FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Discovery and Formation Models for Socio-computational Crowd Environments

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Sozial- und Wirtschaftswissenschaften

by

Florian Skopik

Registration Number 0325234

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.-Prof. Dr. Schahram Dustdar

The dissertation has been reviewed by:

(Prof. Schahram Dustdar) (Prof. Martin Gaedke)

Wien, 31.03.2013

(Florian Skopik)

Acknowledgements

First and foremost, I thank my adviser Prof. Schahram Dustdar for his continuous support over the past years and the unique chance to extend my prior computer science studies to new research areas and publish them as a further PhD thesis, focusing besides technical aspects also on social theories and novel business models. Additionally, I like to thank Prof. Martin Gaedke from the TU Chemnitz for his valuable comments to improve this thesis and for being my examiner.

Kamal Bhattacharya and his team offered me the opportunity to elaborate and apply approaches of this thesis at the IBM India Research Lab in Bangalore. It was an outstanding cultural experience to work with them in India and I'd like to thank each of them for welcoming and hosting me for several months. I further like to express my gratitude to Prof. Christos Nikolaou from the University of Crete who invited me to the Service-Oriented Computing (SOC) Summer School 2011 to chair a session as a tutorial lecturer, which enabled me to discuss and spread my ideas.

I thank all my colleagues from the Distributed Systems Group, especially[1], Christoph Dorn, Lukasz Juszczyk, Harald Psaier, Daniel Schall and Martin Treiber for successful collaborations centered around the Web Services Testbed 'Genesis 2' and various crowdsourcing applications. I thank the rest of the team for their support, and besides work, for having so much fun at our frequent social events. I am grateful for the constant financial support from the EU projects COIN (FP7-216256), SM4ALL (FP7-224332), COMPAS (FP7-215175), and S-CUBE (FP7-215483) funding the research discussed in this dissertation and enabling an unique amount of business trips and conference visits in the last years.

Finally, I owe special gratitude to my family for continuous and unconditional support: To my parents for all the opportunities they made available to me, and for the support they have given me along the way; to my fiancée Tina and our daughter Fabiana for their enduring patience and love.

Florian Skopik
31.03.2013
Vienna, Austria.

[1] names in alphabetical order

Abstract

Service-oriented computing is 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. The emergence of service-oriented systems has paved the way for a new computing paradigm that not only applies to software-based services (SBS) but also human actors. In such open and flexible enterprise environments, people contribute their capabilities as human-provided services (HPSs).

Crowdsourcing applications are typically open Internet-based platforms where problem-solving tasks are distributed among a group of humans. Crowdsourcing follows the ‘open world’ assumption allowing humans to provide their capabilities through the platform by registering themselves as members. While conventional companies count on easy manageable and well organized structures, crowdsourcing has a more loosely-coupled, dynamic, and flexible structure and depends in particular on the preferences and behavior of the individual members. These properties make service-oriented architectures (SOAs) the ideal approach to realize crowdsourcing applications.

With the increasing complexity of tasks, we argue that in future crowdsourcing environments, numerous HPSs and SBSs will need to be flexibly composed in order to cover a wide range of business requirements. These *socio-computational systems* pose additional challenges. With the human in the loop, traditional SOAs transform from pure technical systems into socio-technical systems. These systems are characterized by both technical and human/social aspects that are tightly bound and interconnected. The technical aspects are very similar to traditional SOAs, including facilities to deploy, register and discover services, as well as to support flexible interactions. Additionally, the social system includes people and their habitual attitudes, values, behavioral styles and relationships. In particular, considering drifting interests of people, evolving skills, and varying collaboration incentives requires enhanced technical infrastructures in terms of flexibility and adaptability.

This work deals with the foundational models and novel concepts to enable discovery and formation of actors, being HPS or SBS, in socio-computational crowd environments. In particular (i) *SOA-based Socio-computational Crowd Models* deal with foundational concepts to establish collaborative Web-based environments applying service-oriented approaches; (ii) *Discovery and Formation Models* focus on approaches to identify, connect and facilitate social compositions of crowd members.

Kurzfassung

Service-orientierted Computing ist ein aufstrebendes Paradigma um flexibel erweiterbare Systeme großen Maßstabs zu realisieren. Da Interaktionen und Kompositionen, welche mehrere Firmen überspannen, zunehmend alltäglich werden, beginnen organisatorische Grenzen zu verschwinden. Das Aufkommen von service-orienterten Systemen hat den Weg für ein komplett neues Gestaltungsparadigma geebnet, welches nicht nur bei Software-basierten Services (SBSs) Anwendung findet, sondern auch für menschliche Akteure geeignet ist. In offenen und flexiblen Umgebungen bieten diese ihre Fähigkeiten als sogenannte Human-Provided Services (HPSs) an.

Crowdsourcing Anwendungen sind meist offene Internet-basierte Plattformen wo Aufgaben zur Problemlösung unter den Mitgliedern einer Personengruppe verteilt werden. Crowdsourcing folgt der Annahme der "offenen Welt", d.h., dass Personen ihre Fähigkeiten und Arbeitskraft freiwillig über eine Plattform zur Verfügung stellen können, indem sie sich dort selbst als Mitglieder registrieren. Während konventionelle Firmen auf einfach zu verwaltende und gut organisierte Strukturen setzen, folgt Crowdsourcing eher lose gekoppelten, dynamischen und flexiblen Strukturen, und hängt vor allem von den Präferenzen und dem Verhalten der einzelnen Mitglieder ab. Diese Eigenschaften machen Service-orientierte Architekturen (SOAs) zum idealen Ansatz um Crowdsourcing Anwendungen zu realisieren.

Wir argumentieren, dass mit zunehmender Komplexität der Aufgaben in zukünftigen Crowdsourcing Umgebungen, eine Vielzahl von HPSs und SBSs flexibel kombiniert werden müssen um neuartige Anforderungen bestmöglich abdecken zu können. Diese *"Socio-computational Systems"* implizieren eine ganze Reihe neuartiger Herausforderungen. Mit dem Menschen im System wandeln sich traditionelle SOAs von puren technischen Systemen in sozio-technische Systeme. Diese Systeme sind sowohl durch technische als auch menschliche/soziale Aspekte charakterisiert. Technischen Aspekte sind ähnlich denen traditioneller SOAs, wie das Bereitstellen, Registrieren und Auffinden von Services und Unterstützung von dynamischen Interaktionen. Zusätzlich bringt aber das soziale System weitere Aspekte ein, z.B. Verhaltensmuster von Personen, sowie deren Werte, Bestrebungen und Beziehungen untereinander. Im Speziellen benötigen sich erweiternde Interessen, entwickelnde Fähigkeiten und ändernde Anreize zur Zusammenarbeit neue und verbesserte technische Strukturen in Bezug auf Flexibilität und Anpassbarkeit.

Die vorliegende Dissertation behandelt neuartige Konzepte um das Auffinden und Zusammenführen von Akteuren in Socio-Computational Crowd Umgebungen zu ermöglichen. Im Detail behandeln (i) *SOA-basierte Socio-Computational Crowd Modelle* die grundlegenden Konzepte um eine kollaborative Web-basierte Umgebung mittels Service-orientierter Ansätze zu realisieren, und (ii) *Discovery und Formation Modelle* fokussieren auf Ansätze um soziale Kompositionen von Crowd Mitgliedern zu identifizieren, herzustellen und zu unterstützen.

Publications

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

1. **Skopik F.**, Schall D., Dustdar S. (2012). *Discovering and Managing Social Compositions in Collaborative Enterprise Crowdsourcing Systems*. International Journal of Cooperative Information Systems (IJCIS), Volume 21, Issue 4, pp. 297-341. World Scientific.

2. **Skopik F.**, Schall D., Dustdar S. (2012). *Trusted Information Sharing using SOA-Based Social Overlay Networks*. International Journal of Computer Science and Applications (IJCSA), Volume 9, Issue 1, pp. 116-151. Technomathematics Research Foundation.

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

4. **Skopik F.**, Schall D., Dustdar S. (2011). *Opportunistic Information Flows Through Strategic Social Link Establishment*. IEEE/WIC/ACM International Conference on Web Intelligence (WI), August 22-27, 2011, Lyon, France. IEEE.

5. **Skopik F.**, Schall D., Psaier H., Treiber M., Dustdar S. (2011). *Towards Social Crowd Environments using Service-oriented Architectures*. it - Information Technology: Special Issue on Knowledge Processes and Services, Oldenbourg Wissenschaftsverlag, Volume 53, Issue 3, pp. 108-116.

6. **Skopik F.**, Schall D., Dustdar S. (2011). *Managing Social Overlay Networks in Semantic Open Enterprise Systems*. 1st International Conference on Web Intelligence, Mining and Semantics (WIMS), May 25-27, 2011, Sogndal, Norway. ACM.

7. **Skopik F.**, Schall D., Dustdar S. (2011). *Computational Social Network Management in Crowdsourcing Environments*. 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), April 27-29, 2011, Las Vegas, USA. IEEE.

8. 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.

9. **Skopik F.**, Schall D., Psaier H., Dustdar S. (2011). *Adaptive Provisioning of Human Expertise in Service-oriented Systems*. 26th ACM Symposium On Applied Computing (SAC), March 21-25, 2011, Taichung, Taiwan. ACM.

10. **Skopik F.**, Schall D., Psaier H., Dustdar S. (2010). *Social Formation and Interactions in Evolving Service-oriented Communities*. 8th European Conference on Web Services (ECOWS), December 1-3, 2010, Ayia Napa, Cyprus. IEEE.

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

12. Schall D., **Skopik F.** (2010). *Mining and Composition of Emergent Collectives in Mixed Service-Oriented Systems*. 12th IEEE Conference on Commerce and Enterprise Computing (CEC), November 10-12, 2010, Shanghai, China. IEEE.

13. 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.

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

AMT	Amazon Mechanical Turk
B4P	BPEL for People (*also:* BPEL4People)
BPEL	*see WS-BPEL*
BPMN	Business Process Modeling Notation
BQDL	Broker Query and Discovery Language
COIN	Collaboration and Interoperability of Networked Enterprises (EU project)
FOAF	Friend of a Friend
FP7	EU Seventh Framework Program
GNUPG	GNU Privacy Guard
HPS	Human Provided Services
NFP	Non-Functional Property
PKI	Public Key Infrastructure
PVC	Professional Virtual Community
QoS	Quality of Service
RDF	Resource Description Framework
REST	Representational State Transfer
RFS	Request for Support
SBS	Software-based Service
SLA	Service Level Agreement
SOA	Service-oriented Architecture
SOAP	*originally:* Simple Object Access Protocol
SPARQL	SPARQL Protocol And RDF Query Language
SRDA	SourceForge Research Data Archive
VO	Virtual Organization
WOT	Web of Trust
WS	Web Service
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Service Description Language
WSML	Web Services Modeling Language
WSMO	Web Services Modeling Ontology
WSMX	Web Services Modeling Execution Environment
XFN	XHTML Friends Network
XML	Extensible Markup Language
XSD	XML Schema Definition

CHAPTER 1

Introduction

The Web 2.0 paradigm has radically changed the way people collaborate. By utilizing social media, such as blogs, wikis and forums users spread their ideas, thoughts and knowledge around the world. Collaborations are no longer bound to closed company-internal legacy systems, but are spanning a wide variety of open and social Web platforms. This offers novel and unique opportunities, for instance, the efficient discovery of people with similar interests or collaboration partners with free resources. While people use such platforms for leisure activities since years, they are nowadays utilized also for serious business. For example, task-based platforms for human computation and crowdsourcing [Howe, 2008], enable access to the manpower of thousands of people on demand by creating human-tasks that are processed by the crowd. Human-tasks include activities such as designing, creating, and testing products, voting for best results, or organizing information. This paradigm is increasingly utilized by today's companies to outsource tasks to external experts when lacking particular expertise or time. By dividing work in separate pieces, activities can be distributed across crowd members. However, we argue that as a consequence, initially decomposed work eventually needs to be composed and integrated again to obtain the final result. This process requires a lot of coordination effort among crowd members, which adds a collaborative flavor to common crowd sourcing platforms, where members typically act only isolated.

Web services [Alonso et al., 2003] enable loosely-coupled cross-organizational collaborations. In particular, they provide the means to specify well-defined interfaces and let customers and collaboration partners use an organization's resources through dedicated access points. However, offered resources are not restricted to information and software-based services. Also *human expertise* can be provided in a service-oriented manner. For that purpose, the Human-Provided Services (HPS) Framework [Schall et al., 2008b] enables human participation in a SOA environment. A typical example is a document translation service that could be implemented in software, or in the same manner provided by humans by letting them receive and process requests through Web service interfaces. With the human in the loop, traditional service-oriented architectures (SOA) transform from pure technical systems into socio-technical systems [Cherns, 1976]. These systems are characterized by both technical and human/social aspects that are tightly bound and interconnected. The technical aspects are very similar to traditional SOAs, including facilities to deploy, register and discover services, as well as to support flexible interactions. Additionally, the social system includes people and their habitual attitudes, values, behavioral styles and relationships. In particular, considering drifting interests of people, evolving skills, and varying collaboration incentives requires enhanced technical infrastructures in terms of flexibility and adaptability. Due to the support of loose coupling, sophisticated discovery mechanisms, and dynamic binding, Web services and SOA deem to be the ideal technical framework to realize large-scale socio-technical systems.

In this thesis, we introduce an approach to a special form of a socio-technical system, based on the concept of human computation (e.g., see [Gentry et al., 2005]) in the context of crowdsourcing. While common crowdsourcing environments lack to a great extent collaborative aspects, the approach presented in this thesis, leverages social network principles to support the collaborative processing of crowdsourced tasks. On the one side, in this environment people can be discovered and composed like services, and interact in a service-oriented manner. On the other side, we account for social aspects when composing people, such as evolving relations, personal experiences, and interest similarities. So, the actors in this system process requests collaboratively, i.e., they compute solutions together and on behalf of others. We therefore call this kind of socio-technical system a *socio-computational crowd environment*.

1.1 Motivation for Socio-Computational Crowd Environments

Let us first study an example that demonstrates the actual need for socio-computational crowd environments. Today, large-scale enterprises are facing the challenge of effective management and exploitation of the employees' knowledge and resources. Usually, expertise in numerous fields is available but often this knowledge is neither discovered nor captured appropriately. Since decades, researchers invent models and approaches to overcome that issue [O'Leary, 1998] with sophisticated enterprise knowledge management systems which store and manage the employees' skills in centralized databases. Enterprise crowdsourcing [Vukovic, 2009] follows a different path. Here, employees are encouraged to participate in a *private crowd environment*, where they *actively* offer their skills and expertise to other departments of the company. On the one side, rare expertise can be discovered and, on the other side, free capacities in one department can be used to tackle peak loads in other departments by outsourcing especially non-critical activities. Thus, enterprise crowdsourcing deems to be an elegant new paradigm to harness people's capabilities in a flexible and far more effective manner compared to rather static traditional cross-department collaborations. Service-orientation is the ideal means to realize such private crowds (i.e., not open to the public), because members can offer their own services, can be dynamically discovered, are loosely coupled and thus composed at run-time, and flexibly assigned to activities.

Let us further assume a majority of the employees participate in the described socio-computational crowd environment. In contrast to the common notion of crowdsourcing, we do not use this environment to outsource tasks to single individuals only. We rather outsource compositions of problems, e.g., the creation of technical artifacts having interdependencies, to compositions of crowd members. Therefore, one major challenge is to identify reliable social compositions, i.e., groups of crowd members that have proven their reliable and successful collaboration behavior before. After the assignment of tasks, these members finally create, modify, and extend the required (or given) artifacts. This activity requires an extensive amount of interactions to coordinate work, align artifacts, and ensure a smooth integration of software modules later on. Today, a wide range of service-oriented communication, coordination, and collaboration tools are available for crowd members. Furthermore, since interactions are performed through these tools, they can be observed and even analyzed. Thus, valuable information about real collaboration behavior and spirit can be obtained and used to approve and update the social trust network between crowd members. This network is the basis for an effective future discovery of reliable crowd member compositions.

1.2 Research Questions and Contributions

This thesis provides answers to numerous questions concerning the implementation of efficient socio-computational crowd environments. Eventually, the most important aspects are, first, the integration of human actors together with all social aspects into a service-oriented environment; second, the linking of these actors to enable the collaborative dimension; and third, the realization with today's Web technologies. In context of these aspects, we formulate three highly relevant research questions which we answer in detail in this work:

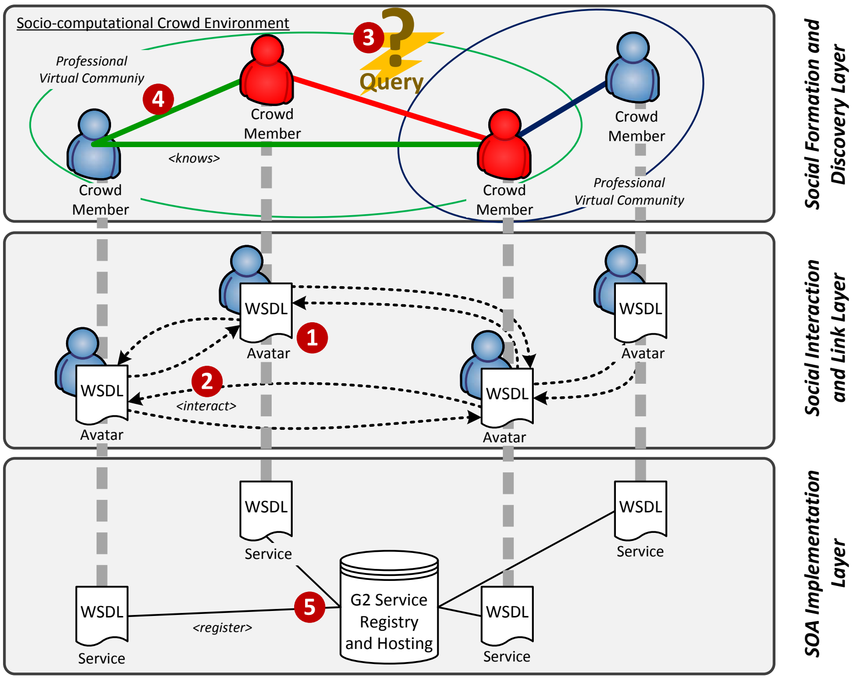
- **Research Question RQ-A:** What are applicable social theories to predict, explain, and support the formation of human groups on the Web who provide skills and expertise in a service-oriented manner?

- **Research Question RQ-B:** How can we facilitate community growth and emerging social ties, and how can suitable and approved compositions be discovered at run time, especially in highly dynamic environments?

- **Research Question RQ-C:** What SOA technologies and Web standards exist to implement supporting systems, and how does a prototype implementation look like?

Figure 1.1 depicts a rough overview about the approach to socio-computational crowds. Basically, we answer above research questions on three layers. From bottom to top, the SOA Implementation Layer deals with *RQ-C*, in particular the whole technical grounding, application of (Semantic) Web standards, performance and scalability etc. The Social Interaction and Link Layer covers *RQ-A*, i.e., the embedding of humans through services (so-called avatars), their interaction styles and methods, and the life cycle of social ties. On the top layer, Social Formation and Discovery Layer, we deal with answers to *RQ-B*, especially query and discovery models for social compositions and group formation models.

Effectively, all three layers must be put on top of each other to get a holistic view on the whole system. The key contributions are numbered from one to six, as given in the following list. In short, we introduce models to (1) establish hybrid services, i.e., avatars, consisting of human-provided and software-based components; (2) manage dynamically changing social relations accounting for temporal constraints; (3) define new query and discovery mechanisms for socio-computational crowds; (4) invent novel formation models based on these discovery mechanisms;



Socio-computational Crowd Environment
Professional Virtual Communiy
Professional Virtual Community
Query
Crowd Member
Crowd Member
Crowd Member
Crowd Member
<knows>
WSDL
Avatar
<interact>
Service
G2 Service Registry and Hosting
<register>
1
2
3
4
5
Social Formation and Discovery Layer
Social Interaction and Link Layer
SOA Implementation Layer

Figure 1.1: The approach to socio-computational crowds and major contributions of this thesis: (1) hybrid service-oriented avatars; (2) dynamic social relation model; (3) social query and discovery models; (4) social formation models; (5) SOA grounding.

and (5) demonstrate the realization of these models and mechanisms by applying and extending state-of-the-art Web technologies and standards.

Contribution 1: Hybrid Service-oriented Avatars

Techniques enabling human participation in a service-oriented Web exist since several years. Standards such as WS-HumanTask [Ford et al., 2007] and BPEL4People [Agrawal et al., 2007] as well as other approaches such as Human-Provided Services (HPS) [Schall et al., 2010] that fit in flexible collaboration environments, have been invented recently.

Contribution to the state of the art: In this work, we demonstrate the concept of SOA-based avatars that combine software-based parts and Human-Provided Services into one hybrid component. Software-based parts perform certain preprocessing of requests, including decisions about acceptance of work, queuing and prioritizing requests, and even respond to highly repetitive well-known requests; while a human cares for more complex (and usually more interesting) requests.

Contribution 2: Dynamic Social Relation Model

With the human user as an integral part of a SOA, also social relations between humans need to be modeled. While existing standards typically address static relations only, such as FOAF

[Brickley and Miller, 2010] or XFN [GMPG, 2011], and need to be defined manually, the concept of social trust [Skopik, 2010] has been introduced recently.

Contribution to the state of the art: Until now, we just tackled issues related to the initial establishment (for instance the definition of interaction metrics, their measurement, and rule-based trust interpretation; see thesis [Skopik, 2010]), but here, we further investigate the whole life cycle of trust relations. Besides these temporal constraints, another focus is on an efficient computational model to automatically capture and manage social trust networks.

Contribution 3: Social Query and Discovery Models

Since the social (trust) network is represented by sets of FOAF profiles using the Resource Description Framework (RDF), generic languages, such as SPARQL (SPARQL Protocol and RDF Query Language) [W3C, 2008] are suitable for querying this graph. With SPARQL one can find nodes that match certain properties, and discover even complex subgraph structures.

Contribution to the state of the art: Due to its generic nature the application of SPARQL is quite complex and error-prone. Therefore, here we propose and formulate a novel query language – the BQDL - Broker Query and Discovery Language – that is especially designed for social networks in order to efficiently and conveniently discover important members of socio-computational crowd environments; for instance, brokers who connect separated communities or member clusters. Effective discovery mechanisms allows people to find potential collaboration partners and are thus the basis for sophisticated formation models.

Contribution 4: Social Formation Models

Several generally applicable social theories exist that explain the behavior of individuals in group formation processes. For instance, reciprocity theory [Falk and Fischbacher, 2006] states that in open environments, people tend to establish a weighted symmetric relation of mutual give and take. Further examples are bounding based on interest similarity [Ziegler and Golbeck, 2007] and opportunistic positioning (cf. structural holes theory [Kleinberg et al., 2008]).

Contribution to the state of the art: Not all of these theories apply with the same degree in Web-based systems. On the one hand the openness and scale of the environment mostly prevents people to connect on a deeply personal level, on the other hand often anonymous and short-lived connections exhibit fundamentally different properties compared to traditional social relations. So, we identify suitable models that explain the emergence of social and collaborative networks, which are a foundational pillar for establishing socio-computational crowds.

