 2004.

[109] The Linux Kernel Archives. http://www.kernel.org/, last access Jan 2012.

[110] M. Treiber, L. Juszczyk, D. Schall, and S. Dustdar. Programming Evolvable Web Services. In *ICSE*, pages 43–49. ACM, 2010.

[111] W. Tsai. Social capital, strategic relatedness, and the formation of intra-organizational strategic linkages. *Strategic Management Journal*, 21(9):925–939, 2000.

[112] twitter. http://twitter.com/, last access Jan 2012.

[113] uTest. http://www.utest.com/, May 2011.

[114] M. Vukovic. Crowdsourcing for Enterprises. In *Proceedings of the 2009 Congress on Services*, pages 686–692. IEEE Computer Society, 2009.

[115] W3C. Sparql query language for rdf, 2008. Online: http://www.w3.org/TR/rdf-sparql-query/.

[116] G. Wang, C. Wilson, X. Zhao, Y. Zhu, M. Mohanlal, H. Zheng, and B.Y. Zhao. Serf and turf: Crowdturfing for fun and profit. *Arxiv preprint*, 2011.

[117] Wikipedia. http://www.wikipedia.org/, last access Jan 2012.

[118] WS-Addressing, last accessed Jan, 2012. `http://www.w3.org/Submission/ws-addressing/`.

[119] WS-Policy. http://www.w3.org/TR/ws-policy/, last accessed Jan, 2012.

[120] Yahoo. Answers scoring system. http://answers.yahoo.com/info/scoring_system, last access March 2011.

[121] C.N. Ziegler and J. Golbeck. Investigating interactions of trust and interest similarity. *Dec. Sup. Syst.*, 43(2):460–475, 2007.