Contribution 5: SOA Grounding

A wide variety of SOA standards, such as the whole WS-* stack [Alonso et al., 2003] are available, and eventually a perfect match to implement supporting systems for socio-computational crowd environments.

Contribution to the state of the art: We map all described concepts above to concrete technologies and SOA standards in order to evaluate and test our approaches. In contrast to our previous work (in particular thesis [Skopik, 2010]), here, we make one big leap forward by hosting avatars (and sole software-based services) in the WS Testbed environment G2 [Juszczyk and Dustdar, 2010]. G2 enables unique monitoring and adaptation opportunities at run-time in order to study group formation processes; not only to evaluate basic theories but also their applicability using novel WS-* technologies.

1.3 Dependencies and Relation to other Works

Research results presented in this thesis rely on joint effort from several persons. Some other PhD theses build the basis for our work and/or have been worked out in parallel. In particular, we make use of the following works:

- **Human-Provided Services (HPS):** The HPS framework is the means to enable a seamless integration of humans in service-oriented architectures. It provides an infrastructure for human interaction monitoring using appropriate SOAP interceptors and logging services. In context of *Contribution 1: Hybrid Service-oriented Avatars*, we discuss how HPSs can be applied and extended, however, we do not address fundamental HPS design decisions. This information can be found in the corresponding thesis of Daniel Schall [Schall, 2009].

- **Dynamic Social Trust:** The fundamental concepts of social trust have been addressed in a previous thesis [Skopik, 2010]. Now, we assume – based on these preceding results – we are able to infer a social trust network, and discuss in context of *Contribution 2: Dynamic Social Relation Model* its application to create socio-computational crowd environments.

- **Web Services Testbed G2:** We use the Web services Testbed Genesis2 (G2) for hosting technical components of service-oriented socio-computational crowd environments and demonstrate in context of *Contribution 5: SOA Grounding* its application for simulating and evaluating our novel concepts. However, the actual implementation of G2 is not in focus of this thesis, but is discussed in detail in the thesis of Lukasz Juszczyk [Juszczyk, 2011].

- **Run-Time Adaptations in Crowd-environments:** Numerous interaction-centric approaches for optimizing the run-time of crowd environments were investigated in the thesis of Harald Psaier [Psaier, 2012], for instance, interaction guidance and self-healing. We study these results in order to design the discovery and formation techniques of *Contribution 4: Social Formation Models*. However, we do not create models and algorithms to optimize interactions, but rather to facilitate efficient discovery and group formation before.

1.4 Organization of the Thesis

This thesis is structured[1] as follows. *Chapter 2 - Related Work* deals with the state of the art in relevant fields, including crowdsourcing, socio-technical systems, service-oriented computing

[1] This thesis is basically a compilation from a set of most relevant research papers published in course of PhD studies. The original papers, however, were modified to harmonize contents, establish a logical order, and avoid

and many more.

The main part of this work starts with *Chapter 3 - Social Formation and Interactions in Evolving SOA Communities* [Skopik et al., 2010f], which revisits previous research results (including, social trust and Human-Provided Services) that build the basis toward socio-computational crowd environments. Using these concepts we illustrate a naive discovery approach for crowd member compositions possible with state-of-the-art methods.

After the demonstration of current methods, we introduce the basic entities that are required for socio-computational crowd environments: dynamic social ties and avatars (cf. Figure 1.1). For that purpose, *Chapter 4 - Computational Social Network Management* [Skopik et al., 2011a] motivates the concept of collaboration in crowdsourcing and extend the previously discussed trust model with temporal aspects and sophisticated update mechanisms. Due to short-lived relations and short but repetitive tasks, an automatically managed social network model is mandatory to fit in large-scale and highly dynamic *socio-computational* crowd environments. Then *Chapter 5 - Adaptive Provisioning of Human Expertise on the Web* [Skopik et al., 2011d] introduces the concept of avatars that provide humans the ability to act in a service-oriented manner in socio-computational crowd environments. Avatars are an extension of HPSs with further software-based components that are able to pre-process requests and make simple decisions. We further demonstrate how such avatars are implemented, hosted and adapted at run-time using the agile SOA platform G2.

Then, having avatars that are thoroughly cross-linked, we show various mechanisms to facilitate group formation. For that purpose, *Chapter 6 - Bridging Socially-Enhanced Virtual Communities* [Schall et al., 2011] highlights a query mechanism to support the formation of expert groups in a crowd, and bridge desperate crowd segments with dedicated brokers. Furthermore, on top of crowd environments, we discuss mechanisms to establish a social semantic overlay network using an enriched FOAF concept in *Chapter 7 - Managing Social Overlay Networks in Semantically-enriched Crowds* [Skopik et al., 2011b]. This concept enables us to embed crowd networks in semantic enterprise environments using well-established SOA facilities to interact in a Web 2.0 manner. The efficient propagation of information is a major objective in social and collaborative networks. Thus, *Chapter 8 - Information Flows Through Strategic Social Link Establishment* [Skopik et al., 2011c] introduces a novel formation model that applies social theory concepts to predict people's linking behavior in open environments.

Finally, having presented all concepts to establish a socio-computational crowd environment, i.e., discovery, formation and interactions in social crowd networks, we highlight a new mechanism to identify and manage effective member compositions in *Chapter 9 - Discovering and Managing Member Compositions in Crowds* [Skopik et al., 2012a]. For that purpose, we describe a large-scale use case of socio-computational crowd environments and evaluate our work by studying a real world dataset from the SourceForge[2] community.

The thesis is concluded in *Chapter 10 - Conclusion and Future Research* summarizing our work and showing potential impact of research in the domain of socio-computational crowd environments.

redundant information. Regarding the actual contributions, each chapter contains the research results that have been previously published in a corresponding paper.

[2] SourceForge: `http://sourceforge.net`

CHAPTER 2

Related Work

This dissertation deals with discovery and formation models in socio-computational crowd environments. Since this topic is closely connected to many current research fields, we structure related work into the following sections:

- *Crowdsourcing* deals with the emerging concept of outsourcing single problem-solving tasks on the Web.
- *Socio-technical Dependencies in Software Development* describes the strong interdependencies between social structures and artifact compositions in this domain.
- *Service-oriented Computing* deals with the application of service-centric systems and architectures to build socio-computational systems.
- *Collaboration Monitoring and Social Networks* describes related work in the field of link measurement and prediction in social networks through monitoring and mining techniques.
- *Discovery and Formation Models* deals with novel models to discover collaboration partners and formation of groups on the Web using well-proven social theories.
- *Interplay of Social and Semantic Web* describes the technical application of Semantic Web standards in socio-computational crowd environments.

In the following, we discuss the basic principles and related approaches to above given research areas that are the foundational pillars of novel discovery and formation concepts.

2.1 Crowdsourcing

Crowdsourcing applications [Brabham, 2008; Howe, 2008; Vukovic, 2009] are online, distributed problem-solving and production models that have emerged in recent years. A vast number of registered individuals offer solutions to various problems and offer their workforce online. Crowdsourcing follows the ‘open world’ assumption allowing humans to provide their capabilities to the platform by registering themselves as members. Apart from this benefit of multiple, redundant workforce and collective intelligence crowdsourcing poses some difficult challenges related to its distributed and open nature. The main challenge remains how to organize and manage the crowd. In the first place, this includes the effort to capture capabilities of crowd members, required for sophisticated discovery mechanisms that find crowd members for given tasks based on matching skills. Then, there is the need to compose crowd members to address complex tasks that require numerous skills which a single member cannot provide. Both challenges, i.e., discovery and formation models, are addressed in this thesis. Two operating modes for crowd platforms have been identified [Vukovic, 2009]. In marketplace oriented models, crowds are organized by providers or brokers that bid for and distribute requests. In competition based models the request is an open call and the winning submission is picked. In this work we adopt the first type, which requires well interconnected crowd members in order to delegate and distribute tasks.

Crowdsourcing offers some distinct benefits such as multiple redundant workforces that can be utilized on demand [Kittur et al., 2008]; and collective intelligence used to rate items and vote for best results [Alonso et al., 2008]. However, many research challenges remain related to the distributed and open nature of crowdsourcing. In this work, we additionally study *private crowd* environments that are established by employees of large-scale enterprises [Stewart et al., 2009]. Thus, some assumptions can be made and typical issues relaxed, such as the motivation of crowd members to participate in activities and sufficient skills and experience of actors. Since members of enterprises have been hired by human resource offices, these are no issues in our discussed use cases.

2.2 Socio-technical Dependencies in Software Development

Developing complex software systems, requires the involvement of large groups of software designer, developer and tester, and produces an extensive amount of technical artifacts, including, code, specifications, manuals, and reports [Herbsleb et al., 2001]. As recognized by *Conway’s Law* [Conway, 1968], social structures reflect technical structures and vice versa. That means, there are strong similarities between the coupling of team members (social dependencies) and compositions of artifacts they produce (technical dependencies) [Souza et al., 2007; Trainer et al., 2005]. For instance, tighter coupled software modules require stronger coupled teams, since more technical dependencies demand for thorough coordination and alignment of work. Especially, when applying modern agile software development techniques with short incremental cycles the role of interpersonal interactions and social relations must be revisited. The impact of socio-technical dependencies has been considered to study new approaches on analyzing the fault-proneness of individual software components within a system [Bird et al., 2009] or to optimize structures of professional software development teams by learning from the organic formation of social structures in open source software development [Bird et al., 2008]. Furthermore, file repository logs are a valuable source that reflect technical dependencies between software modules *and* social dependencies such as co-authorship of code [D’Ambros et al., 2008; Souza et al., 2005]. Supporting explicitely software development with these principles has been studied by [Souza et al., 2007]. In this work, dependencies are visualized to support the manual discovery of single developers.

2.3 Service-oriented Computing

Service-Oriented Computing (SOC) promises a world of cooperating services loosely connected, creating dynamic business processes and agile applications that span organizations and platforms [Georgakopoulos and Papazoglou, 2008]. Service-oriented architectures (SOA) have emerged as the defacto standard to design and implement large-scale enterprise collaboration systems on the Web. They allow for loose coupling between single components and enable sophisticated discovery mechanisms based on functional (e.g., supported features) and non-functional (e.g., QoS) properties. Web service technology [Alonso et al., 2003] enables cross-organizational interactions in collaborative networks [Camarinha-Matos and Afsarmanesh, 2006]. Major software vendors have been working on standards addressing the lack of human interaction support in service-oriented systems. WS-HumanTask [Ford et al., 2007] and Bpel4People [Agrawal et al., 2007] were released to address the emergent need for human interactions in business processes. These standards specify languages to model human interactions, the lifecycle of human tasks, and generic role models. Role-based access models [Ford et al., 2007] are used to model responsibilities and potential task assignees in processes. While Bpel4People-based applications focus on top-down modeling of business processes, *service-oriented crowds* target flexible interactions and compositions of Human-Provided [Schall et al., 2010, 2008b]. This approach is aligned with the vision of the Web 2.0, where people can actively contribute services. In such networks, humans may participate and provide services in a uniform way by using the HPS framework [Schall et al., 2008b]. We call a system comprising software-based services (SBSs) and HPSs a Mixed Service-oriented System. An example of a mixed system where humans and software components are tightly coupled is a hybrid human-computer document translation system as discussed by [Shahaf and Horvitz, 2010], however, not focusing on the realization with SOA principles. A similar view is shared by [Petrie, 2010] who defines *emergent collectives* which are networks of interlinked valued nodes (services).

In our work, we combine SOA concepts and social principles. We consider *open service-oriented crowds* wherein services can be added at any point in time. Following the open world assumption, humans actively shape the availability of services. The concept of Human-Provided Services (HPS) [Schall et al., 2008b] supports flexible service-oriented collaborations across multiple organizations and domains. Similarly, emergent collectives as defined by [Petrie, 2010] are networks of interlinked valued nodes (services). Open service-oriented systems are specifically relevant for future *crowdsourcing applications* [Brabham, 2008]. While existing platforms (e.g., Amazon's Mechanical Turk[1]) only support simple interaction models (tasks are assigned to individuals), social network principles support more advanced techniques such as formation, delegation, and adaptive coordination.

[1] Amazon MTurk: `http://www.mturk.com`

2.4 Collaboration Monitoring and Social Networks

Enhanced flexibility of complex systems is introduced by establishing a cycle that feeds back environmental conditions to allow the system to adapt its behavior and learn from past events. The MAPE cycle [Dobson et al., 2006; IBM, 2005] is considered as one of the core mechanisms to achieve adaptability through self-* properties. Based on the observed context of the environment, different adaptation strategies can be applied [Di Nitto et al., 2008] to guide interactions between actors, the parameters of those strategies, and actions to prevent inefficient use of resources and disruptions. While autonomic computing allows for autonomous elements and applies these principles to distributed systems, current research efforts leave the human element outside the loop.

The availability of rich and plentiful data on human interactions in social networks has closed an important loop [Kleinberg, 2008], allowing one to model social phenomena and to use these models in the design of new computing applications such as crowdsourcing techniques [Brabham, 2008]. Semantic Web service communities as introduced by [Medjahed and Bouguettaya, 2005] foster the creation of structured social compositions with predefined community interfaces and functionality. However, ontology structures are not well suited for crowds, because crowd structures emerge bottom up and are difficult to capture with regard to functionality and interactions between crowd members. Also, value networks [Allee, 2000] are of interest when business aspects are investigated in crowd settings, i.e., the value that can be generated by such networks based on crowd capabilities and knowledge. Social network construction in the context of formalized business processes was discussed in [van der Aalst and Song, 2004]. However, in contrast to this work, we assume that activities rather emerge freely at run-time instead of being strictly preplanned. While others aim at detecting hidden social dependencies only from mining file repository logs [D'Ambros et al., 2008; Souza et al., 2005], we apply activity structures [Cozzi et al., 2006; Schall et al., 2008a] that act as a kind of container capturing performed actions on artifacts and interactions between actors. This concept enables the reliably correlation of social- and technical dependencies, since performed work is modeled explicitly (as usual in collaborative systems).

Social Trust [Artz and Gil, 2007; Jøsang et al., 2007; Mui et al., 2002; Sabater and Sierra, 2002] in service-oriented systems has become a very important research area. SOA-based infrastructures are typically distributed comprising a large number of available services and huge amounts of interaction logs. Therefore, trust in SOA has to be managed in an automatic manner [Malik and Bouguettaya, 2009]. Depending on the environment, trust may rely on the outcome of previous interactions [Mui et al., 2002] and interest similarity [Golbeck, 2009; Matsuo and Yamamoto, 2009]. Especially [Golbeck, 2009] proofed with data from real systems that trust between users emerges based on interest similarities. We adopt this finding to justify our approach of link establishment. Eventually, social trust is an indicator for the strength and degree of social coupling, which we utilize to discover matching social structures to given technical artifact compositions. In our approach, metrics express social behavior influenced by the context in which collaborations take place [Skopik et al., 2010a]. For instance, *reciprocity* [Falk and Fischbacher, 2006] is a concept describing that humans tend to establish a balance between provided support and obtained benefit from collaboration partners.

Aging models for the WWW [Brewington and Cybenko, 2000] describe common characteristic change rates of Web pages. Similar principles have been applied in social network analysis to update user profiles. For example, sliding window filters have been studied [Kossinets and Watts, 2009] to construct network approximations from interactions. Interaction behavior, interest similarities and joint group memberships, social relations, such as trust, can be predicted to some extent automatically [Matsuo and Yamamoto, 2009; Ziegler and Golbeck, 2007]. Work by [Hang and Singh, 2010; Liben-Nowell and Kleinberg, 2003] discusses link prediction based on network similarity, focusing on structural graph properties such as number of neighbors and number of in/out links. However, in our model, these links reflect social trust relations. Thus, we study the emergence of social relations from a multitude of social interaction metrics and behavioral styles.

2.5 Discovery and Formation Models

Query mechanisms are important to discover crowd members with certain properties. In [Ronen and Shmueli, 2009], a query language for social networks was presented, which has some similarities with our BQDL (described in this thesis), however, without supporting the discovery of complex sub communities based on metrics and interaction mining techniques. A more general query language is SPARQL [W3C, 2008], which has been designed to query ontological data.

In context of strategic formation in social networks and communities [Tsai, 2000], the theory of structural holes was developed by Burt [Burt, 2004] and 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 [Kleinberg et al., 2008]. Our approach is based on interaction *mining and metrics* to dynamically discover brokers suitable for connecting communities in service-oriented collaborations.

Game-theoretic models [Osborne and Rubinstein, 1999] allow to explain the behavior of single actors. In coalitional games, people attempt to find collaboration partners to increase their benefits gained from the network. Depending on the environment, social relations are established based on the outcome of previous interactions [Skopik et al., 2010a] and interest similarity [Golbeck, 2009]. In our approach, various metrics express social behavior influenced by the context in which collaborations take place [Skopik et al., 2010a]. For instance, *reciprocity* [Falk and Fischbacher, 2006] is a concept describing that humans tend to establish a balance between provided support and obtained benefit from collaboration partners. A further important concept in group formation is the *structural holes theory* [Burt, 1992]. This theory states that actors actively attempt to position themselves in beneficial positions within a community network. Many works have considered implications of this theory, for instance [Goyala and Vega-Redondo, 2007] in economics, and also mapped to Web environments [Kleinberg et al., 2008].

2.6 Interplay of Social and Semantic Web

FOAF [Brickley and Miller, 2010] is a standard format to describe an actor's profile, including, name, contact details, interests and organizational involvements, and furthermore, allows to link actors through ‘knows’-relations. Thus FOAF allows to build up whole social networks. Basically, FOAF is described with the Resource Description Framework (RDF) file format, which enables the convenient aggregation of multiple FOAF files and reasoning on sets of profiles using Semantic Web technologies. As such, FOAF is well suited for describing members of socio-computational crowds as it provides all necessary aspects for managing collaboration links.

In contrast to many common top-down approaches that model user profiles at least partly by the means of ontologies [Middleton et al., 2001, 2004], we create interest profiles fully dynamically through mining tagged interactions. Tagging and its meaning has been studied by [Golder and Huberman, 2006]. While others create tagging profiles with hierarchical clustering models, we apply a more lightweight approach using various analytical models from the domain of information retrieval, including term-frequency and inverse document frequency metrics [Salton and Buckley, 1988]. Thus, we gather relevant data through mining (bottom up) and manage these data by defining personal profiles (top-down).

Social networks become more and more interlinked with enterprises and collaborative platforms [Breslin et al., 2009]. Semantically-enriched service platforms following the SOA paradigm such as WSMX [Haller et al., 2005] provide the means to discover and compose services in cross-organizational environments based on standardized languages (see WSMO [Lara et al., 2004]). These platforms not only enable interactions between technical services across boundaries, but also human interactions on top of these services. The convergence of social interactions in flexible service-oriented environments makes it essential to extend well-established data formats for describing the structure of social networks such as FOAF with access control techniques.

A large amount of information is exchanged online using social networking platforms. It becomes thus essential to adapt and influence the information exchange in an automated manner [Skopik et al., 2010c]. Selective dissemination of information (SDI) [Altinel and Franklin, 2000; Diao et al., 2004] is used to filter restricted data by considering user profiles. The mechanisms for signing RDF graphs have been presented in [Giereth, 2005]. The combination of FOAF and SSL [Story et al., 2011] enables secure access to FOAF profiles. The embedding of access control mechanisms in FOAF has been illustrated in [Hollenbach et al., 2005; Kruk et al., 2006].

CHAPTER 3

Social Formation and Interactions in Evolving Service Communities

The global scale and distribution of companies have changed the economy and dynamics of businesses. Web-based collaborations and cross-organizational processes typically require dynamic and context-based interactions between people and services. However, finding the right partner to work on joint tasks or to solve emerging problems in such scenarios is challenging due to scale (number of involved people and services) and the temporary nature of collaborations. Furthermore, actor skills and competencies evolve over time requiring dynamic approaches for the management of actor properties. Web services and SOA are the ideal technical framework to automate interactions spanning people and services. In this chapter, we recapitulate a novel discovery mechanism based on social trust to support formation and dynamic interactions in service-oriented collaboration networks. We argue that trust between members is essential for successful collaborations. Here we discuss profile similarity-based link establishment to connect disparate network segments.

3.1 Introduction

Small and medium-sized organizations create alliances to compete with global players, to cope with the dynamics of economy and business, and to harvest business opportunities that a single partner cannot take. In such networks where companies, communities, and individuals form virtual organizations, collaboration support is a major research track. In this chapter, we focus on using SOA to support the creation and operation of professional virtual communities (PVCs). This kind of communities – also referred to as a special case of a virtual organization – is created by individuals to facilitate the collaboration of professionals. For instance, the members of PVCs may work on new technology standards, discuss current research problems, or offer support to the economy.

Individuals and companies that are interested in collaborations register at dedicated portals, where they can flexibly discover partners to form temporal alliances [Camarinha-Matos and Afsarmanesh, 2006]. The collaborations in such networks usually span numerous individuals distributed over various organizations and locations. Due to the scale of these networks it is impossible for the individuals to keep track of the dynamics in such networks. However, the recent adoption of service-oriented concepts permits the (semi-)automatic management of member profiles and network structures. In particular, SOA provides the functional means to allow loose coupling of entities through predefined interfaces and well-formed interaction messages. Upon SOA, monitoring of interactions enables the inference of social relations and expertise/interest profiles through mining logs. Hence, we use SOA to support and guide human interactions in collaborations by utilizing social relations. The automatic inference and adaptation of relations between network members [Mui et al., 2002; Skopik et al., 2010a] has several advantages. Negative influences, such as using outdated information for partner discovery, do not exist compared to manually declared relations. Moreover, monitoring of interaction behavior allows timely adaptations in ongoing collaborations, for instance, updates of member profiles based on successes in recent collaborations and collected experiences, without major user intervention.

This chapter deals with the following contributions:

- *Social Composition Model.* We introduce concepts to enable the seamless integration of human capabilities in SOA, and the concept of social trust to support the discovery of human-provided services and their interactions.

- *Trust-based Link Establishment in Collaborative SOA.* We study the application of introduced concepts to support group formations in state-of-the-art SOA with the human user in the loop.

- *Prototype and Evaluation.* We discuss the Social SOA formation tool – a prototype implementation on top of well adopted standards, including WSDL, SOAP and FOAF (Friend-Of-A-Friend) [Brickley and Miller, 2010], and evaluate its applicability with a SOA testbed.

3.2 Socially Enhanced SOA Systems

While the traditional SOA concepts deem to be sufficient from a pure technological point of view, the situation changes with the human user in the loop. Considering service-oriented collaboration scenarios on the Web, here we discuss various views on socially-enhanced SOA. In Figure 3.1, three main building blocks are identified that are based on traditional SOA concepts (services, discovery, and interactions). We argue that the role of humans in SOA should be extended so that people are able to shape the availability of services. Furthermore, not only software-based services are part of such systems, but also services provided by human actors.

Human-Provided Services (HPS). HPS act as interaction interfaces toward humans [Schall et al., 2008b], letting users define various HPSs for different collaborative activities indicating their ability (and willingness) to participate in ad-hoc as well as process-centric collaborations. Users can manage interactions, which might span various platforms and services. The very idea

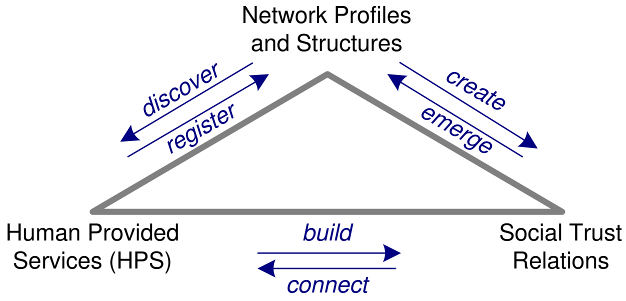


Figure 3.1: Social compositions in SOA.

of HPS is to support humans in offering their skills and capabilities as services (e.g., a 'document review service' provided by one or more human actors). For example, human activities can be defined by the end-user and are mapped onto Web service interfaces.

Social Trust. In this chapter, we focus on supporting formations and interactions in service-oriented collaboration environments by accounting for the individuals' social relations, especially *social trust*. In contrast to a common security perspective on trust, the notion of social trust refers to the interpretation of previous collaboration behavior [Mui et al., 2002; Skopik et al., 2010a] and the similarity of dynamically adapting interests [Golbeck, 2009; Skopik et al., 2010a]. Especially in collaborative environments, where users are exposed to higher risks than in common social network scenarios, and where business is at stake, considering social trust is essential to effectively guide human interactions.

Trust-based Network Profiles. We argue that in a socially enhanced SOA, network profiles replace traditional service registries. Queries for services of collaboration partners are not only based on sole service capabilities and QoS, but increasingly personal relations are of paramount importance. Especially in social environments, provided services of personally known partners are highly favored compared to unknown third party services. Thus, we adopt common standards of the social network domain to reflect personal relations and partner properties; in particular FOAF [Brickley and Miller, 2010]. However, we employ a system that dynamically creates and adapts FOAF structures upon inferred trust relations; thus keeping track of the dynamics in collaboration networks automatically. Network Profiles support the (i) discovery of potential collaboration partners (direct relations *and* recommendations of yet unknown actors through well known actors); (ii) routing of requests and messages in the network; (iii) creation of human-service compositions and (interest) group formation within larger communities.

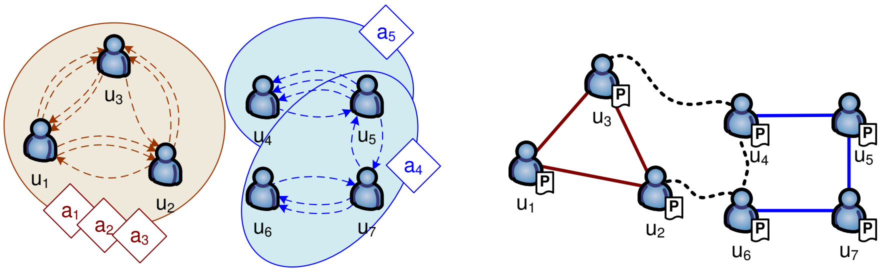
3.3 Emerging Relations in Professional Virtual Communities

We depict a professional virtual community (PVC) environment to familiarize with our concepts, and to demonstrate the dynamic emergence of social relations. A PVC is a virtual community that consists of experts belonging to different physical organizations, and who interact and collaborate by the means of information and communication technologies to perform their work. Nowadays, service-oriented technologies are increasingly used to realize PVCs. The support of loose coupling, convenient discovery, dynamic binding and composition mechanisms makes SOA the ideal grounding for Web-enabled PVCs.

Figure 3.2(a) depicts various member groups that collaborate in context of five different activities. The color of the activity context determines the expertise area an activity is related to. Such activities are, for instance, the specification of new technology standards or scientific dissemination. Activities are a concept to structure information in ad-hoc collaboration environments, including the goal of the ongoing tasks, involved actors, and utilized resources. They are either assigned from outside the community, e.g. belonging to a higher-level process, or emerge by identifying collaboration opportunities. In order to achieve their goals, the members of the PVC interact in context of the currently performed activities. In this chapter, we focus on a special type of interaction: *requests for support (RFSs)* [Skopik et al., 2010d]. PVC members interact using SOA technology. In our scenario we make use of the HPS framework to allow human participation in a service oriented manner, i.e., humans can provide their capabilities as services, and enable human interactions through SOAP. All SOAP messages are logged for later analysis.

Social relations, e.g., reflected in FOAF profiles [Brickley and Miller, 2010], emerge from interactions (Figure 3.2(b)), and are bound to particular scopes (here: expertise areas). As shown later in this work, we model the interaction context with tags and keywords in order to create communities with actors in similar activities. Through analyzing interaction contexts (i.e., tags from exchanged messages that are collected in activities), we determine a community's predominant activity focus and single members' centers of interests. Frequently used keywords are stored in the actors' profiles (see symbol P) and later used to determine interest and expertise similarities. In the given scenario, this similarity measurement is used to support the emergence of trust between PVC members regarding help and support in different expertise areas. We manage trust relations in a directed graph model $G = (N, E)$, where nodes N represent the network members and directed edges E reflect trust relations annotated by their scope. While some kind of social relations already exist in a community, e.g., expressed through FOAF profiles, supporting the emergence of new relations becomes a paramount undertaking to form larger expert group and support the continuous growth of communities.

Trust-based Link Establishment. Consider a scenario in the given PVC in Figure 3.2(b)



(a) Member interactions.

(b) Emerging relations.

Figure 3.2: Collaboration model for service-oriented PVCs: (a) interactions in context of activities; (b) emergence of profiles and relations based on previous interactions.

where collaboration between one community (u_1, u_2, u_3) and another one (u_4, u_5, u_6, u_7) should be facilitated. In that case, actors from both communities should be 'connected', i.e., introduced to each other. However, not just random actors should be picked, but actors having similar interests and therefore, a common basis for future interactions (see dashed lines). We argue that establishing personal contacts in socially oriented environments is of high importance compared to the traditional SOA domain, where services are mostly composed based on their sole properties (e.g., features and QoS) only.

Let us assume we are able to infer meaningful social relations between interacting network members (as detailed in [Skopik et al., 2010a] and partly shown later in this chapter). These relations have major impact on future collaborations in different manners:

- *Supporting the Formation of Expert Groups.* Successful previous compositions of actors should not be dissolved but actively facilitated for future collaborations. Thus, tight trust relations are dynamically converted to FOAF relations.

- *Controlling Interactions and Delegations.* Interactions and delegations of tasks between members can be guided upon FOAF relations. We argue that people tend to favor well-known members over any third parties.

- *Establishing new Social Relations.* The emergence of new personal relations is actively facilitated by establishing links. Connecting actors with similar interests (see dashed edges in Figure 3.2(b)) supports the emergence of future trustworthy compositions.

3.4 Human Interactions in Service Communities

Community members interact to reach a predefined goal. For instance, they request support, exchange information, delegate tasks, and coordinate actions to perform certain activities. Therefore, interactions always take place within certain contexts. Traditional service-oriented architectures focus on modeling and implementing interactions between distributed software-based services using Web services technology. A central part of SOA are standards such as descriptive service interfaces (WSDL) and the exchange of XML-based messages following a standardized format (SOAP). These mechanisms enable the dynamic discovery and invocation of services.

Also human interactions may rely on these SOA principles as discussed in the following. This fact enables the adoption of various available monitoring and logging tools in service-oriented collaboration systems. The XML-based structure of SOAP messages is well-suited for message header extensions, such as addressing and routing information, and annotation with contextual elements (e.g., activity identifier).

Human-Provided Services

As an example, an excerpt of a generic request for support (RFS) schema definition is shown in Listing 3.1. A user may send such a message (instance of the schema) to a HPS in case s/he needs assistance in ongoing collaborations. For that purpose the user defines the `Request`, including a `subject` and the detailed problem (`requ`), links to important `resources`, and `keywords` to categorize the message (such as the expertise area).

```
<xsd:schema tns="http://myhps.org/rfs">
 <xsd:complexType name="GenericResource">
  <xsd:sequence>
   <xsd:element name="Location" type="xsd:anyURI"/>
   <xsd:element name="Expires" type="xsd:dateTime"/>
  <xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="Request">
  <xsd:sequence>
   <xsd:element name="subject" type="xsd:string"/>
   <xsd:element name="requ" type="xsd:string"/>
   <xsd:element name="resource" type="GenericResource"/>
   <xsd:element name="keywords" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:element name="SupportRequest" type="Request"/>
 <xsd:element name="AckSupportRequ" type="xsd:string"/>
 <xsd:element name="GetSupportReply" type="xsd:string"/>
 <!-- reply details omitted -->
 <xsd:element name="SupportReply" type="Reply"/>
</xsd:schema>
```

Listing 3.1: RFS schema definition.

The `GenericResource` defines common attributes and metadata associated with resources such as documents or policies. A `GenericResource` can encapsulate remote resources that are hosted by a collaboration infrastructure (e.g., document management). An *interaction policy* is a special type of resource and plays an important role for controlling interaction flows, e.g., time constraints, delegation behavior including decisions whether to respond to a requester directly or to a 'social broker', and so on. `Request` defines the structure of an RFS (here we show a simplified example). A `Reply` is the corresponding RFS response (we omitted the actual XML defintion).

Listing 3.2 shows the binding of the HPS WSDL to the (HPS) infrastructure. The protocol (at the technical HPS middleware level) is asynchronous allowing RFSs to be stored, retrieved, and processed. For that purpose we implemented a middleware service (HPS Access Layer - HAL) which dispatches and routes RFSs. `GetSupport` depicts a message corresponding to the RFS `SupportRequest`. Upon receiving such a request, HAL generates a session identifier contained in the output message `AckSupportRequ`. A notification is sent to the requester (assuming a callback destination or notification endpoint has been provided) to deliver RFS status updates for example; processed RFSs can be retrieved via `GetSupportReply`. More information about this notification mechanism can be found in [Schall et al., 2008b].

```
<wsdl:portType name="HPSSupportPortType">
 <wsdl:operation name="GetSupport">
  <wsdl:input xmlns="http://.../addressing/wsdl"
   message="GetSupport" wsaw:Action="urn:GetSupport">
  </wsdl:input>
  <wsdl:output message="AckSupportRequ"/>
 </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="..." type="HPSSupportPortType">
 <soap:binding style="document"
  transport="http://xmlsoap.org/soap/http"/>
</wsdl:binding>
```

Listing 3.2: HPS WSDL binding excerpt.

Activity-based Interaction Context Model

Figure 3.3 depicts the applied context model (simplified for brevity - full version in [Schall et al., 2008a; Skopik et al., 2010a]), where `actors`, described by their `profiles`, perform activities. Activities reside in abstract scopes, e.g., all activities of a specific type (activity scope), or all activities belonging to a certain project (project scope). For instance, supporting the creation of white box test cases resides in a software development scope. Furthermore, actors are linked to collaboration partners in the network. These relations are reflected by FOAF profiles, are bound to scopes, and are characterized by various metrics that rely on previous interactions.

3.5 Social Trust in Collaborative SOA

Collaborative networks as outlined in the previous sections are subject to our trust studies. Unlike a security view, we focus on the notion of dynamic trust from a social perspective [Ziegler and Golbeck, 2007]. We argue that trust between community members is inevitable for successful collaborations. The notion of social trust considers the similarity of dynamically adapting skills and interests [Golbeck, 2009; Matsuo and Yamamoto, 2009]. In this chapter, we exemplarily focus on the establishment of trust through measuring interest similarities [Golbeck, 2009; Skopik et al., 2010a; Ziegler and Golbeck, 2007]:

- *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.

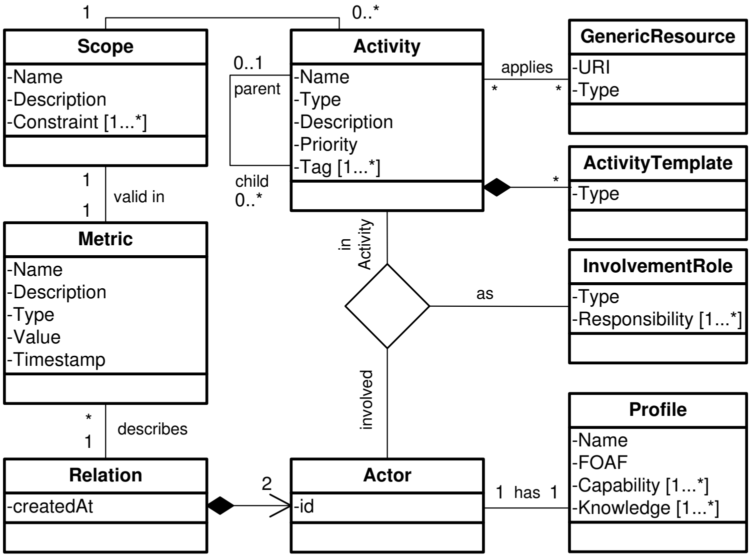


Figure 3.3: Context model: Linked actors perform activities in scopes.

Profile Similarity Measurement

In contrast to common top-down approaches that apply taxonomies and ontologies to define certain skill profiles and expertise areas, we follow a mining approach that addresses inherent dynamics of flexible collaboration environments. In particular, skills, expertise and interests change over time, but are rarely updated if they are managed manually in registries. Hence, we determine and update them automatically through interaction mining. As discussed before, interactions, such as task delegations and support requests are tagged with keywords. These keywords contribute to the description of activities, i.e., describe their focus. As actors process or discard received messages, our system is able to learn their expertise and centers of interests. We use task keywords to create dynamically adapting interest profiles based on tags and manage them in a vector space model [Salton and Buckley, 1988].

We assume that users pick keywords from a globally available taxonomy (such as the ACM taxonomy[1]) instead of adding arbitrary tags. The advantage is that we avoid (i) the use of synonyms, thus leading to inaccurate interest profiles (`notebook` v.s. `laptop` both meaning the same), (ii) equally meant but differently written tags (and their singular/plural forms), e.g., `social network` v.s. `social-networks`). An approach to similarity measurement that compensates such influences has been discussed in [Skopik et al., 2009a].

The profile vector $\mathbf{p_{u_i}}$ of actor u_i in Eq. (3.1) describes the frequencies f the tags $T = \{t_1, t_2, t_3 \dots\}$ are used in requests and delegated tasks accepted by actor u_i.

$$\mathbf{p_{u_i}} = \langle f(t_1), f(t_2), f(t_3) \dots \rangle \tag{3.1}$$

The tag frequency matrix $\mathcal{T}$ (3.2) in Eq. 3.2, built from profile vectors, describes the frequencies of used tags $T = \{t_1, t_2, t_3 \dots\}$ by all actors $N = \{u_1, u_2, u_3 \dots\}$.

$$\mathcal{T} = \langle \mathbf{p_{u_1}}, \mathbf{p_{u_2}}, \mathbf{p_{u_3}} \cdots \rangle_{|T| \times |N|} \tag{3.2}$$

The popular $tf{*}idf$ model [Salton and Buckley, 1988] introduces tag weighting based on the relative distinctiveness of tags; see Eq. (3.3). Each entry in $\mathcal{T}$ is weighted by the log of the total number of actors $|N|$, divided by the amount $n_t = |\{u_i \in N \mid tf(t, u_i) > 0\}|$ of actors who used tag t.

$$tf^{*}idf(t, u_i) = tf(t, u_i) \cdot \log \frac{|N|}{n_t} \tag{3.3}$$

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 $\mathbf{p_{u_i}}$ and $\mathbf{p_{u_j}}$; see Eq. (3.4). The result is a real value $sim_p \in [0, 1]$, whereas 0 denotes no overlap between used tags and 1 reflects identically used keywords.

$$sim_p(\mathbf{p_{u_i}}, \mathbf{p_{u_j}}) = \cos(\mathbf{p_{u_i}}, \mathbf{p_{u_j}}) = \frac{\mathbf{p_{u_i}} \cdot \mathbf{p_{u_j}}}{||\mathbf{p_{u_i}}||\ ||\mathbf{p_{u_j}}||} \tag{3.4}$$

[1] ACM Classification System: `http://www.acm.org/about/class/1998/`

The Interplay of Interest Similarity and Trust

In our model, trust $\tau(u_i, u_j) \in [0, 1]$ mainly relies on the interest and expertise similarities of actors (see [Golbeck, 2009] for details on that assumption). We two major concepts to facilitate the emergence of trust among network members.

Trust Mirroring. Trust τ_{mir} 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. 3.5). There is strong evidence that actors 'similar minded' tend to trust each other more than any random actors [Matsuo and Yamamoto, 2009; Ziegler and Golbeck, 2007]; e.g., movie recommendations of people with same interests are usually more trustworthy than the opinions of unknown persons.

$$\tau_{mir}(u_i, u_j) = sim_p(\mathbf{p_{u_i}}, \mathbf{p_{u_j}}) \tag{3.5}$$

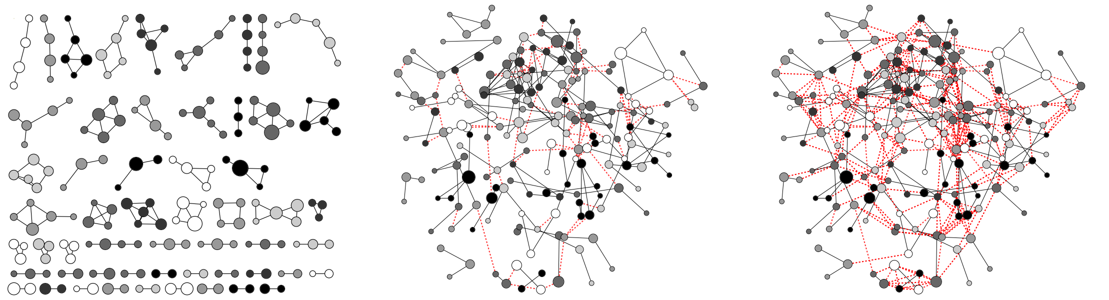
Trust Teleportation. Trust τ_{tele} is applied in sparse trust networks. We assume that u_i has established a trust relationship to u_j in the past, for example, relying on trust mirroring (applied in the following experiments) or based on u_j's capabilities to assist u_i in work activities (see for instance [Skopik et al., 2010a]). Therefore, others having interests and capabilities similar to u_j may become similarly trusted by u_i 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. We attempt to predict the amount of future trust from u_i to a third party u_k by attenuating $\tau(u_i, u_j)$ considering the profile similarity of the trustee u_j and the still unknown actor u_k. Since there may be multiple recommendations, in Eq. 3.6 the degree of teleported trust is additionally weighted by the profile similarities (sim_p) of u_i and each actor in the set of recommenders M'.

$$\tau_{tele}(u_i, u_k) = \frac{\sum_{u_j \in M'} \tau(u_i, u_j) \cdot (sim_p(\mathbf{p_{u_j}}, \mathbf{p_{u_k}}))^2}{\sum_{u_j \in M'} sim_p(\mathbf{p_{u_j}}, \mathbf{p_{u_k}})} \tag{3.6}$$

Eq. 3.6 deals with a generalized case where several trust relations from u_i to members of a group M' are teleported to a still untrusted actor u_k. Teleported relations are weighted and attenuated by the similarity measurement results of actor profiles.

Establishment of Social Relations

Based on a pre-configured profile similarity threshold $\vartheta_T \in [0, 1]$ the system can recommend new links. These links reflect potentially beneficial relations due to actors' interest similarities. Setting $\vartheta_T = 0$ means that all actors will be connected, thus resulting in a fully meshed network; setting $\vartheta_T = 1$ means that virtually no new relations will be introduced (except entire identical tagging profiles). Appropriate top and bottom limits are determined in the evaluation in Sect. 3.6. Practically, there should be enough links introduced to connect yet unconnected subcommunities, however, still considering their differing interests.



(a) No added edges (initial) (metr.: nc=55, nn=1.8, nd=0.010).

(b) Added edges ($\vartheta_T = 0.8$) (metr.: nc=8, nn=2.62, nd=0.014).

(c) Added edges ($\vartheta_T = 0.6$) (metr.: nc=1, nn=5.56, nd=0.03).

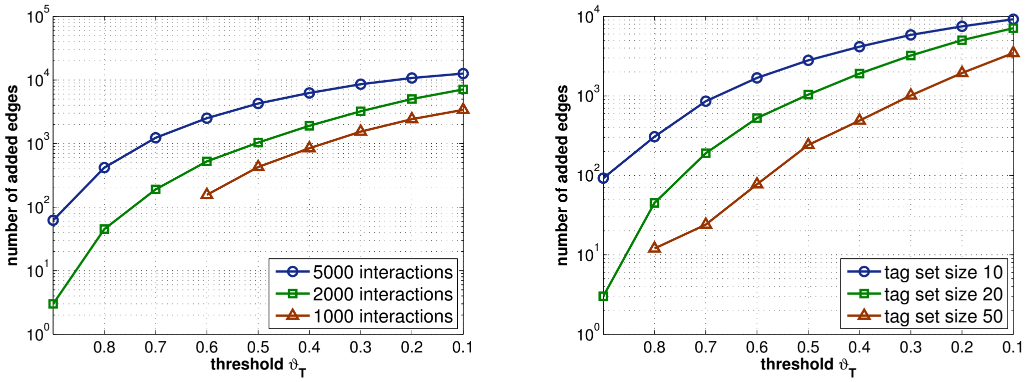
Figure 3.4: Gradually interconnecting trust network based on evolving interest similarities.

3.6 Evaluation and Discussion

We use a Web service testbed to simulate the interaction behavior in a SOA-based PVC. The purpose of the Genesis2 framework [Juszczyk and Dustdar, 2010] (in short, G2) is to support software engineers in setting up testbeds for runtime evaluation of SOA-based concepts and implementations. It allows to establish environments consisting of services, clients, registries, and other SOA components, to program the structure and behavior of the whole testbed, and to steer the execution of test cases on-the-fly. G2's most distinct feature is its ability to generate real testbed instances (instead of just performing simulations) which allows engineers to integrate these testbeds into existing SOA environments and, based on these infrastructures, to perform realistic tests at runtime.

Experiment Setup. The created environment consists of 200 services that interact in small groups of 2 to 5 members; thus 58 groups are built. Typically, groups of that size perform certain activities. During collaboration services interact by delegating tasks and requesting support; thus, in our simulation we let random services interact in fixed time intervals. Each interaction is tagged with a maximum of 3 keywords. We run different tests and vary the number of globally known tags, as well as the amount of occurring interactions. The results of these experiments help to determine appropriate similarity thresholds to introduce new (trust) edges in the collaboration graph for recommending and facilitating future collaborations.

Results. Figure 3.4 demonstrates the effects on the graph structure when new links are introduced (red dashed lines). The size of the nodes denote their involvement in activities, i.e., the number of received interactions. Additionally the single groups are colored for better visibility. In the beginning (Figure 3.4(a)) various small components exist but are not interconnected. These components represent small groups of actors that interact in context of their activities. Links reflect interaction paths that may lead to trust over time (see [Skopik et al., 2010a]). After finishing the simulation, we gradually introduce new links using our concepts of *trust mirroring* and *trust teleportation*. The threshold ϑ_T denotes the lower boundary of tag usage similarity, i.e., all pairs of actors that have higher profile similarity than ϑ_T are connected. Thus, higher ϑ_T leads to less connections. The optimal number of introduced edges in the graph depends on several properties. On the one side, independent components should be connected, so that



(a) Varying num of interactions (20 tags).

(b) Varying tag sets (2 000 interactions).

Figure 3.5: Impact of ϑ_T on number of added edges.

previously unknown actors get introduced to each other. On the other side, simply connecting all actors with each other is obviously not beneficial. An optimal connection is hard to determine, but various graph metrics [Romesburg, 2004] are appropriate indicators, such as *number of connected components* nc, *average number of neighbors* nn, or *network density* nd.

In the following experiments, we determine the number of added edges depending on the configured threshold ϑ_T. The first set of experiments investigates the impact of varying numbers of interactions in our scenario (see results in Figure 3.5(a)). Actors pick up to three tags from a globally available tag set of size 20 to annotate their interactions. Obviously more interactions lead to bigger profiles as more tags are collected. Therefore, after longer collaboration (e.g., 5 000 interactions in the whole scenario) more similar actors can be determined than after a lower amount of interactions (e.g., 2 000). For only 1 000 interactions a threshold of 0.6 is not exceeded. Note, numbers on the x-axis are in the reverse order. Normally, one would start with introducing links between actors with identical profiles ($\vartheta_T = 1$) and than gradually degrade that value until a satisfying degree of connection has been reached. Also note that the y-axis uses a logarithmic scale. In the second set of experiments, 2 000 interactions are performed, however, the number of globally available tags is changed. This means that actors can choose from 10, 20 or 50 different keywords to annotate their interactions. As expected, for smaller tag sets higher profile similarity is achieved (see results in Figure 3.5(b)).

3.7 Prototype and Implementation

Interaction Logging

The previously presented results are based on G2's testbed generation capabilities and a framework for monitoring and logging interactions between services. Interactions are captured through (SOAP) message interceptors deployed within the service runtime environment. Logged messages are persistently saved in a database for analysis. An example interaction log is shown by Listing 3.3, which includes various SOAP header extensions for message correlation and context-aware interaction analysis.

The most important extensions are (see [Skopik et al., 2010a] for details on the implementation):

- `Timestamp` captures the actual creation of the message and is used to calculate temporal interaction metrics, such as average response times.

- `Message flags`, including priority of messages.

- `Activity uri` describes the context of interactions (see Figure 3.3 for the model).

- `MessageID` enables message correlation, i.e., to properly match requests and responses.

- `WS-Addressing` extensions [Box et al., 2004], besides `MessageID`, are used to route requests through the network.

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:vietypes="http://viete.infosys.tuwien.ac.at/Type"
 xmlns:hps="http://www.infosys.tuwien.ac.at/hps/"
 xmlns:rfs="http://.../socialsoa/rfs">
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/.../addressing"
 <soap:Header>
  <vietypes:timestamp value="2010-05-25T17:24:18"/>
  <vietypes:msgflags priority="urgent"/>
  <vietypes:activity url="http://.../Activity#42"/>
  <wsa:MessageID>uuid</wsa:MessageID>
  <wsa:ReplyTo>http://.../Actor#Harald</wsa:ReplyTo>
  <wsa:From>http://.../Actor#Florian</wsa:From>
  <wsa:To>http://.../Actor#Daniel</wsa:To>
  <wsa:Action>http://.../Type/RFS</wsa:Action>
 </soap:Header>
 <soap:Body>
  <hps:Request>
   <rfs:subject>ACM taxonomy for my paper?</rfs:subject>
   <rfs:requ>What ACM categories fit best
             to my paper?</rfs:requ>
   <rfs:resource>
    <vietypes:resource type="doc" uri="http://..."/>
   </rfs:resource>
   <rfs:keywords>document, categorization</rfs:keywords>
  </hps:Request>
 </soap:Body>
</soap:Envelope>
```

Listing 3.3: Simplified RFS via SOAP example.

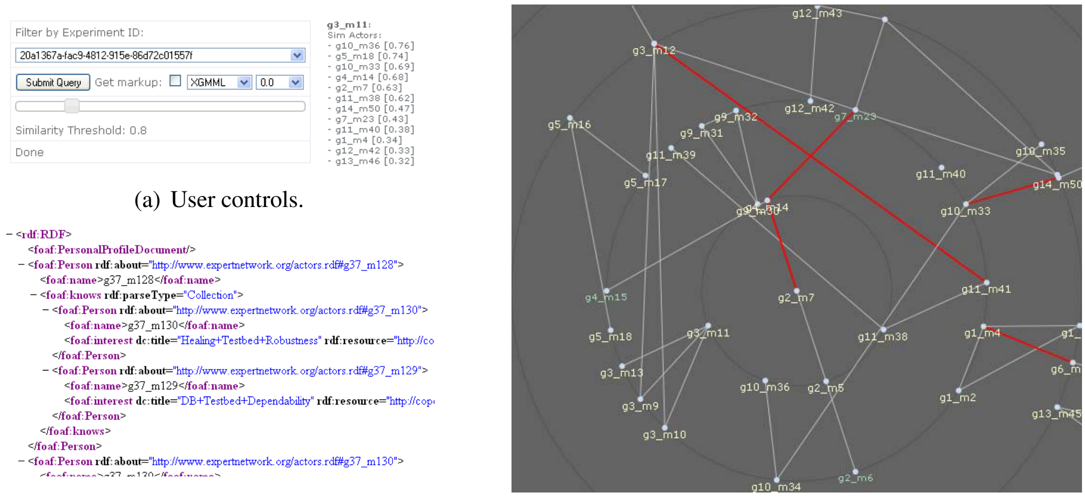
The SOAP body transports the actually exchanged message. In this example a request for support (RFS) [Skopik et al., 2010a] shows how one actor requests some help from another one in the motivating collaboration scenario. Note, interactions are only captured to collect keywords and support the creation of user profiles. Logged data can be purged immediately after keyword extraction. Thus, our approach of interaction observation is less intrusive compared to others (e.g., semantic analysis of captured messages). We understand today's privacy concerns as a big issue of most systems that log user data for adaptation purposes (such as establishing network links).

User Tools

The implemented prototype includes a Web-based formation tool assisting users in analyzing various thresholds for trust-based link establishment between independent networks. Figure 3.6

shows screenshots of the tool and an example FOAF profile that can be retrieved from the Web application. All user interfaces have been implemented using state-of-the-art Web technologies such as ASP.NET MVC hosted by a .NET 3.5 runtime. Our implementation comprises a network visualization view built on top of a JavaScript library[2]. The network view is obtained by mapping raw SOAP-interactions into a graph representation composed of nodes (services) and edges (interaction links). Each link holds additional data such as the number of exchanged messages between services. Nodes are associated with profiles and also groups indicated by a prefix in the view in Figure 3.6(c) representing the initial disconnected components of the interaction network.

As a first step, the user accesses information captured from the service-oriented collaboration environment (Figure 3.6(a)). In our implementation, this is performed by selecting a particular set of logs which are associated with an `Experiment ID`. After issuing the corresponding query, a graph is visualized typically consisting of several disconnected components. The tool queries a `Similarity Service` to obtain a set of similar actors for each node in the network (see list on the right side in Figure 3.6(a)). The presented list shows actor name and degree of similarity. By default, the collaboration network is visualized in a graph view as depicted in Figure 3.6(b). The user is able to select a similarity threshold by moving a slider bar. A reduced (demanded) similarity threshold results in trust edges being added to the visualization (*color online*: depicted as red colored edges between nodes). Alternatively, interactions can be retrieved as FOAF profiles (see Figure 3.6(c)) that include `foaf:interest` tags. This mechanism can be used to retrieve and aggregate captured profiles from distributed environments (e.g., from multiple instances of the logging service).



Filter by Experiment ID:

20a1367a-fac9-4812-915e-86d72c01557f

Get markup: □ XGMML 0.0

Similarity Threshold: 0.8

Done

```
g3_m11:
Sim Actors:
- g10_m36 [0.76]
- g5_m18 [0.74]
- g10_m33 [0.69]
- g4_m14 [0.68]
- g2_m7 [0.63]
- g11_m38 [0.62]
- g14_m50 [0.47]
- g7_m23 [0.43]
- g11_m40 [0.38]
- g1_m4 [0.34]
- g12_m42 [0.33]
- g13_m46 [0.32]
```

(a) User controls.

```
- <rdf:RDF>
    <foaf:PersonalProfileDocument/>
  - <foaf:Person rdf:about="http://www.expertnetwork.org/actors.rdf#g37_m128">
      <foaf:name>g37_m128</foaf:name>
    - <foaf:knows rdf:parseType="Collection">
      - <foaf:Person rdf:about="http://www.expertnetwork.org/actors.rdf#g37_m130">
          <foaf:name>g37_m130</foaf:name>
          <foaf:interest dc:title="Healing+Testbed+Robustness" rdf:resource="http://co
        </foaf:Person>
      - <foaf:Person rdf:about="http://www.expertnetwork.org/actors.rdf#g37_m129">
          <foaf:name>g37_m129</foaf:name>
          <foaf:interest dc:title="DB+Testbed+Dependability" rdf:resource="http://cop
        </foaf:Person>
      </foaf:knows>
    </foaf:Person>
  - <foaf:Person rdf:about="http://www.expertnetwork.org/actors.rdf#g37_m130">
```

(b) Example of FOAF profile.

(c) Network visualization view.

Figure 3.6: Web-based formation tool and network visualization.

[2] Graph visualizations for the Web: `http://thejit.org`.

3.8 Conclusion

In this chapter, we discussed concepts and mechanisms to enable service-oriented virtual communities. These communities rely on SOA technology and enable humans to collaborate in a service-oriented manner. Existing approaches in service-oriented systems typically aim at devising a predefined interaction model between people and services. The presented work attempts to align SOA concepts and service-oriented collaborations driven by dynamics such as evolving skills and preferences. We argue that the automated management of social aspects including trust are key issues. Since personal relations are of paramount importance in these social environments, we introduced the concept of social trust to establish links between community members.

CHAPTER 4

Computational Social Network Management

Flexible interactions in complex social and service-oriented collaboration systems increasingly demand for automated adaptation techniques to optimize partner discovery and selection. Today, applications of complex service-oriented systems can be found in crowdsourcing environments. In such environments, collaborations are typically short-lived and strongly influenced by incentives and actor behavior. As actors prove their reliable and dependable behavior in jointly performed activities, they become increasingly considered as invaluable partners. A social network builds a strong basis to enable successful collaborations between crowd members. In order to keep track of the dynamics in such systems, it is inevitable to apply an autonomous approach to manage social network structures automatically using captured interaction data. Thus, we introduce an adaptation concept that accounts for emerging social relations based on varying interaction behavior of collaboration partners. We describe the foundational concepts for dynamic social link management in Web-based collaborations. We highlight major concerns of computational models in highly dynamic networks and deal with temporal aspects such as supporting the emergence of relations, efficient update mechanisms, and aging of relations.

4.1 Introduction

Over the past years, the Web has transformed from a pool of statically linked information to a people-centric Web. Various Web-based tools and services have become available enabling people to communicate, coordinate, and collaborate in a distributed manner. *Crowdsourcing* has emerged as an important paradigm for providing human problem solving techniques on the Web. More often than widely recognized, companies outsource even short repetitive tasks which are difficult to be processed by software. Applications range from enterprise environments [Vukovic, 2009] to open Internet based platforms such as Amazon Mechanical Turk

(MTurk[1]). These online platforms distribute problem-solving tasks among a group of humans. Crowdsourcing follows the 'open world' assumption allowing humans to provide their capabilities to the platform by registering themselves as services. Some of the major challenges are monitoring of crowd capabilities, detection of missing capabilities, strategies to gather those capabilities, and tasks' status tracking [Brabham, 2008].

Service-oriented architectures [Alonso et al., 2003] (SOA) enable the design of applications that are composed from the capabilities of distributed services that are discovered at runtime. Unlike traditional SOA-based approaches, we consider complex service-oriented systems that are established upon the capabilities of human and software services [Schall et al., 2008b]. The integration of human capabilities in a service-oriented manner is motivated by the difficulties to adopt human expertise into software implementations. Instead of dispensing with human capabilities, people handle tasks behind traditional service interfaces. In contrast to process-centric flows (top-down compositions), we advocate flexible compositions wherein services can be added at any time exhibiting new behavior properties. Hence, our service-oriented approach to the design and implementation of flexible collaboration networks enables the realization of versatile application scenarios. However, especially the involvement of and dependencies on humans as a part of flexible compositions has a major impact on all aspects of the system since dynamics and evolution are driven by software services and human behavior [Psaier et al., 2010b]. We propose an interaction mining and self-adaptation approach for service-oriented collaboration networks. In socio-computational crowdsourcing environments, where people (and services) *dynamically* interact to perform activities, reliable and dependable behavior promotes the emergence of *social relations* and *trust* [Skopik et al., 2010a]. As collaborations are increasingly performed online, supported by service-oriented technologies, interactions have become observable. That fact facilitates the application of *computational social link management models*.

We argue that relations from a social perspective can neither be reliably identified in advance nor defined statically. They rather emerge dynamically upon interaction behavior of humans and services, and evolve over time. Sophisticated social network management models need to account for these properties to reflect real situations as close as possible. Moreover, in highly flexible environments, interaction behavior may alter quickly and, therefore, the underlying model has to be updated in sufficiently short cycles. However, in large-scale networks with potentially thousands of participants, updating relations in short intervals is not feasible due to limitations of computational power. Hence, relations have to be updated and altered selectively.

Here, we mainly address two issues of *computational social network models* resulting from interaction flexibility:

- *Efficiency* in terms of performance is realized by carefully selecting the most critical relations in a network to be refreshed in adaptive update cycles. Scheduling of updates fundamentally depends on the actors' interaction behavior, and the community's utility of frequent updates.

- *Effectiveness* in terms of functionality deals with the application of algorithms to let a model reflect the dynamically changing environment as close as possible. Our approach

[1] MTurk: `http://www.mturk.com/`

accounts for the different lifecycle phases of relations: *emergence-, update-, and aging phase*.

4.2 Service-orientation in Crowds

We motivate our work with a concrete scenario and outline fundamental principles that have been discussed in earlier publications [Skopik, 2010; Skopik et al., 2010a], building the basis for the work presented here.

Scenario

Let us consider a service-oriented crowdsourcing environment[2] to introduce our concepts. The environment consists of professionals and experts who interact and collaborate by the means of information and communication technologies to perform work. The actors, i.e., the community network members, provide *help and support* on requests of each other and thus, perform activities collaboratively. In such a socio-computational crowdsourcing environment actors have to register at a central community management service to become part of the network. Typically they register basic profile properties, including their education, employment status, certified skills and project experience. Furthermore, they can register HPSs [Schall et al., 2008b] they provide. In the described environment, network members perform activities. Activities structure information in ad-hoc collaboration environments, including the goal of the ongoing tasks, involved actors, and utilized resources. In the proposed socio-computational crowdsourcing environment, activities are assigned from outside the community. Typically, they are flexibly created by a crowd management system, when external tasks, e.g. belonging to a higher-level process, require crowd members' attention.

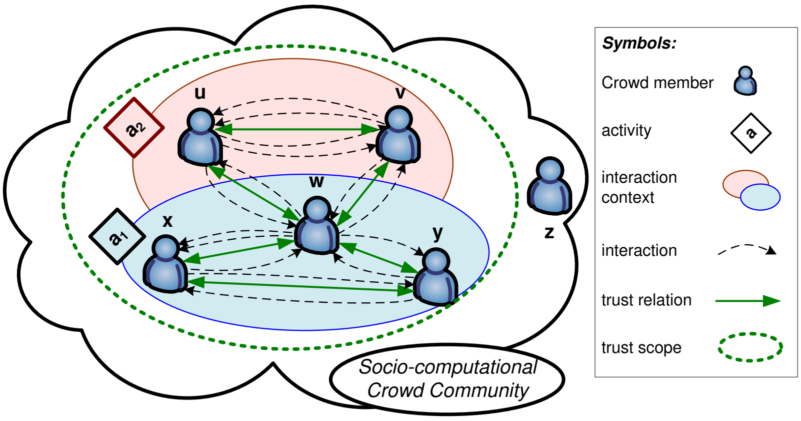


Figure 4.1: Service-oriented socio-computational crowdsourcing.

[2] The basic properties of this environment are derived from [Skopik et al., 2010e]. Thus the foundational concepts are similar.

In the scenario depicted in Figure 4.1, we particularly investigate crowd internals. The major objective here is that crowd members *collaboratively* perform activities. In particular, members w, x, y are involved in activity a_1, and members u, v, w are assigned to activity a_2. Since w is involved in both activities, interaction contexts overlap. Further crowd members, for instance z may be present, but are of no interest for further explanations. We assume activity a_1 is a software implementation activity and a_2 is a software testing activity in some higher-level software development process (not depicted here). Members in the respective activities coordinate their contributions (e.g., the creation of single classes and test cases) by using HPSs and sending requests for support (RFSs) - as discussed in the last chapter of this thesis. That way, they delegate work and notify partners about (partly) finished artifacts. The dashed arrows represent such kinds of interactions. The interaction contexts, described by activity a_1 (reflected by the blue-shaded area) and a_2 (reflected by the red shaded area), hold information about involved actors, goal of the activity, temporal constraints (start, duration, milestones), assigned resources, planned costs, risk with respect to the whole software development process and so on. The detailed description is out of scope of here, however, we conclude, that an activity holistically describes the context of an interaction in our environment model.

Social trust emerges from interactions and is bound to a particular scope (e.g., expertise area, or project boundaries). Therefore, we aggregate interactions that occurred in a pre-defined scope, calculate metrics (numeric values describing prior interaction behavior), and interpret them to establish trust (models are extensively discussed in [Skopik et al., 2010a]). The scope of trust is reflected by the green dashed ellipse in Figure 4.1. In the given scenario, the scope comprises trust relations between crowd members regarding collaborations in ‘software development’. So, regardless of whether interactions took place in context of activity a_1 or a_2, interactions of both contexts are aggregated to calculate metrics, because both interaction contexts adhere to the scope of software development. Finally, interaction metrics are interpreted using rules, and the degree of trust between each pair of previously interacting members is determined.

Fundamental Principles

We extensively studied flexible interactions [Schall et al., 2008b], metrics [Skopik et al., 2009a, 2010a], and monitoring [Psaier et al., 2010b] of service-oriented collaboration environments in our previous work. We overview the main principles that are the basis for the proposed adaptive social network management model. We previously investigated models and techniques to cover:

- *Context-aware Interaction Models.* Usually, interactions on the Web are easily observable. In particular, service-oriented systems allow for context-aware logging of SOAP-based interactions.

- *Mining of Interaction Metrics from SOAP Logs.* Metrics describe the interaction and collaboration behavior of users (e.g., in terms of reliability, openness, contributing behavior) and can be determined through advanced log analysis.

- *Inference and Interpretation of basic Social Relations.* Rule engines and fuzzy inference approaches allow for a situation-based interpretation of interaction metrics.

- *Interaction Patterns spanning numerous Actors.* Interaction patterns support the emergence of social relations by introducing (i.e., connecting) previously unknown actors; for example, by enabling delegations of requests to third parties.

4.3 Computational Link Model

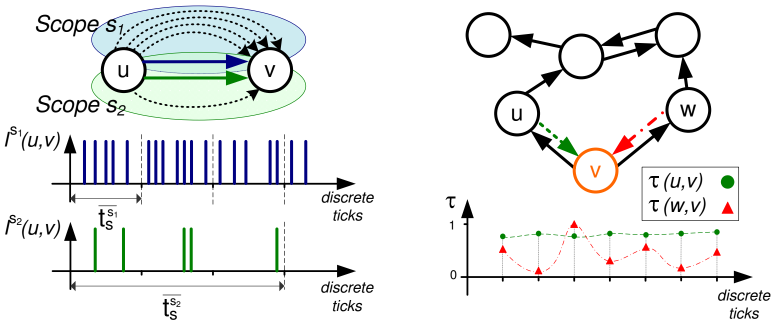
Reliable social trust relations in dynamic (crowd) environments typically cannot be statically defined, but evolve over time. An efficient social trust management model must frequently refresh its data to keep track of the real situation.

Challenges

We model the following fundamental lifecycle phases of social trust relations to account for their dynamic nature and temporal aspects: (i) *Emergence* deals with introducing new relations upon ongoing interactions; (ii) *Update* deals with refreshing existing relations based upon experiences made in recent interactions; (iii) *Aging* deals with degrading and deleting outdated relations.

Trust Emergence. Several concepts of link prediction exist to introduce new relations between actors. Recent research shows, that there is a strong dependency between interest similarities and trust [Golbeck, 2009; Skopik et al., 2009a; Ziegler and Golbeck, 2007]. Furthermore, there are approaches to recommend collaboration partners due to required expertise and reliable working styles. However, there is no evidence that collaboration partners will behave trustworthy according to predictions. Thus, in our model social trust relations are only built upon personal experiences from recent interactions and, hence, only if there is a sufficient amount of interactions to reliably infer trust. We enable the application of this concept through delegations, where unconnected actors start interacting within triadic closures [Watts, 2003] that support the emergence of trust relations.

Trust Update. Since the behavior of actors in a network may change due to various reasons, e.g., shift of interests, work overload, and search for new work opportunities, trust relations will alter as well. Hence, frequent synchronization with the real world is critical to computational



(a) Interaction sparsity.

(b) Actor (non-)uniformity.

Figure 4.2: Challenges for interaction-based social relation update mechanisms in dynamic environments.

trust models. For that purpose, the behavior of actors is sampled (i.e., observed through monitoring) in subsequent intervals and results are used to update the strength of social trust relations. A major challenge is to determine the appropriate sampling intervals (e.g., see also [Domingo et al., 2002]). Figure 4.2 visualizes two fundamental challenges of trust update mechanisms:

- *Interaction Sparsity.* In different scopes s_1, s_2 occur varying types and amounts of interactions. Since a larger amount of interactions is needed to detect trends in an actor's behavior (e.g., responsiveness, availability), it is a challenging task to set the right size of sampling intervals ($\overline{t_s}$). Intervals that are too short prohibit reliably behavior analysis; however, if intervals are too long, sudden changes of behavior cannot be detected accordingly. The length of $\overline{t_s}$ mainly depends on the scope and the regular interaction behavior of actors therein.

- *Actor Uniformity.* The uniformity describes the consistency of an actor (i) towards the same partner over time; (ii) towards different interaction partners. In Figure 4.2(b) v behaves consistently trustworthy towards u, therefore, the level of trust $\tau(u, v)$ remains high over several sampling intervals. However, v alters dynamically his behavior toward w, and hence, w's trust in v changes rapidly over time[3]. Intuitively, in the second case of quickly changing behavior, smaller sampling intervals are required to capture v's behavior changes, while in the first case, the sampling interval to refresh already well-known constant behavior can be longer. Apart the actors' interaction behavior, external adaptation requirements may influence the determination of appropriate sampling intervals; e.g., in the case of pre-defined upper time limits to quickly react on sudden changes.

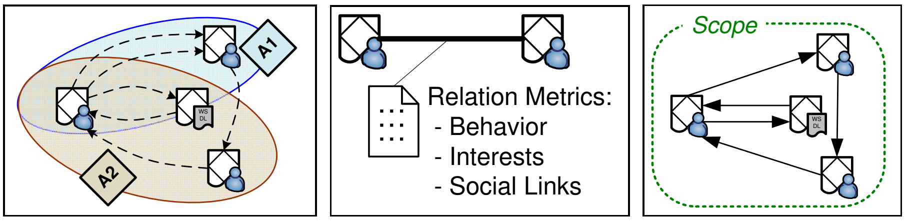
Trust Aging. If the amount of interactions between two actors falls below a certain threshold, or two actors completely stop interacting, trust relations undergo an aging process. Since in this phase no further evidence occurs for reliably interaction behavior, relations are not updated any longer. Therefore, trust relations (i.e., their strength) will degrade to a neutral state and are finally removed from the graph G. Intuitively, well established and consolidated long-term relations mature slower compared to fragile short-term relations. Hence, strengthened long-term relationships are able to bridge longer 'interaction gaps', while short-term relations disappear faster.

On the Emergence of Trust

In contrast to a widely used security perspective, we define (social) trust[4] to rely on the interpretation of previous collaboration behavior [Skopik et al., 2010e] and additionally consider the similarity of dynamically adapting interests [Golbeck, 2009; Skopik et al., 2009a]. Especially in collaborative environments, where users are exposed to higher risks than in common social network scenarios [Dwyer et al., 2007], and where business is at stake, considering social trust

[3] Here, one could argue that oscillating interaction behavior is not trustworthy at all. However, we apply an optimistic point of view and appreciate recovery from unreliable behavior by increasing trust levels accordingly.

[4] This concept is discussed in depth in [Skopik, 2010]. Here, we revisit only the basics that are fundamental for the extensions discussed in this work.



(a) Interactions.

(b) Relation metrics.

(c) Scoped trust.

Figure 4.3: 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) trust inference in scopes by interpretation of metrics.

is essential to effectively guide interactions [Metzger, 2004]. Hence, we define trust as follows [Grandison and Sloman, 2000; Mui et al., 2002; Skopik et al., 2010e]:

> *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.*

As introduced in the previous chapter, not only service interactions, but also human interactions may rely on SOAP (e.g., see Human-Provided Services [Schall et al., 2008b] and BPEL4People [Agrawal et al., 2007]), which is the state-of-the-art technology in service-oriented environments, and well supported by a wide variety of software frameworks. This fact enables the adoption of various available monitoring and logging tools to observe interactions in service-oriented systems. Various metrics can be calculated from analyzing interaction logs. These 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, we argue that social trust relations largely depend on personal interactions. We model a community of actors 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*.

An outline of our approach to automatic interaction-based trust inference is depicted in Figure 4.3. As motivated in the introduced use case, people interact to perform their tasks. This work is modeled as activities, that describe the type and goal of work, temporal constraints, and used resources. As interactions take place in context of activities (Figure 4.3(a)), they can be categorized and weighted. Interaction logs are used to infer metrics that describe the relation of single actors (Figure 4.3(b)), such as their behavior in terms of availability and reciprocity. We support the diversity of trust by enabling the flexible aggregation of various interaction metrics that are determined by observing ongoing collaborations. Finally, available relation metrics are weighted, interpreted, and composed by a rule engine [Skopik et al., 2010a]. The result describes trust between the actors with respect to scopes (Figure 4.3(c)). For instance, trust relations in a scope 'scientific dissemination' could be interpreted from interaction behavior of actors in a set of paper writing activities.

Adaptive Trust Update and Aging Models

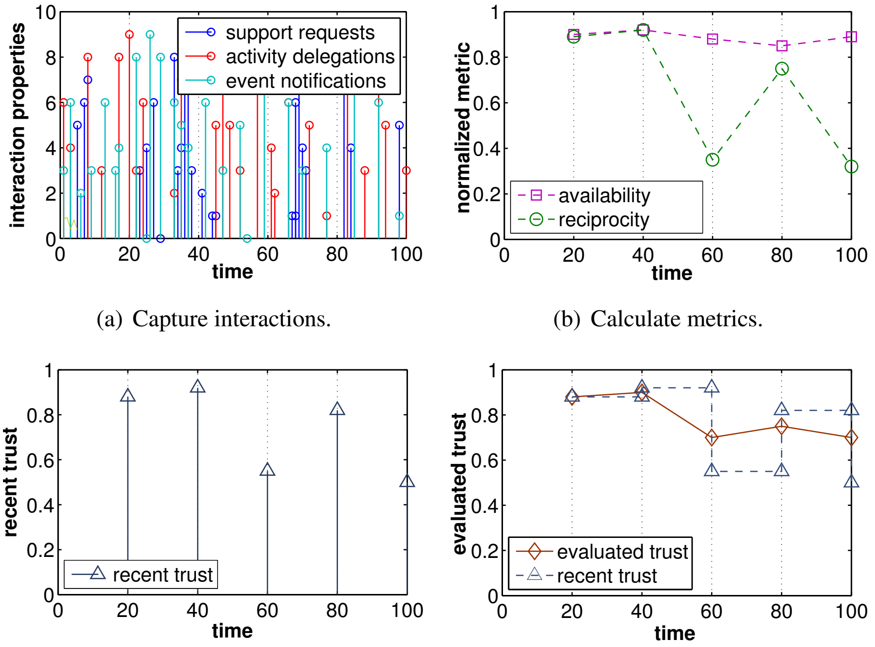
In our model, the selection of relations to be updated and update intervals rely on two influential factors (i) the variance of user behavior, reflected by the dynamics of interaction metrics, (ii) the sparsity of interaction, i.e., a certain amount of interactions is required to reliably determine interaction behavior. Note, for the initial establishment of social trust relations interactions are not mandatory, but relations can be introduced manually. This is a sufficient assumptions, as in real environments actors are selected based on recommendations or reputation values. However, once established, manually introduced relations are automatically updated by the system considering update and aging parameters. The advantage of manually introduced relations is the reduced effort when processing interaction logs. In this case, only interactions between actors who are already linked in the social network need to be handled.

Fundamental Update Mechanisms. Figure 4.4 summarizes the fundamental mode of operation of our temporal social trust management approach. Interactions from u to v (a), occurring between two sampling instants (in this example every 20 ticks), are utilized to calculate interaction metrics $M(u, v)$ (b). These metrics describe actor v's behavior toward u in scope s, and is inferred in consecutive sampling intervals $\overline{t_s}$; for instance, v's availability to u's requests and its reciprocity [Mui et al., 2002]. In the given example, the availability remains high, while v's reciprocity toward u is unsteady. We assume the level of trust $\tau^s(u, v)$ relies on both metrics. Therefore, in (c), recent trust, grounded in previous interaction behavior of v toward u in time interval $\overline{t_s}$, is inferred. This $\widehat{\tau}^s(u, v)$ is visualized in Figure 4.4(c) at the sampling instants.

The strength/weight of evolving long-term relations τ_i^s (see Figure 4.4(d)) is updated periodically in successive time intervals t_i (e.g., days in mid-term collaboration scenarios), numbered with consecutive integers starting with zero. We denote trust values in scope s (context) calculated at time step i as τ_i^s. As the strength of relations is evolving over time, we do not simply replace old values, i.e., τ_{i-1}^s, with newer ones, but merge them accordingly. For this purpose we apply the concept of exponential moving average (EMA), to smoothen the sequence of calculated values as shown in Eq. 4.1. Using this method, we are able to adjust the importance of the most recent behavior (leading to $\widehat{\tau}^s$) compared to historical values. The smoothing factor $\alpha \in [0, 1]$ can be dynamically adapted. The impact of the most recent values $\widehat{\tau}$ on well established long-term relations might be lower than on recently emerged and still fragile short-term relations. Long-term relations are normally based on large sets of previous experiences and sporadic short-term behavior changes, e.g., sporadic unreliability, may not have major impact. Indeed, this behavior is subjective, and our model can not dictate the application of this feature, but provides the means to cover such situations appropriately.

$$\tau_i^s = \alpha \cdot \widehat{\tau}^s + (1 - \alpha) \cdot \tau_{i-1}^s \tag{4.1}$$

Adaptive Sampling. The fundamental approach simply updates τ_i at each time tick t_i, and the interval between instant t_i and t_{i+1} is constantly $\overline{t_s}$. However, in most real situations an adaptive sampling interval $\overline{t_s}$ is desired due to two reasons: (i) interaction sparsity, and (ii) actor (non-)uniformity (see Section 4.3). Intuitively, relations to erratic actors that change their behavior quickly and dynamically have to be updated more often, than the relations to actors with stable/consistent behavior. From a performance perspective, longer update cycles of stable con-



(a) Capture interactions.

(b) Calculate metrics.

(c) Interpret trust.

(d) Support trust evolution.

Figure 4.4: Illustration: update of trust relations based on captured recent interactions.

nections allows the system to focus on unstable connections. Hence, while in the fundamental case we set the update interval in a particular scope s to $\overline{t_u^s} = \overline{t_s}$ (equal to the system sample interval), we introduce now an approach to adapt $\overline{t_u^s}$ dynamically within the limits according to Eq. 4.2.

$$\overline{t_{u_{min}}^s} \leq \overline{t_u^s} \leq \overline{t_{u_{max}}^s} \tag{4.2}$$

Both limits are pre-configured and determined by the interaction sparsity. Furthermore, $\overline{t_{u_{min}}^s} = \lambda_1 \cdot \overline{t_s}$ and $\overline{t_{u_{max}}^s} = \lambda_2 \cdot \overline{t_s}$ and $\lambda_1 \leq \lambda_2$ for $\lambda_1, \lambda_2 \in \mathbb{N}$. The basic challenge is to find appropriate update intervals $\overline{t_u^s}$, in terms of efficiency and effectiveness of social trust management. Remember, although static relations do not need frequent updates, sudden behavior changes must not be neglected. The mode of operation of our adaptive approach is exemplarily depicted in Figure 4.5.

We interpret actor behavior, reflected by metrics M as a continuous 'signal' that is sampled from interactions in consecutive time intervals $\overline{t_s}$. Therefore, metrics reflect the changeability of an actor's behavior. Figure 4.5(a) shows the temporal evaluation of two interaction metrics. While the values of one metric are nearly constant over time, the other suddenly drops at time tick 40, remains low, and increases again at tick 180. We detect such rapid changes with precisely configured *event triggers*. Once sudden events are detected, such as the variance of the most recent values is above a threshold, or the number of unreplied requests is considerably high, an update operation is triggered (see tick 40). Then, when the metric values are stable, $\overline{t_u^s}$ is extended by one $\overline{t_s}$ in each update cycle. In our examples $\overline{t_s} = 20$, therefore, after tick 40 the next update intervals have the lengths $\overline{t_s}(= \overline{t^s_{u_{min}}})$, $2 \cdot \overline{t_s}$, and $3 \cdot \overline{t_s}$. However, at tick 180 a sudden behavior change is detected and link weights are sampled as soon as possible (instead of waiting a period of $4 \cdot \overline{t_s}$.

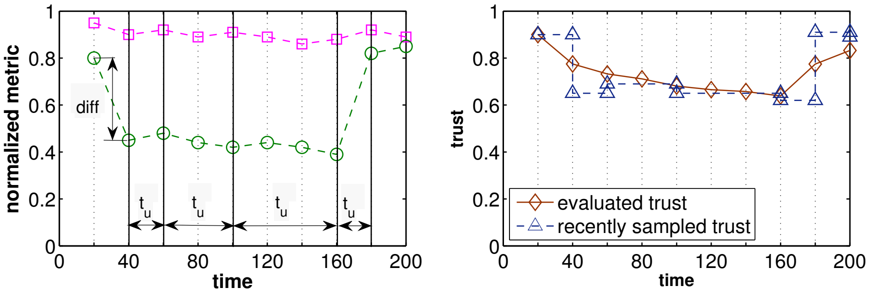
Typically rather simple and easily computable metrics that characterize the actor behavior and can efficiently capture behavior changes, are used to trigger update actions. While at least this set of metrics, is calculated at each $\overline{t_s}$, the larger amount of (typically more complex) metrics and finally trust values are refreshed only after adaptive intervals $\overline{t^s_u}$. This is visualized in Figure 4.5(b). Sampled trust $\widehat{\tau}$ is only captured at intervals $\overline{t^s_u}$. However, a temporal evaluation (Eq. 4.1) is still applied at each t_i (as in the fundamental approach), but based on the most recent $\widehat{\tau}$.

Trust Aging Model. As social trust relations in the real world degrade if people do not frequently interact, also relations in the computational model underlie an aging process. While it is intuitive that relations will become invalid over time, it is quite hard – if not impossible – to realistically reflect this aspect in a mathematical model. Our approach, as defined in Eq. 4.3, provides some parameters for tuning the aging process, while it is not too complex to be applied in real environments.

$$\tau^s_n = \tau^s_i \cdot e^{-(\tau^s_{n-1} \cdot \Delta t)^{2\gamma}} \tag{4.3}$$

The variable τ^s_i represents the latest determined value based on interactions in the update procedure that is degraded exponentially, configured by the decay factor γ ($\gamma \leq 1$). So, trust τ^s_n at time tick t_n is calculated by degrading τ^s_i depending on the time span $t_n - t_i$. The quality of computed relationships suffers if links are not periodically refreshed through new interactions. While immediately after updating a relation ($\Delta t = 0$) the strength is not altered ($\tau_n = \tau_i$), the aging process produces trust results asymptotic to zero with $\Delta t \to \infty$.

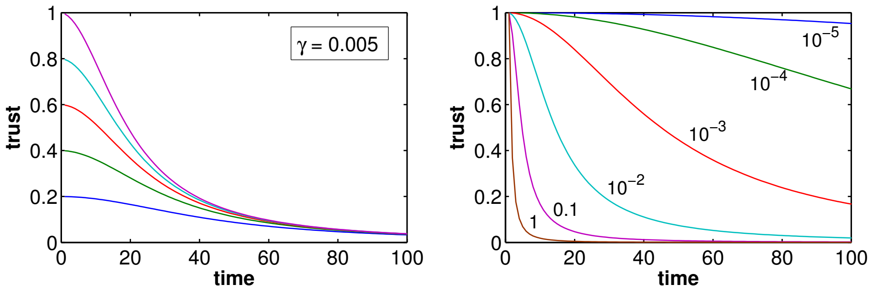
Adaptive aging refers to the dynamic adaptation of γ, hence, the older a relation, the slower may be the applied aging process. Furthermore, as in real life, the decay of links can be comparably fast in the beginning, while the actual removal of relations takes longer time. The adaptation of the decay factor may depend on actors interaction consistency. The configuration of the aging model (see Figure 4.6) is still an open issue. On the one side domain experts could care for this based on best practice, on the other side there exist concepts that let systems adapt parameters



(a) Trigger metric changes.

(b) Adaptive evaluation.

Figure 4.5: Illustration: adaptive update of trust relations through behavior triggers.



(a) Varying levels of trust w.

(b) Varying decay factors γ.

Figure 4.6: Illustrative example of trust aging from different strength levels w and for different decay factors γ.

autonomously [Bryl and Giorgini, 2006] to optimize the aging process. However, we design the computational model to be flexible enough to cover various demands on temporal properties; e.g., sampling intervals ($\overline{t_s}$), impact of new values (α), decay factor (γ).

Computational Social Network Algorithm

Algorithm 1 formulates the emergence of new trust relations (Line 26), updates of existing ones (Line 8), and their aging in case no interactions take place between connected actors (Line 31). It manages links between a subset of nodes $N' \subseteq N$ in an existing trust network $G_T = (N, E_T)$ for a predefined set of scopes (depending on already existing links that need to be updated – see Line 5). In case the amount of interactions to reliably infer behavior (and trust) is above a predefined threshold (ϑ_I^s depends on the 'usual' amount of interactions in a scope), new relations are introduced and existing ones updated respectively. New edges are added to G_T if a significant amount of interactions took place between two actors but no trust relations exist yet (Line 26). The level of trust (τ) is inferred from measured metrics (see [Skopik et al., 2010a] for details about rule-based trust inference) and updates are scheduled as soon as possible – still accounting for interaction sparsity in the given scope ($\overline{t_{u_{min}}^s}$). Updates are performed due to two events: (i) an update has been scheduled for a given relation; (ii) a rapid change in an actor's behavior has been triggered and thus, connecting links have to be updated to reflect this change in the model accordingly. In the first case (see Line 9), update cycles are extended up to $\overline{t_{u_{max}}^s}$ in order to optimize performance. Hence, for longer stable interaction behavior of actors, update intervals are increased.

However, if considerable sudden changes in behavior are detected (e.g., someone does not reply to requests anymore) (see Line 17), an immediate update is triggered and consecutive updates are performed in shorter intervals until stable behavior (trust levels) is detected again. If the amount of interactions drops below a given threshold ϑ_I^s, update intervals are increased to collect a sufficient number for reliable trust determination. However, if the update interval become too long ($> \overline{t_{u_{max}}^s}$), the previously described aging process is applied. Function `applyAging()` (Line 36) is implemented as Eq. 4.3 that continuously degrades trust links, and finally, removes

Algorithm 1 Social trust update algorithm executed every tick t_i.

```
/* access $G_I = (N, E_I)$ in interaction databases */
/* access $G_T = (N, E_T)$ in social trust model */
for each $u \in N'$ do
    for each $v \in N'$ do
        for each $s \in Scopes$(edge($E_T, u, v$)) do
            if $|E_I(u, v)| > \vartheta^s_I$ then                                ▷ enough interactions to reliably infer trust
                $e_\tau \leftarrow E_T(u, v)$
                if $\exists\ \tau^s \in e_\tau$ then
                    if isUpdateScheduled($e_\tau, s$) then                          ▷ previously scheduled update
                        $M^s(u, v) \leftarrow$ calcMetrics($E_I(u, v), s$)
                        $\widehat{\tau}^s(u, v) \leftarrow$ inferTrust($u, v, M^s(u, v)$)
                        $\overline{t^s_u} \leftarrow$ getUpdateInterval($e_\tau, s$)
                        if $\overline{t^s_u} \leq \overline{t^s_{u_{max}}}$ then
                            scheduleUpdate($e_\tau, s, \overline{t^s_u} + \overline{t_s}$)
                        else
                            scheduleUpdate($e_\tau, s, \overline{t^s_{u_{max}}}$)
                    else                                                             ▷ trigger changing behavior
                        $M^s_T(u, v) \leftarrow$ calcTriggers($E_I(u, v), s$)
                        if isUpdateTriggered($e_\tau, M^s_T(u, v)$) then
                            $M^s(u, v) \leftarrow$ calcMetrics($E_I(u, v), s$)
                            $\widehat{\tau}^s(u, v) \leftarrow$ inferTrust($u, v, M^s(u, v)$)
                            scheduleUpdate($e_\tau, s, \sigma_\tau, \overline{t^s_{u_{min}}}$)
                        else
                            $\widehat{\tau}^s(u, v) \leftarrow \widehat{\tau}^s_{i-1}(u, v)$                  ▷ stable behavior, no updates
                    $\tau^s_i(u, v) \leftarrow$ update($\tau^s_{i-1}(u, v), \widehat{\tau}^s(u, v)$)          ▷ smoothen trust values
                else                                                                 ▷ introduce new links
                    $M^s(u, v) \leftarrow$ calcMetrics($E_I(u, v), s$)
                    $\tau^s_i(u, v) \leftarrow$ setInitialTrust($M^s(u, v)$)
                    addLink($e_\tau, s, \tau^s_i$)
                    scheduleUpdate($e_\tau, s, \overline{t^s_{u_{min}}}$)
            else                                                                     ▷ if too few interactions
                $\overline{t^s_u} \leftarrow$ getUpdateInterval($e_\tau, s$)
                if $\overline{t^s_u} \leq \overline{t^s_{u_{max}}}$ then
                    scheduleUpdate($e_\tau, s, \overline{t^s_u} + \overline{t_s}$)               ▷ increase update intervals
                else
                    applyAging($e_\tau, s$)                                          ▷ age out existing relations
/* write back updated $G_T$ */
```

an existing edge from the graph model.

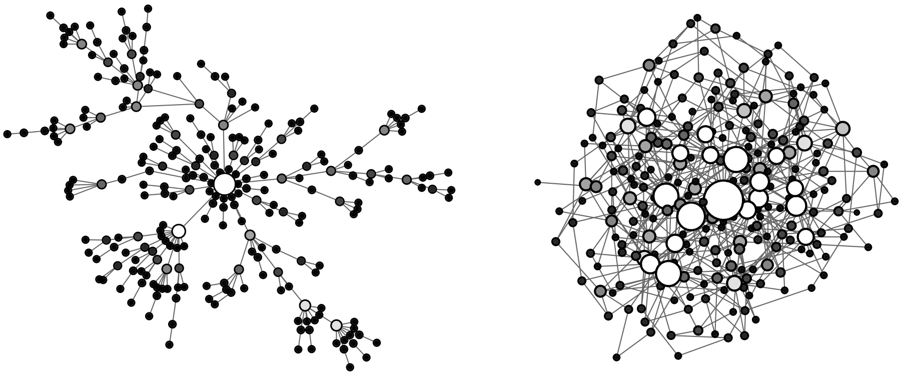
Algorithm 1 is periodically executed to keep G_T fresh. The execution interval needs to be adapted to the inherent dynamics of the environment. Since the algorithm processes interaction logs and relations only for a subset N' of all nodes, computational effort can be distributed over several instances that handle only parts of the whole network G_T.

4.4 Evaluation and Discussion

Since we have not yet applied our approach in real large-scale environments, we do not have sufficient real testing data. Therefore, we generate artificial scale-free network structures that we would expect to emerge under realistic conditions in typical collaboration networks [Reka and Barabási, 2002] to test and discuss our computational social trust model.

Experiment Setup

Collaboration Network Generation. We utilize the *preferential attachment model* of Barabasi and Albert [Reka and Barabási, 2002] to create graphs with power-law distributed degrees depicted in Figure 4.7. These network structures are the basis to generate interaction logs that follow a realistic distribution among members. For a graph $G = (N, E)$, we generate in total $100 \cdot |E|$ interactions between pairs of nodes (u, v). In our experiments we assume that 80% of interactions take place between 20% of the most active users (reflected by hub nodes with high degree). Generated interactions have a particular type (support request/response, activity success/failure notification) and timestamp, and occur in one of two abstract scopes. Through utilizing available interaction properties, we calculate three metrics (i) *availability* (amount of responded support requests), (ii) *interest similarity* (based on extracted tags from successfully finished activities), and (iii) *support reciprocity* (ratio of served to requested support). The actual strength (weight respectively) of a social trust relation is determined by combining and weight-



(a) hierarchical (avg deg=2).

(b) democratic (avg deg=4).

Figure 4.7: Generated scale-free networks for studying adaptive social trust models.

ing these metrics with a rule based approach (see [Skopik et al., 2010a] for details). For all experiments, we calculate the following interaction metrics:

Interest Similarity $isim$. This metric determines the overlap of actor interests, which is an important measure to find motivated partners in the same interest area. We manage keywords used by actors u and v as interest profile vectors $\mathbf{p}_u$ and $\mathbf{p}_v$ respectively (see previous chapter), and determine the similarity of profiles through the cosine between their profile vectors (Eq. 4.4). The result is a value between 0 (no overlap) and 1 (full overlap).

$$isim(u,v) = cos(\mathbf{p}_u, \mathbf{p}_v) = \frac{\mathbf{p}_u \cdot \mathbf{p}_v}{|\mathbf{p}_u||\mathbf{p}_v|} \tag{4.4}$$

Reciprocity $recpr$. A typical social behavior metric is reciprocity [Mui et al., 2002] that here reflects the ratio between obtained and provided support in a community. Let $REQ(u,v)$ be the set of u's sent support requests to v, and $RES(u,v)$ the set of u's provided responses to v's requests. Then we define reciprocity in $[-1,1]$ as in Eq. 4.5; hence, 0 reflects a balanced relation of mutual give and take.

$$recpr(u,v) = \frac{|RES(u,v)| - |REQ(u,v)|}{|RES(u,v)| + |REQ(u,v)|} \tag{4.5}$$

Availability $avail$. This metric describes u's availability for v's requests, i.e. the amount of answered requests. The result of Eq. 4.6 is a value in $[0,1]$.

$$avail(u,v) = 1 - \frac{|REQ(v,u)| - |RES(u,v)|}{|REQ(v,u)|} \tag{4.6}$$

Model Setup. As described earlier, the computational model infers social trust by interpreting various measured metrics; here: $isim$ and $recpr$. Changing interaction behavior is triggered by varying availability ($avail$) of actors regarding requests from other members in the network. This means that $avail$ is periodically sampled, while trust relations are updated based on $isim$ and $recpr$ only due to major changes of $avail$ (or the maximum update interval has been reached). We argue that these metrics are appropriate examples for reflecting reliable (i.e., trustworthy) behavior in typical activity-centric collaborations. In particular, for successful collaboration mutual interests are of importance, while also a cooperative behavior (expressed by support reciprocity) is highly rewarded. In contrast to that, in an emergency help and support environment (see [Skopik et al., 2010a]) fast and reliable response behavior is of paramount importance; thus, different metrics (responsiveness, success rate) denote trustworthy behavior there.

Effectiveness of Adaptive Update Strategy

We prove the advantages of selective and adaptive updates with several evaluations. For the following experiments, we set up a simulation environment as follows: We directly model different user behavior here to demonstrate the applicability of adaptive update intervals. In this round-based simulation the metrics $avail$, $recpr$, and $isim$ are modified for a fixed amount of actors. In particular, 5%, 10%, and 20% of (erratic) actors change in each round (with length $\overline{t_s}$)

respective metric values randomly between 1% and 50%. We introduce $G_R = (N, E_R)$ which is a graph reflecting the reality, and modify metrics assigned to its edges $e_r \in E_R$. Social trust is managed in $G_T = (N, E_T)$ and its edges $e_\tau \in E_T$ updated by Algorihtm 1 according to changing metrics in G_R (reflects basic behavior sampling). The main goal of adaptive updates, compared to periodic intervals, is the reduction of update cycles due to performance reasons. However, by delaying updates a deviation (Eq. 4.7) between G_T and G_R is introduced that has to be kept to a minimum.

$$dev(G_R, G_T) = \frac{\sum_{e \in E} |\tau(e_r) - \tau(e_\tau)|}{|E|} \tag{4.7}$$

The average deviation $dev(G_R, G_T)$ reflects the effectiveness of update models. The proposed update approach in Section 4.3 has several tuning parameters. Among the most important ones are the settings of minimum/maximum update intervals, configured as $\overline{t^s_{u_{min}}} \leq \overline{t^s_u} \leq \overline{t^s_{u_{max}}}$; whereas $\overline{t^s_{u_{min}}} = \lambda_1 \cdot \overline{t_s}$ and $\overline{t^s_{u_{max}}} = \lambda_2 \cdot \overline{t_s}$ and $\lambda_1 \leq \lambda_2$. Hence, λ_2 allows to extend the scheduled updates of stable relations up to $\overline{t^s_{u_{max}}}$ and thus, to significantly reduce computational effort.

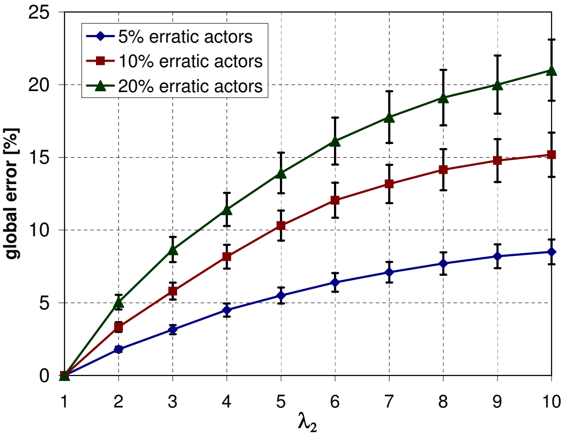


Figure 4.8: Deviation of trust values (global error) between simulated network and captured model for differently configured update strategies ($\lambda_1 = 1$).

The introduced *global error* due to adaptive updates (compared to fixed interval updates) is expressed as the average $dev(G_R, G_T)$ in percent. Figure 4.8 depicts this error for different λ_2. In this experiment, the behavior trigger mechanism (compare Line 17 in Algorithm 1) has been deactivated. Instead, we decrease $\overline{t^s_u}$ by one $\overline{t_s}$ after each update operation. Hence, the lengths of future update intervals directly depend on the lengths of recent update intervals, but are only moderately influenced by sudden behavior changes. It is demonstrated that even for small λ_2 considerable error rates are introduced. Since simulated behavior relies on various randomly changed metrics, error bars indicate the spread of results for multiple runs of this experiment. Although λ_1 determines $\overline{t^s_{u_{min}}}$, there is also an additional trigger mechanism that initiates immediate updates independent from $\overline{t^s_{u_{min}}}$ if actors change their behavior very quickly. With the trigger threshold ϑ_t the limit of tolerated behavior change without triggering an immediate update can be set. This threshold is defined as the deviation in percent of metric values in the interval $\overline{t_s}$. We trigger behavior changes by frequently observing the metric $avail$. With this trigger mechanisms, an upper limit of global error rate can be guaranteed, because rapid behavior changes (reflected in G_R) are detected and immediate updates of G_T performed. Hence, deviations are not added up over multiple sampling intervals (up to $\overline{t^s_{u_{max}}}$).

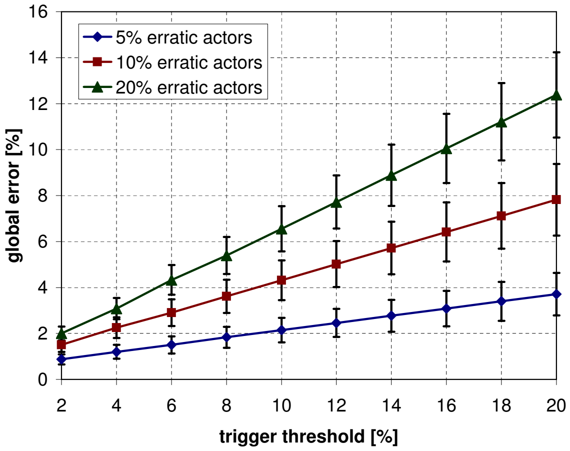


Figure 4.9: Deviation of trust values (global error) between simulated network and captured model for differently configured trigger thresholds ϑ_t ($\lambda_1 = 1$, $\lambda_2 = 5$).

Figure 4.9 visualizes that with the trigger mechanism in place, the global error rates can be considerably decreased. Typically, a higher number of erratic actors in the network still causes a higher average global error. The reason for that is a significant amount of actors who change their behavior slightly below the trigger threshold. Thus, a deviation of G_R to G_T is caused, but no updates triggered. However, setting a smaller λ_2 results in a smaller $\overline{t^s_{u_{max}}}$ and forces frequent updates; therefore, introduces an upper limit of global error rates over time. Since we have now demonstrated that we can keep the global error rate low, even when we apply adaptive updates (especially with a behavior change trigger in place), we demonstrate now the performance advantages. For that purpose, we utilize a generated graph G_R with $10\,000$ nodes and $20\,000$ edges (i.e., $d = 4$). In particular, we investigate the average amount of update operations per $\overline{t_s}$ for different λ_2. Higher λ_2 cause less frequent updates of relations. Note, updates of relations are not synchronous, i.e., all at the same point in time, but time instants are set for each edge individually in multiples of $\overline{t_s}$.

Theoretically, without adaptive updates and behavior triggers (i.e., $\lambda_1 = \lambda_2 = 1$), approximately $20\,000$ ($= |E|$) operations per $\overline{t_s}$ would be required to keep the example graph up-to-date with an error rate virtually equal to zero. However, since updates may be postponed until $\overline{t^s_{u_{max}}}$ if no rapid behavior changes are detected, the number of required update operations in G_T drops exponentially for higher λ_2, as shown in Figure 4.10. The dashed line visualizes the number of updated edges due to scheduled updates, even if actors do not change their behavior (then, all updates are performed in intervals of $\overline{t^s_{u_{max}}}$). The other lines show the upper limit of performed updates, i.e., the case that the set of relations with scheduled updates and relations with detected

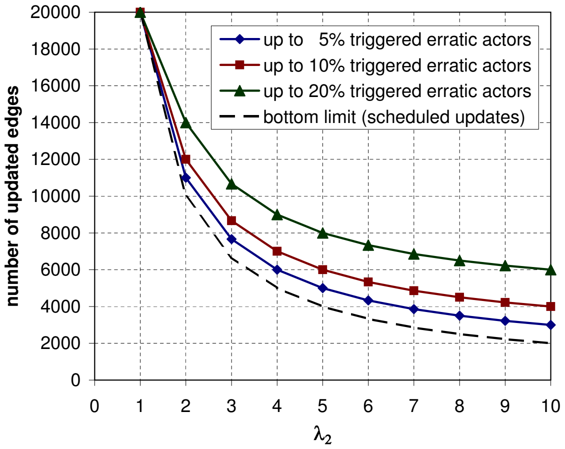


Figure 4.10: Number of processed edges after updating the trust model according to the simulated network ($|E| = 20\,000, \lambda_1 = 1$).

behavior changes do not overlap. Usually, the number of required updates is somewhere between these two limits.

Finally, Table 4.1 summarizes our results by comparing introduced global errors and average number of required update operations; thus, demonstrating potential savings.

λ_2	**global error [%]**	**average number of updates**
1	0	20 000
3	1.7	8 666
5	3.8	6 000
10	5.1	4 000

Table 4.1: Summary of evaluation ($\lambda_1 = 1$, $\vartheta_t = 10\%$, *amount of erratic actors*$=10\%$).

4.5 Conclusion

In this chapter we highlighted the application of the widely adopted MAPE approach for adaptations in complex interaction networks. Adaptation techniques, accounting for contextual constraints and emerging social relations (e.g., trust) are among the key research areas in flexible service-oriented collaboration environments. The evaluation of our model discovered important design issues, such as the mode of operation and configuration of dynamic update models. Our approach has important implications on adaptations in complex systems, because it reduces configuration burdens for the users and permits self-regulation of collaborations. As business is more and more performed online, the application of self-managed social network models, relying on collected interaction data, user actions, and personal profiles, in large-scale crowdsourcing environments seems to be of paramount importance in the future.

CHAPTER 5

Adaptive Provisioning of Human Expertise on the Web

Web-based collaborations have become essential in today's business environments. Due to the availability of various SOA frameworks, Web services emerged as the defacto technology to realize flexible compositions of services. While most existing work focuses on the discovery and composition of software based services, we highlight concepts for a people-centric Web. Knowledge-intensive environments clearly demand for provisioning of human expertise along with sharing of computing resources or business data through software-based services. To address these challenges, we introduce an adaptive approach allowing humans to provide their expertise through services using SOA standards, such as WSDL and SOAP. The seamless integration of humans in the SOA loop triggers numerous social implications, such as evolving expertise and drifting interests of human service providers. Here we propose a framework that is based on interaction monitoring techniques enabling adaptations in SOA-based socio-technical systems.

5.1 Introduction

The demand for models to support larger-scale flexible collaborations has led to an increasing research interest in adaptation techniques to enable and optimize interactions between collaboration partners. Such ecosystems comprising people and services that interact in different organizational units are difficult to model in a top-down manner. Challenges include, for example, changing interests and expertise of people, evolving interaction patterns due to dynamically changing roles of collaboration partners, or evolving community structures.

Web services enable loosely-coupled cross-organizational collaborations. In particular, they provide the means to specify well-defined interfaces and let customers and collaboration partners use an organization's resources through dedicated operations. However, offered resources are not restricted to information and software-based services. Also *human expertise* can be provided in a service-oriented manner. For that purpose, the Human-Provided Services (HPS) Framework [Schall et al., 2008b] enables human participation in a SOA environment. A typical example is a document translation service [Shahaf and Horvitz, 2010] that could be implemented in software too, but mostly only with insufficient quality. HPS allows humans to provide translation services in the same manner by letting them receive and process requests through Web service interfaces. With the human in the loop, traditional service-oriented architectures (SOA) transform from pure technical systems into socio-technical systems [Cherns, 1976]. These systems are characterized by both technical and human/social aspects that are tightly bound and interconnected. The technical aspects are very similar to traditional SOAs, including facilities to deploy, register and discover services, as well as to support flexible interactions. Additionally, the social system includes people and their habitual attitudes, values, behavioral styles and relationships. In particular, considering drifting interests of people, evolving skills, and varying collaboration incentives requires enhanced technical infrastructures in terms of flexibility and adaptability. Due to the support of loose coupling, sophisticated discovery mechanisms, and dynamic binding, Web services and SOA deem to be the ideal technical framework to realize large-scale socio-technical systems on the Web. We call the mix of software services and humans interacting on the Web a *Mixed Service-oriented System*.

The foundational pillars of such mixed systems are as follows: (i) *Human-Provided Services.* We discuss the HPS concept letting people participate in pure service-oriented environments. People reflect their ability and willingness to contribute by defining and offering their own services using state-of-the-art SOA techniques. (ii) *Flexible Interaction Models.* Interaction monitoring and mining is applied to determine the behavior of services and user relations. Detecting behavior and relation changes is the basis for effective service adaptations. (iii) *Adaptive Service Infrastructure.* We discuss the need for run-time adaptation. In particular, we do not only adapt service behavior, but also provided service features at run-time.

Approach Outline. Figure 5.1 depicts the overall approach to *flexible run-time provisioning of human expertise*. In the monitoring phase, service interactions, i.e, SOAP messages, and major system events, such as service updates, are captured. All interactions are annotated with tags and keywords to categorize requests. Then, this data is analyzed to learn about the service behavior in terms of reliability and dependability that is described by various interaction metrics [Skopik et al., 2010a]. Relations between clients and services are established based on these calculated ranking metrics. The actual analysis is context-aware, e.g., considers message tags to determine service behavior with respect to expertise areas. In the planning and adaptation phase respectively, services are rewarded and punished for their behavior. Capabilities influence the future provisioning of particular service operations. In the execution phase future service discovery and usage is influenced by adaptations to achieve optimal expertise provisioning in SOA.

Contributions. This work aims at addressing the following technical challenges found in mixed systems by applying Web services technologies and social network concepts:

- *Service Avatar.* This concept is used to represent human capabilities as services on the Web. A combination of WSDL and FOAF elements describe functional and non-functional properties.

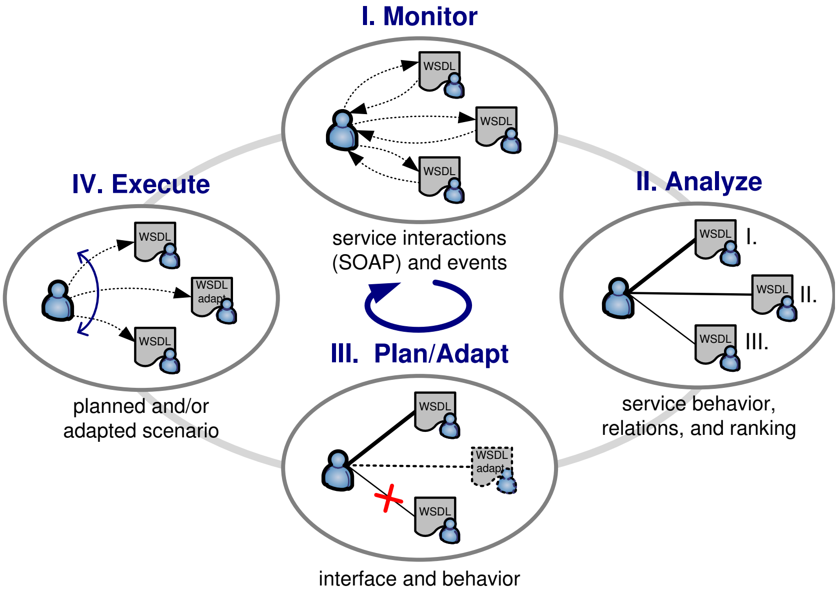


Figure 5.1: Adaptive run-time provisioning.

- *Personal Provisioning.* Social aspects require personalized service provisioning by establishing peer-to-peer relations between clients and service providers on demand.

- *Feedback-based Adaptation.* Observing and analyzing annotated SOAP interactions enable context-aware customization of personal services.

5.2 Human Involvement in SOA

We start with discussing the concept of avatars on the Web and the realization using Web service standards.

Avatars on the Web

Avatars are a computer user's representation of himself/herself, e.g., in form of a nickname or icon, in Internet communities. More advanced models further include interests and capabilities, such as in online gaming platforms. This makes an avatar the ideal metaphor to represent humans and their capabilities in service-oriented systems. Furthermore, an avatar does not only represent a human's services in an SOA environment, but can also actively act on behalf of the human it represents. Based on contextual constraints, such as the current load and assigned expertise areas, that software component can automatically categorize or reject requests. This process is configured through policies and rules in advance to shape the behavior of services and unburden the human from frequent but simple decisions. Figure 5.2 depicts the conceptual overview and explains our notion of avatar.

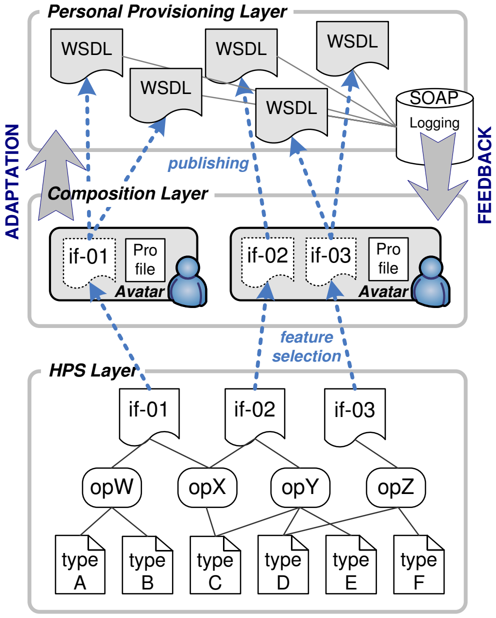


Figure 5.2: Conceptual framework for adaptive human expertise provisioning.

HPS Layer. HPS [Schall et al., 2008b] enhances the traditional SOA-based systems by enabling people to provide services with the very same technology as used by implementations of software-based services (SBS). Various operations for different collaborative activities indicate a provider's ability (and willingness) to participate in ad-hoc as well as process-centric collaborations. The HPS Framework provides predefined data types (XML schemas), operations, and compiled interfaces to provide particular services. The design of services is supported via a Web-based 'toolbox' (graphical user interface) enabling users to create services in a simplified manner. The creation of services does not require any knowledge related to SOA, Web services standards, or SOA runtime aspects. Based on the designed HPS, a script is parameterized to create and deploy corresponding avatars in the Genesis Hosting Environment (detailed in the following section).

Composition Layer. People who provide their expertise as services on the Web, select the required features, e.g., Web service interfaces to interact with clients, and compose them, thus, predefine the capabilities of the instances managed by their avatars. While these initial decisions represent the rather static properties of an avatar, personal profiles (modeled as FOAF [Brickley and Miller, 2010]) are periodically updated by our system to reflect social aspects, such as interests, interaction behavior and provided service quality. Together these static and dynamic properties characterize the avatar. The link between situation dependent profiles and composition decisions of the owner define the avatar's current providable instances. A high current load of the owner, for example, must not only update the current profile but also influence the avatars deployment strategy. Furthermore, to propagate the current situation to its instances the avatar provides instances with a connection to the current profile's state.

Personal Provisioning Layer. Clients discover avatars by accounting for (i) *functional prop-*

erties (FPs), i.e., the type of supported interfaces, and (ii) *non-functional properties (NFPs)*, i.e., social aspects. Here, **for each single client an own service instance is deployed** (peer-to-peer style). Clearly, humans providing services cannot serve thousands of concurrent requests as software services do. However, publishing dedicated instances enables our system to personalize them gradually for each individual client that has a long-term contract with the corresponding avatar.

On-demand Creation and Deployment

The concept of personalized provisioning is enabled by creating dedicated service instances for each single customer of service providers. A standard service is instantiated (derived from the avatar) and gradually customized according to a client's requirements and a provider's behavior.

```
def profile = profilePlgIn.connect(); //current profile
def Language=datatype.create("tconf.xsd","langTypeA") //imports
def Status=datatype.create("tconf.xsd","statType")
def i=callinterceptor.create() //interaction logging
i.hooks=[in:"RECEIVE", out :"PRE_STREAM"] //hooks on streams
i.code={ m -> ...} //logged message

def arrSrv=webservice.build {//interface definition
  // create web service
  TranslationService(binding:"doc,lit", namespace="http://...") {
    interceptors+=i //attach interceptor
    docQueue = [:] //current document queue
    repEP = "" //reporting endpoint
    // create translateDoc operation, return doc refId
    translateDoc(docref:String, fromLang:Language,
                toLang:Language, response:int) {
       def refId = genId(docQueue) //new id for doc
       //active pre-processing with checks
       if (profile.checkTotalLoad() < LOAD_THR)
          docQueue.put(refId,docref)
       return refId
    }
    getJobStatus(refId:int, response:Status){
       return report(refId)
    }
    cancelJob(refId:int){
       docQueue.remove(refId)
    }
    setAsyncReportEP(wsdl:String) {
       repEP = wsdl
    }
  }
}
def srv=arrSrv[0] // only one service declared, take it
def h=host.create("somehost:8181") // import back-end host
srv.deployAt(h) // deploy service at remote back-end host
```

Listing 5.1: Service deployment script.

Listing 5.1 displays a Groovy[1] script for the Genesis environment (G2) [Juszczyk and Dustdar, 2010] that allows to create a *Document Translation Service* with the approach shown in Figure 5.2. The first line connects the script content to the human's current profile via a G2-Plugin (`profile`). In the following two lines the script imports type definitions from the *HPS Layer* (`Language` and `Status` enumeration types). The service definition follows. An array (`arrSrv`) collects the services. In our case only the `TranslationService` is defined as follows. The queue `jobQueue` collects the current jobs of a service instance which resides in the *Personal Provisioning Layer*. Operation `translateDoc` checks the assignment in an exchangeable behavior closure[2], e.g., according to the current overall load of the human deter-

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

[2] A groovy closure is a reusable 'code block'.

mined by the profile. Then the document is moved into the input queue if all checks pass. The operation returns a unique `refId` which enables clients to manage their requests (e.g., request job status). The last three operations implement remote management. The first, `getJobStatus`, provides auto-generated job status reports. The second, `cancelJob`, allows the client to cancel an ongoing operation. The last one enables the client to set a callback endpoint for asynchronous human responses and notifications. The last three statements deploy the service in the G2 environment.

Listing 5.2 shows an excerpt of a document translation service WSDL, created with the script in Listing 5.1, that is provided by a human. Besides the `translateDoc` operation for submitting documents (and the omitted but mandatory `setAsynchReportEP` to define the reporting endpoint for notifying about finished jobs), there are further management operations, including `getJobStatus` and `cancelJob`. Complex data types, as shown for `Language`, are used to increase the semantics of the service description, e.g., by providing enumerations of available options.

```
<?xml version="1.0" encoding="utf-8"?>
<definitions ...>
<types>
 <schema elementFormDefault="qualified"
   targetNamespace="http://socsoa.infosys.tuwien.ac.at/">
   xmlns="http://www.w3.org/2001/XMLSchema">
 <element name="translateDocRequ">
  <complexType>
   <sequence>
    <element name="document" type="xsd:anyURI" />
    <element name="fromLang" type="Language" />
    <element name="toLang" type="Language" />
   </sequence>
  </complexType>
 </element>
 <simpleType name="Language">
  <restriction base="xsd:string">
   <enumeration value="German" />
   <enumeration value="English" />
  </restriction>
 </simpleType>
 ...
 </schema>
</types>
<message name="translateDocRequest">
 <part name="parameters" element="xsd1:translateDocRequ"/>
<message>
...
<portType name="TSPortType">
 <operation name="translateDoc">
  <input message="tns:translateDocRequest" Action=.../>
  <output message="tns:translateDocResponse" Action=.../>
 </operation>
 <operation name="getJobStatus"> ... </operation>
 <operation name="cancelJob"> ... </operation>
 ...
</portType>
<binding type="tns:TSPortType" name="..."> ... </binding>
<service name="TranslationService"> ... </service>
</definitions>
```

Listing 5.2: Document translation WSDL excerpt.

Traditional service development procedures are clearly insufficient in highly dynamic environments. As human capabilities evolve over time and interests or incentives for offering expertise change, provided operations of an HPS (and their signature) need to be adapted accordingly. For instance, a human providing a document translation service may learn a new language or discontinues the support of rarely requested options. Furthermore, some kind of support might be of low quality and/or not frequently used within a community. Another reason for changing a service's operations is the permanent delegation of responsibilities and balancing of features among a set of services.

Service Description and Discovery

An adaptive environment requires flexible service description and discovery mechanisms. Thus, before each request the client gathers dynamically compiled metadata on the current functional (FP) and non-functional properties (NFP) to update its view. This is realized by wrapping the HPS's WSDL file and an extended FOAF description into a *WS-Metadata-Exchange*[3] (MEX) document.

Basically there are two different reasons for initiating a discovery. First, for discovering an avatar. In that case FPs are of primary interest, combined with *capability metrics* that reflect the overall satisfaction of an avatar's clients. Second, before a request, the potentially adapted profile (including personalized metrics that reflect the avatar's behavior in the past) is retrieved. In both cases, our framework uses SPARQL[4] to define search queries on FOAF structures.

```
<mex:Metadata>
<mex:MetadataSection Dialect="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:definitions>
   <!-- Omitted -->
  </wsdl:definitions>
 </mex:MetadataSection>
 <mex:MetadataSection Dialect="http://xmlns.com/foaf/0.1/">
  <rdf:RDF xmlns:foaf = "http://..."
   xmlns:capability = "http://.../capability.owl#">
   <foaf:Person rdf:about="http://www.infosys.../staff/">
    <foaf:name>Harald Psaier</foaf:name>
    <foaf:interest rdf:resource="http://.../hpsaier/interests.rdf"/>
    <!-- Omitted -->
    <capability:op>
     <capability:port id="TSportType">
      <capability:op id="translateDoc">
      <capability:opwsdlxpath>
            wsdl:operation/[@name="TSportType"]
      </capability:opwsdlxpath>
      <capability:opmetricgrounding
            rdf:resource="http://.../grounding-translateDoc.xml"/>
      <capability:opmetric>
       <capability:opmetricid>cost</capability:opmetricid>
       <capability:opmetricvalue>100.0</capability:opmetricvalue>
      </capability:opmetric>
      <capability:opmetric>
       <capability:opmetricid>reliability</capability:opmetricid>
       <capability:opmetricvalue>0.8</capability:opmetricvalue>
      </capability:opmetric>
      ...
      </capability:op>
         </capability:port>
   </foaf:Person>
  </rdf:RDF>
 </mex:MetadataSection>
</mex:Metadata>
```

Listing 5.3: Dynamically created avatar description.

Listing 5.3 shows the sample response message to a MEX `GET` request. The main response body comprises the currently offered operations in a WSDL (omitted, see Listing 5.2) and the related NFPs in the second `MetadataSection` in FOAF format. The elements with the `capability` prefix provide the current NFP values for a related operation defined in the WSDL section. In our current implementation, such NFPs are costs and primarily quality

[3] WS-Metadata-Exchange: `http://www.w3.org/Submission/WS-MetadataExchange/`

[4] SPARQL: `http://www.w3.org/TR/rdf-sparql-query/`

metrics, such as an avatar's reliability and responsiveness. The *XPath statement* identifies an operation uniquely. The following metric grounding resource `opmetricgrounding` links a document with metric definitions (meaning, measurement, unit, range of values, etc.) to the listed metric ids. The description for those mined metrics is similar to [Object Management Group, 2008] for modeled QoS.

5.3 System and Service Adaptation

We discuss our approach to adaptive service provisioning by highlighting the fundamental building blocks, and in particular the adaptation of service instances itself.

Architectural Overview

Figure 5.3 shows an architectural overview of the whole framework that enables provisioning of human expertise.

The major components are organized in three layers:

- *Monitoring Layer.* SOAP *interactions* and environment *events* are logged and processed. Identified composite events are *triggered* and forwarded to the *adaptation module*.
- *Infrastructure Layer.* The adaptation module checks pre-defined rules to take appropriate steps, i.e., adapting the HPS templates in the HPS registry if a service does not provide sufficient QoS or adapting the deployed services in the *G2 hosting environment* [Juszczyk and Dustdar, 2010], e.g., removing unused or expired operations from a service instance and its WSDL interface (as shown later).
- *User Portal.* Users can discover potential services using the *discovery module*; and interact with particular instances through the *interaction module*. These interactions are logged to trigger future adaptations.

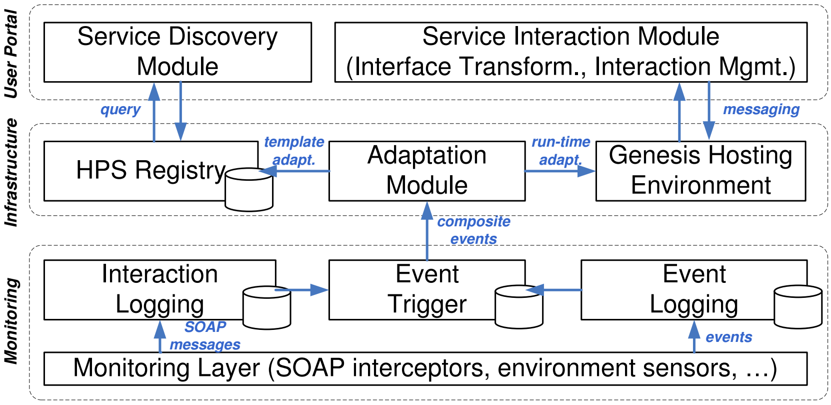


Figure 5.3: Architectural overview.

Parts of this system are described by references, for instance the G2 hosting environment [Juszczyk and Dustdar, 2010], and event triggering based on SOAP monitoring [Psaier et al., 2010b]. Therefore, in this chapter, we revisit the *interaction monitoring* concept in SOA environments from a technical point of view; deal with *service descriptions* in terms of functional and non-functional properties to support the discovery process; and demonstrate how to enable *run-time adaptations* in the Genesis hosting environment (see also [Psaier et al., 2010a]).

Interaction Monitoring and HPS Profiling

Interaction Model. Avatars are not statically bound to clients but are discovered at run-time. Thus, interactions are ad-hoc and dynamically performed with often not previously known partners. In SOA, interactions are typically modeled as SOAP messages. Besides standard SOAP structures we use various header extensions, such as WS-Addressing [Box et al., 2004], temporal properties (timestamps, deadlines), and contextual annotations. The latter are realized through tags/keywords that are assigned to messages to annotate interactions.

Our system utilizes temporal properties of SOAP calls to infer behavior metrics, such as the average time required to process a request, availability or responsiveness metrics (see [Psaier et al., 2010b; Skopik et al., 2010a] for details). As demonstrated in the previous chapters, metrics are calculated using the most recent history, and updated with a sliding window approach. Thus, old data ages out automatically. For the sake of simplicity, we only consider simple request-response patterns. A request can be accepted by a service and further processed by the corresponding avatar; or rejected immediately (e.g., due to the lack of free capacities). More complex, long-running interactions consisting of numerous intermediate responses are not in the scope of this work.

Dynamic Behavior Profiles. Since interests and skills of people regarding their capabilities to process requests from different domains usually widely vary, behavior metrics are context sensitive, i.e., bound to particular expertise areas. Collections of these behavior metrics are used to calculate NFPs and finally, to calculate service capabilities. For instance, someone may be highly rewarded for providing a document translation service while his/her document review service for scientific papers is not highly ranked. Moreover, the document translation service might be successfully used for research papers in computer science, while it is not frequently used to translate business documents. Human skills and expertise evolve over time. Furthermore, interests alter and drift. Thus, our monitoring and mining approach is the key to timely compensation of behavior changes.

Adaptation Strategies

Various reasons require timely adaptations of services that may affect the whole mixed service-oriented system. In particular, we study:

- *Client-driven interventions* are the means to protect customers from unreliable services. For example, services that miss deadlines or do not respond at all for a longer time are replaced by other more reliable services in future discovery operations.

- *Provider-driven interventions* are desired and initiated by the service owners to shield themselves from malicious clients. For instance, requests of clients performing a denial of service attack by sending multiple requests in relatively short intervals are blocked (instead of processed) by the service.

In general, adaptations can be less or more intrusive. We basically focus on two distinct mechanisms: (i) interface adaptations, and (ii) behavior adaptations. Interface adaptation means that single operations of a service are temporarily or permanently modified or removed. These changes can be triggered by the system due to request overloads or falling capabilities of services (that receive low ratings from clients). Behavior adaptations are more intrusive and alter the behavior of avatars respectively of their deployed service instances (though we cannot alter the behavior of humans it represents). The Genesis framework [Juszczyk and Dustdar, 2010] provides the ideal technical grounding to perform seamless run-time modifications, i.e., without being forced to take a service offline and redeploy it later again.

Client-driven interface adaptation example. Listing 5.4 demonstrates a typical adaptation desired by clients. In that case we assume that an avatar has missed deadlines several times. The system tries to protect the affected clients by automatically undeploying the `translateDoc` operation of the corresponding service instances (considered as 'lazy service'). Thus, clients can still retrieve the job status of ongoing translation requests, but are not able to send new ones. So, they are urged to discover alternative avatars or at least to negotiate a new contract with the same avatar (not shown here) who would deploy a new dedicated service instance.

```
def lazySrvArr=analysis.getLazyServices()
lazySrvArr.each { lazySrv ->
   webservice(name:lazySrv) { s-> name in s.name} { s->
     def o = s.getOperation("translateDoc") //get operation
     s.deleteOperation(o) //delete operation
     s.redeploy() //redeploy service and wsdl
   }
}
```

Listing 5.4: Adapt interface and undeploy operation.

Provider-driven behavior adaptation example. Listing 5.5 demonstrates a typical adaptation desired by providers. In that case we assume an avatar has highly varying working speeds. Thus, in case of request bursts (exceeding predefined thresholds `THR`), the system adapts the acceptance behavior regarding incoming requests before missing any deadlines. For instance, subsequent requests are rejected (or delegated to similar services [Skopik et al., 2010a]) instead of being queued for a longer time. The acceptance behavior is modified considering lower and upper queue size limits.

```
def busySrvArr=analysis.getBusyServices()
busySrvArr.each { busySrv ->
   webservice(name:busySrv) { s->
   if ("translateDoc" in s.operations.name && name in s.name)
      def op=s.operations.grep{o -> o.name == "translateDoc"}[0]
      op.behavior = {
           if ( (profile.checkTotalLoad() < LOAD_THR) &&
              (profile.checkRquFrq() < FRQ_THR)  &&
              (jobQueue.size() < QUEUE_THR) )
                 docQueue.put(refId,docref)
           ...
        }
      s.redeploy()
}{}
```

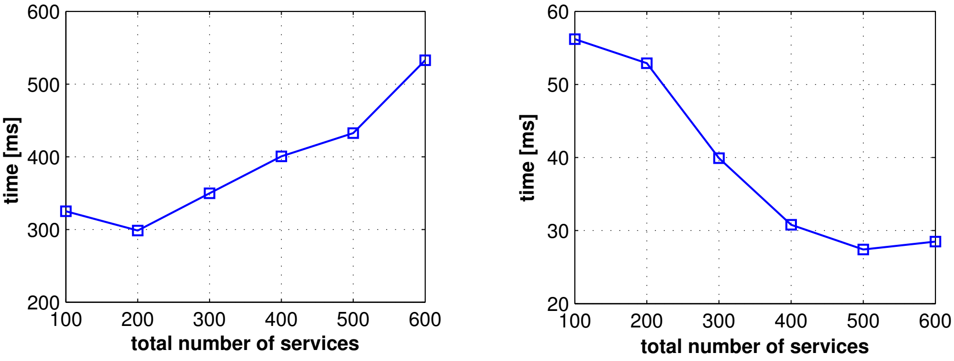
Listing 5.5: Adapt job acceptance behavior.

5.4 Evaluation and Discussion

We run a system evaluation in terms of performance and scalability, and functional properties, i.e., the power of the presented adaptation approach. For that purpose, we perform measurements directly in G2 which runs an round-based agent simulation.

Performance and Scalability

We evaluated the performance of our approach regarding the whole adaptation cycle based on this simulation environment. This loop includes logging of interactions, analyzing and inferring NFPs, evaluating pre-configured triggers (e.g., a service's reliability falls below a lower bound) and performing the actual adaptation. Costs for monitoring, analysis and triggering have been measured and discussed in detail in [Skopik et al., 2010a]. Thus, we focus on the actual adaptation in our flexible hosting environment.



(a) service interface update.

(b) service closure update.

Figure 5.4: Adaptation performance in G2.

We investigate both kinds of adaptation mechanisms, (i) interface adaptation, and (ii) behavior adaptation. Our testbed consists of variable amounts of deployed services (from 100 to 600), and we assume that an adaptation for 20% of the total number of instances is triggered. Figure 5.4(a) depicts the *average* time in milliseconds that is required to perform both, undeploying an operation of exactly *one* service and redeploying the modified WSDL interface. The scaleable adaptation approach of G2 aggregates change requests and performs modifications in bulks. Thus, for only 100 services in total (i.e., 20 modified instances) the average time is higher then for 200 services, but then rises nearly linearly. Figure 5.4(b) shows the required time for service behavior adaptations in the same environment. Here, the actual implementation of a single operation is exchanged. Note that again due to bulk modifications the *average* time for adapting one service instance decreases for higher amounts of services (approximately until 500 services). Further note that closure exchanges are approximately 10 times faster than interface adaptations which require a redeployment of the interface (but not of the decoupled underlying service instance).

Scenario Simulation

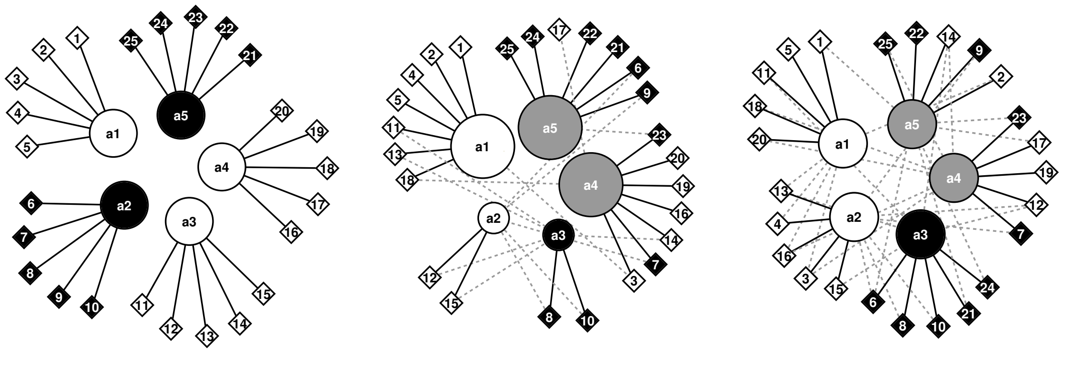
We run a small-scale simulation using the *Repast Simphony*[5] simulation toolkit to (i) test the implementation of our framework, and (ii) show the effects of different adaptation strategies. For that purpose we simulate human behavior in terms of reliability, expertise evolvement, unsteady working styles and interest drifts; and show the application of our adaptation approach.

Simulation Setup. The round-based simulation environment consists of 5 avatars (a_1 to a_5) and 25 clients that have different interests and behavior. Clients already have relations to avatars, i.e., there are dedicated service instances deployed for each client. Clients send one request every 10 rounds. An avatar needs between 1 and 2 rounds (random) to process that request. Thus, an avatar can serve an average of 7 concurrent clients. Client and service/avatar interactions are produced by simulated agents while for hosting the services, capturing and analyzing interactions, and performing adaptations our actual framework is utilized.

Experiment Setup. We distinguish between two distinct expertise areas, where each area is described by 10 different tags. Avatars have interest profiles consisting of 5 tags that reflect the types of requests that they are willing to process. Clients send requests that are annotated with up to 3 different tags. Initially each avatar has a clear profile, either in area *A* (white nodes), or area *B* (black nodes) – see Figure 5.5(a). Clients always send requests that match exactly one expertise area (white or black) and avatars accept these requests if they match more than 50% with their profile. In the initial state we have optimal conditions. No avatar is overloaded and they match exactly their clients requirements in terms of expertise.

Experiment Run. Major problems of human roles in technical systems are caused by people's drifting skills, evolving expertise and varying incentives and perception of risks. In short, people do not follow strict specifications such as software components do. In contrast, humans that provide services may change their focus of work. Our service provisioning system is able to tackle this problem by performing appropriate run-time adaptations.

We assume that avatars shift their interests and therefore, their expertise areas. However, the deployed services for long-term clients normally remain the same (e.g., consider a provider



(a) initial state. (b) final state (conservative adapt.). (c) final state (aggressive adapt.).

Figure 5.5: Evolving service community structures.

[5] Repast Simphony: `http://repast.sourceforge.net`

having multiple clients, but who specializes on translating documents in new domains). While a_1's interest profile remains unchanged, the other simulated avatars (agents) gradually change their profiles to reflect such natural interest shifts. Figure 5.5 visualizes the setup. The centered circles represent the 5 avatars and the diamond shaped symbols reflect a certain client's *static* requirements, manifested as a service instance that enables interactions between that client and the serving avatar. Comparing Figure 5.5(a) and 5.5(b) reveals that a_2 changes from the black to the white area, a_3 does exactly the opposite, and a_4 and a_5 extend their expertise areas to include both black *and* white (represented by gray nodes) expertise. As a consequence avatars refuse requests that no longer match their expertise areas. Furthermore, they register their new capabilities in the centralized service registry. Profile changes are linearly performed in the first 100 simulation rounds. Our system has additional 150 rounds to apply adaptations and to re-organize the network.

In particular, after interest shifts some avatars will not match their clients' requirements, and thus, begin to reject their requests. Our system will undeploy operations of corresponding services that are used to submit new requests and subsequently the whole service instance that connects a client and an avatar exclusively. This forces clients to query for new avatars that deploy new dedicated service instances to interact with their clients. Queries for new avatars account for matching profiles *and* previously reliable behavior (i.e., request success rate). Furthermore, based on changed interest profiles, the acceptance behavior of services is modified so that the serving avatar gets requests that match his/her new work area(s). Let us define the notion of *success rate* sr as the amount of successfully served requests in percent of one client-avatar relation, and *global success rate* gsr as the average of all single success rates. We use these metrics to measure the efficiency of applied adaptations. Note, success rates decrease if avatars attract too many clients and as a consequence become overloaded.

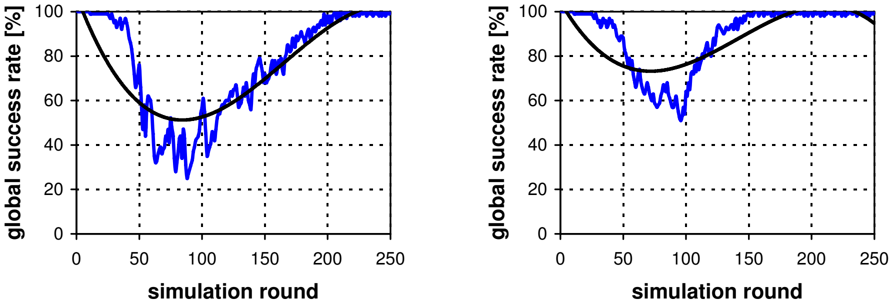
We demonstrate the impact of two fundamentally different adaptation strategies, (i) a conservative and even more tolerant strategy, and (ii) an aggressive strategy:

Conservative Strategy: The system collects multiple consecutive failures and violations of the clients interaction policies. In our simulation, clients are forced to stop interacting with an avatar if more than 5 requests are unreplied in less than 25 simulation rounds (interface adaptation due to dropping success rate). The client's memory has only a depth of 25 rounds. Thus, with that strategy, clients are considered more tolerant, they forgive short-time unreliability, e.g., caused by temporal work overloads of avatars and stay as long as possible with the same service provider.

Aggressive Adaptation Strategy: The system urges the clients to change their service providers (i.e., avatars) after the first triggered misbehavior of avatars. This strategy is more dynamic than the conservative one.

Experiment Results. Figure 5.5(b) visualizes the resulting network for the conservative strategy, and Figure 5.5(c) for the aggressive one. Solid lines represent active relations to avatars, while dashed lines visualize earlier relations. For instance, in both cases client 15 changed from using services from avatar a_3 to avatar a_2 according to their interest shifts. Obviously – and as expected – the aggressive approach triggers significantly larger numbers of adaptations compared to the conservative one.

Regarding the conservative strategy, clients change avatars rarely and only if the success



(a) gsr (conservative adapt.)

(b) gsr (aggressive adapt.)

Figure 5.6: Global success rate (and approximation).

rate of a serving avatar drops significantly and does not recover within 25 rounds. Thus, load on providers is unequally distributed (see Figure 5.5(b)). As shown in Figure 5.6(a), interest drifts cause longer adaptation cycles until the whole system returns to a steady state. The gsr sharply drops and begins to recover at round 100 (when all profile changes are finished). Note, the depth of the decrease highly depends on the tolerance of the system and clients towards unreliable avatars; i.e., if less failures and misbehavior are tolerated adaptations are triggered earlier which guarantees a higher global success rate. The aggressive approach requires much more system interventions, e.g., service adaptations and (re-)deployments. The costs (compare performance evaluation before) have to be considered with respect to the overall size of the environment. However, the sr can be kept much higher during the adaptation phase. Furthermore, the adaptation phase is shorter compared to a more relaxed adaptation approach, and a nearly equal distribution of load is reached (see Figure 5.5(c) where each avatar serves approximately the same amount of clients).

5.5 Conclusion

In this chapter we motivated the trend towards socio-technical systems in SOA. In such environments social implications must be handled properly. With the human user in the loop numerous concepts, including personalization, expertise involvement, drifting interests, and social dynamics become of paramount importance. Therefore, we discussed related Web standards and showed ways to extend them to fit the requirements of a people-centric Web. In particular, we outlined concepts that let people offer their expertise in a service-oriented manner and covered the deployment, discovery and selection of Human-Provided Services. Future challenges include more fine-grained monitoring and adaptation strategies. An example is the translation service presented in this chapter, where some language options are typically used more often, or even more successfully than others. In that case, data types could be modified to reduce the number of available language options in the WSDL interface description and to restrict input parameters. Harnessing delegation patterns that involve various participants, a complex social network perspective is established in which connections are also maintained among avatars. The later chapters of this thesis deal with this vision of a collaborative crowdsourcing environment.

CHAPTER 6

Bridging Socially-Enhanced Virtual Communities*

Interactions spanning multiple organizations have become an important aspect in today's collaboration landscape. Organizations create alliances to fulfill strategic objectives. The dynamic nature of collaborations increasingly demands for automated techniques and algorithms to support the creation of such alliances. Our approach bases on the recommendation of potential alliances by discovery of currently relevant competence sources and the support of semi-automatic formation. The environment is service-oriented comprising humans and software services with distinct capabilities. To mediate between previously separated groups and organizations, we introduce the *broker* concept that bridges disconnected networks. We present a dynamic broker discovery approach based on interaction mining techniques and trust metrics. We evaluate our approach by using simulations in real Web services' testbeds.

6.1 Introduction

The rapid advancement of ICT-enabled infrastructure has fundamentally changed how businesses and companies operate. Global markets and the requirement for rapid innovation demand for alliances between individual companies [Camarinha-Matos and Afsarmanesh, 2006]. Web services and service-oriented computing offer well established standards and techniques to model and implement interactions spanning multiple organizations. Collaborative service-based systems are typically knowledge intensive covering complex interactions between *people* and *software services*. 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

*Notice, additionally to the original research paper, research results presented here have also been published in another PhD thesis [Psaier, 2012], because this work was carried out as shared effort.

organizational boundaries: (i) the dynamic discovery and composition of resources and services, and (ii) 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. 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 [Kleinberg, 2008]). 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. We propose 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 [Burt, 2004]. Thus, such intermediaries can potentially *broker* information and aggregate ideas arising in different parts of a network [Kleinberg et al., 2008].

In this work, we present the following key contributions:

- We introduce *brokers* to establish connections between independent subgroups in professional virtual communities (PVCs). Our approach enables the dynamic selection of brokers based on changing interest profiles.
- We define metrics and their application to support the discovery and selection of brokers including *social trust* in service-oriented collaborations.
- Our approach is to introduce the *Broker Query and Discovery Language* (BQDL) 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 PVCs.

6.2 Emerging Virtual Communities

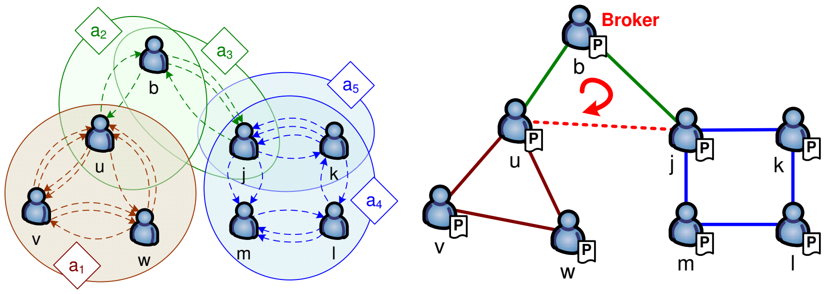
A PVC is a virtual community [Camarinha-Matos and Afsarmanesh, 2006] that consists of experts who interact and collaborate supported by ICT to perform their work. In today's systems, service-oriented technologies are increasingly used to realize PVCs. The support of loose coupling, sophisticated discovery, dynamic binding and various composition mechanisms make SOA the ideal technical grounding for Web-enabled PVCs.

Collaboration Scenario

Let us discuss an actual collaboration scenario in PVCs 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 6.1(a)). 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. Activities (e.g., see [Moran et al., 2005]) 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. PVC members use SOA technologies to interact in the context of ongoing activities. The HPS Framework [Schall et al., 2008b] allows human participation in a service-oriented manner. Humans can provide their capabilities and expertise as *services* to enable human interactions using standardized messages (i.e., SOAP). Interactions are logged for analysis. Relations emerge from interactions as illustrated in Figure 6.1(b), and are bound to particular scopes (*expertise areas*). The context in which interactions take place is based on tags applied to various artifacts exchanged between collaboration partners. Tags are used to combine similar activities to create scopes (i.e., boundaries of activities). In the given scenario, a scope comprises relations between PVC 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), we determine users' centers of interest. Frequently used keywords are stored in the actors' profiles (see symbol *P*) and later used to determine their interests and expertise areas. Second, we aggregate interactions that occurred in a pre-defined scope, calculate metrics (numerical values describing prior interaction behavior), and interpret them as *social trust* that is based on reliability, dependability and success.

Brokering and Compositions

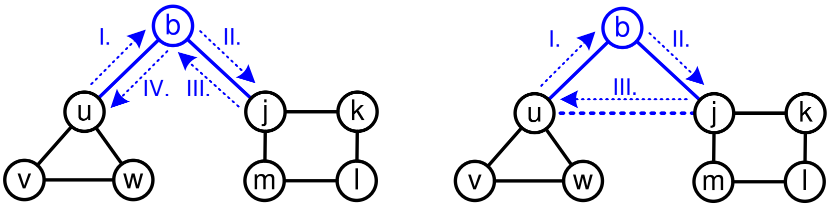
Consider a scenario in the given PVC in Figure 6.1(b). Suppose u wants to set up an activity that requires at least one additional expert from the *brown* $\{u, v, w\}$ and *blue* domain $\{j, k, l, m\}$. Since u personally knows v and w from previous collaborations, which is reflected by Friend-of-a-Friend (FOAF) [Brickley and Miller, 2010] `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



(a) Member interactions.

(b) Emerging relations.

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



(a) Persistent pattern.

(b) Triadic pattern.

Figure 6.2: Exogenous broker behavior patterns.

connected to j. Therefore, b could potentially act as a broker and forward requests or invitations to join u's current activity to j. We argue that establishing personal contacts in socially-oriented environments is of high importance 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: (i) *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**). (ii) *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. (iii) *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.

6.3 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 work, we define different types of brokers. 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 a collaboration. (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.

We argue 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 [Kleinberg et al., 2008]. 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 BQDL Specifications

Here we define the key elements of BQDL. 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, we present a set of BQDL queries along with their meaning considering a graph G and a set of subgraphs $G' \subseteq G$. We structure discussions related to a BQDL query into 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 6.3, consider two initially disconnected communities (sets of nodes) depicted as variables `var source` = $\{n_1, n_2, \ldots, n_i\}$ and `var target` = $\{n_j, n_{j+1}, \ldots, n_{j+m}\}$ residing in the graph G.

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 the target community $\{g, h, i\}$ (i.e., see `From` construct).

O1: The output of the query is (the example shown in Figure 6.3) 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 our graph representation, this means that each 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 our system. A specific type of metric (e.g., trust) is denoted by the

```
1  Input: Graph G, var source = {n_1, n_2, ..., n_i},
2         var target = {n_j, n_{j+1}, ..., 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
```

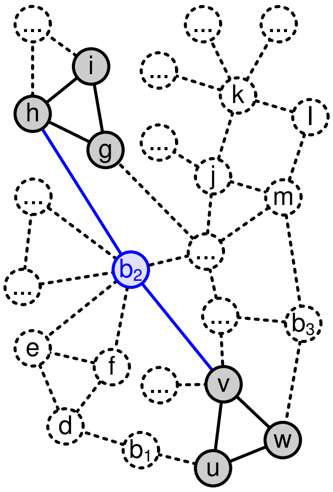


Figure 6.3: BQDL ex. 1: find broker to connect two predefined communities.

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 6.3 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 [Kleinberg et al., 2008]). The detailed mechanisms of this procedure are not the focus of this work.

Finding Communities

The broker discovery example in the previous section (Figure 6.3) is straightforward because 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 6.4 is to find sub-communities (or sub-graphs) 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 6.4 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

```
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
```

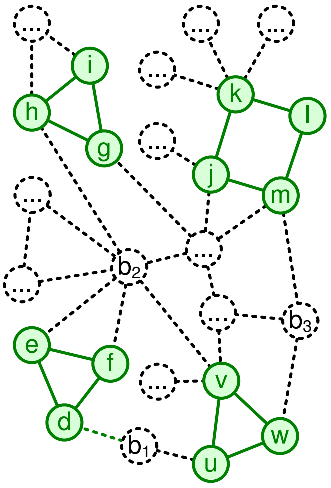


Figure 6.4: BQDL ex. 2: find ranked communities based on search criteria and metrics.

`to [1..*] 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 6.5 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 6.3). Second, communities are retrieved and stored in `SG1` (line 9). Again, this is based on the same principle as introduced previously in Figure 6.4. We call brokers connecting exactly one community *exclusive brokers*. 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.

```
1  Input: Graph G, var source = {n1, n2, ..., ni},
2         var search = {t1, t2, ..., tn}
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
```

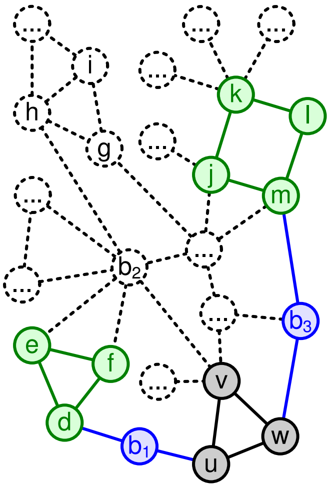
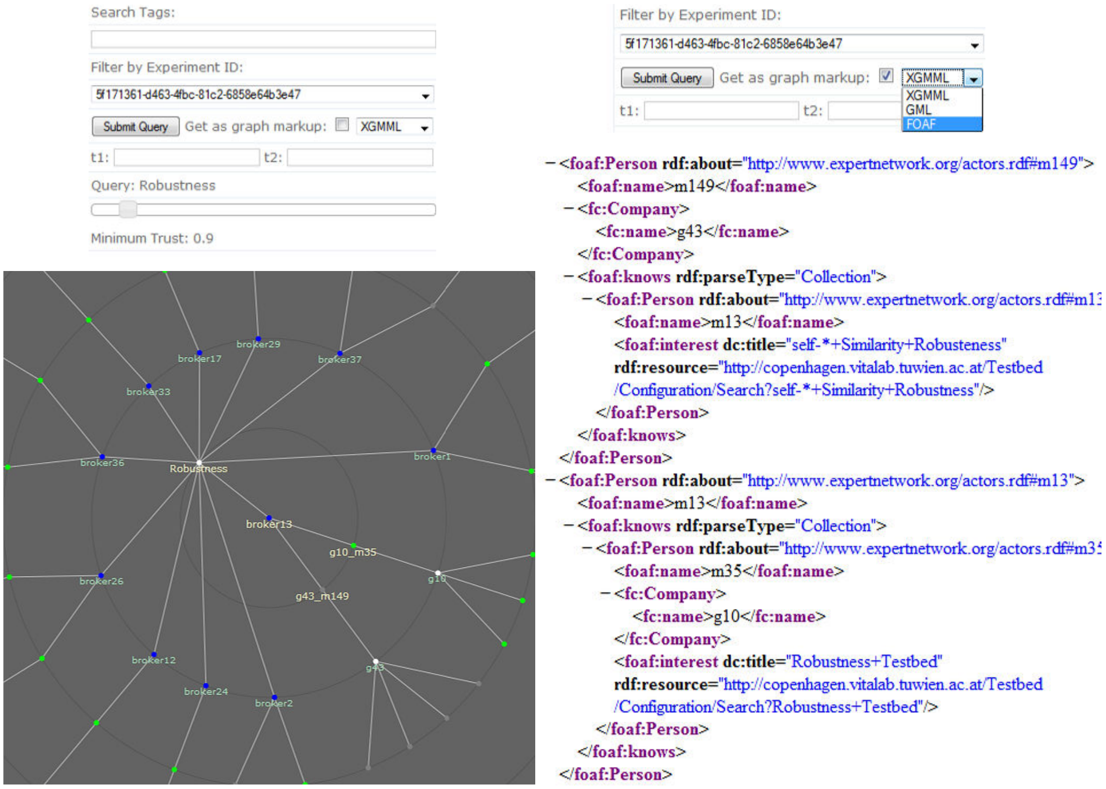


Figure 6.5: BQDL ex. 3: find exclusive brokers to connect two communities.



Search Tags:

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

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

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 6.6: Web-based broker discovery and network visualization tool.

6.5 Implementation and Discussion

The implementation of BQDL is part of our initiative to create a testing environment for socially-enhanced SOA. The environment consists of a Web service-based simulation environment using the Genesis2 [Juszczyk and Dustdar, 2010] framework and a middleware implementing user tools, logging, and eventing capabilities. Here we focus on tools assisting 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 6.6 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 PVC environment. The network view is obtained by mapping raw SOAP-interactions into a graph representation composed of nodes (services) and edges (interaction links). In our 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 6.6(a). 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 6.6(b)) that include `<foaf:interest>` tags.

SOA Testbed Environment

Our evaluations were gathered using the logging features of the Genesis2 framework [Juszczyk and Dustdar, 2010]. 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, we deployed Genesis2 Backends to the *Amazon Elastic Compute Cloud*[1]. We launched, 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. In the following, we provided each instance with the same Genesis2 Backend snapshot via mountable volumes from the *Elastic Block Store*. Finally, we deployed the following environment setup from a local Genesis2 Frontend. It included SOA-based PVCs 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 receivers capabilities. Therefore, they are tagged by three keywords one of which must match the picked receivers 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 [Macdonald et al., 2005] 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

We conducted several experiments to test the performance of our BQDL implementation under varying characteristics such as varying number of nodes and groups. The results are summarized in Figure 6.7. We simulated environments with different numbers of nodes and interactions to obtain insights in performance aspects. BQDL tools (Figure 6.6) and BQDL related graph libraries implemented in C# have been deployed on our local (lab-based) blade servers equipped

[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 for concurrent req. (in milliseconds).

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 average processing time.

Figure 6.7: BQDL processing statistics in simulated environment.

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 6.7(a)) is created on a separate machine (Intel Core2 Duo CPU 2.50 GHz, 4GB RAM) to perform parallel invocations 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 6.7(a). We vary the number of concurrent requests, denoted as RP, 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, we use RP=50 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. We increased the number of nodes and interactions 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 6.7 (experiments 2-4) we focus on a request pool with RP=50 and 100 requests to be processed by the BQDL service using different keywords (see Table 6.7(c)). To compare the experiments 1-4, we query the interaction graph using the keywords `Robustness Logging`. Increasing the number of nodes by a factor $\approx$ 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%. Our experiments show that BQDL scales with larger testbed environments linearly. Furthermore, we used different BQDL query keywords as shown in Table 6.7(c). The number of discovered brokers increases given mutliple keywords (see Table 6.7(b) for the set of available tags). The average BQDL processing time is not significantly influenced by the number of used keywords.

6.6 Conclusion

In this work we introduced the notion of *brokers* in socially-enhanced service-oriented environments. The idea of our broker approach is derived from theories found in social sciences (structural holes). Brokers can be modeled as Human-Provided Services to support the seamless integration of human capabilities in service-oriented infrastructures. The novelty of our approach is that brokers are not discovered based on static policies or static broker capabilities. In this work, we proposed the discovery of brokers based 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. Furthermore, we introduced 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. Furthermore, we discussed the implementation and performance aspects of BQDL.

CHAPTER 7

Managing Social Overlay Networks in Semantically-enriched Crowds

Semantic Web technologies are the basis to establish enterprise interoperability. Capabilities of services are semantically described and reasoning techniques support the discovery of services at run-time. In contrast to Semantic Web technologies that cover interactions between (technical) services, human collaborations emerge based on *social preferences*. Social networks are used to manage personal contacts and to share profile information with friends. These principles are increasingly harnessed in businesses environments. In a manner similar to service-oriented systems, they enable flexible discovery and dynamic collaborations between participants. Here, we discuss the concept of social overlays for Web service based collaboration infrastructures, which enable information flows between actors to allow for flexible group formations in crowd environments.

7.1 Introduction

The rapid advancement of ICT-enabled infrastructure has fundamentally changed how businesses and companies operate. Global markets and the requirement for rapid innovation demand for alliances between individual companies. Such alliances are created on different scales ranging from short- to long-term. A long-term alliance is typically a merger of companies or individual organizational units. Short- to mid-term alliances are commonly created to perform joint collaborations with the goal of fulfilling business objectives. Organizations have become *open enterprises systems* (OES) that offer capabilities as services. Capabilities can be discovered and composed to form new alliances. However, such systems do not only span automated interactions among (technical) services, but require humans actors to be in the loop. Today's Web applications facilitate interactive knowledge sharing, information exchange, user-centered content creation, and collaboration on the WWW. Even in business environments, *Web 2.0* tools increasingly provide users the free choice to interact or collaborate with each other in virtual communities. The Web becomes thereby a medium of interwoven human and service interactions. These principles have also changed models for computing on the Web by utilizing human manpower through crowdsourcing platforms (e.g., Amazon Mechanical Turk).

There are two obstacles hampering the establishment of seamless communications and collaborations across organizational boundaries: (i) the dynamic discovery and composition of resources and services, and (ii) flexible and context-aware interactions between people residing in different departments and companies. Here we address challenges related to human interactions in dynamic service-oriented systems. Semantic technologies and platforms [Berners-Lee et al., 2001] provide the means to automate the discovery and interactions of compositions. Semantically-enriched collaboration services provide the means for flexible interaction support. The technical composition layer of a service-oriented system (SOA) has received considerable attention in recent years from both the research community and industry. Considerably less attention was devoted to human aspects and interaction preferences in such systems. For example, people use services to perform collaborations.

We focus on *social aspects* in cross-organizational collaborations enabled by SOA. In order to take advantage of social preferences, we propose social network principles to overcome limited information flows in collaborative environments. Social interactions between network members allows to influence and control information flows.

In this work we address challenges related to the automated management of social network based on interactions in cross-organizational collaborations.

- Top-down composition and interaction models are typically designed for long-term use. Dynamic environments that are short- to medium-lived such as open enterprise systems require *dynamic interaction models*. Flexible interactions with the purpose of communicating, coordinating, and collaborating need to be supported in a service-oriented manner.
- Theories found in social network analysis are promising candidate techniques to support flexible interactions. Since interactions take place dynamically, capturing the purpose and context of interactions to infer meaningful social relations remains challenging.
- Social network principles such as formation algorithms help to overcome limited information exchange in separated collaborative networks through propagation of profile data. From the technical point of view, adaptive information flows need to be supported using services technology. Information needs to be discovered and exchanged based on the underlying social network.

The Semantic Web and related technologies have made important contributions to pave the way towards the effective interoperability of enterprise systems and infrastructures. Due to the proliferation of Web 2.0 collaboration principles and Semantic Web technologies, a combination of these approaches seems to be promising to create novel cross-enterprise collaboration systems. In the following we give an outline of our approach.

Figure 7.1 illustrates the fundamental motivation of applying *and* combining Semantic Web methodologies with Web 2.0 concepts. We show two main building blocks (i) *Enterprise Interoperability* and (ii) *Enterprise Collaboration* to support a seamless service-oriented infrastructure for cross-organizational collaboration in open enterprise systems. The *Semantic Web*

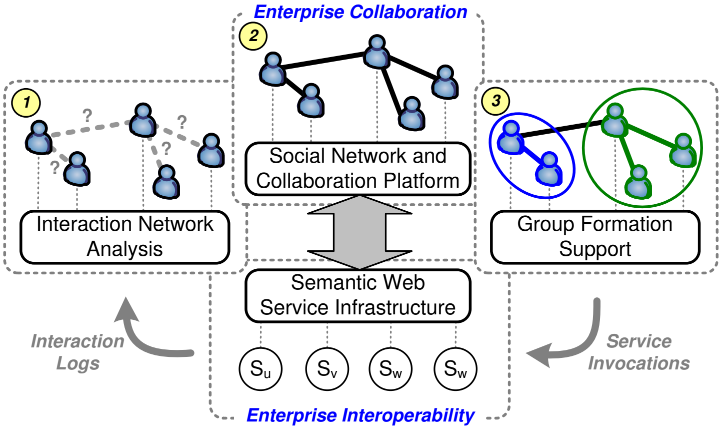

Enterprise Collaboration
2
Social Network and Collaboration Platform
1
?
Interaction Network Analysis
3
Group Formation Support
Semantic Web Service Infrastructure
Interaction Logs
Service Invocations
S_u
S_v
S_w
S_w
Enterprise Interoperability

Figure 7.1: Enterprise collaboration and interoperability through social and Semantic Web techniques.

Service Infrastructure provides the means to enable efficient and dynamic interactions spanning humans that belong to different organizational units. Underneath, Web services build an abstraction mechanism for intra-organizational infrastructures and resources and therefore, are the ideal technical grounding to enable interactions across organizational boundaries. Observing interactions and collecting collaboration data (*Interaction Network Analysis*) helps to support humans in building up new relationships by recommending new partners or notifying about possibly interesting business opportunities. A *Social Network and Collaboration Platform* allows people to manage their personal contacts and interact with well-known collaboration partners in context of certain projects. *Group Formation Support* concepts applied in collaborative networks allow actors to discover unconnected members using profile information, to build alliances, and to dynamically establish reliable information flows in order to exchange profiles.

In this chapter we deal with:

- *Cross-Organizational Application Model.* Cross-organizational scenarios are supported considering social aspects of interacting humans on the Web and technological interoperability using Semantic Web concepts.
- *Group Formation.* Formation is typically based upon sophisticated member discovery techniques. Thus, enabling actors to share personal profiles and information in a trustworthy manner is a key concept of our work. We discuss a social trust based access control (TBAC) mechanism that accounts for dynamically changing trust relations.
- *Specification and Implementation.* We discuss the implementation of social overlay networks using today's Web technologies, including Semantic Web services, interaction mining techniques, public key infrastructures, and the Friend-Of-A-Friend (FOAF) ontology.
- *Evaluation and Discussion.* We evaluate proposed models and their application in virtual communities, and derive general findings for designing applications for socially-enhanced service-oriented environments.

7.2 Social Overlays in Semantic SOA

Enterprise collaboration and interoperability services are going to become an invisible, pervasive, and self-adaptive knowledge and business utility for any industrial sector and domain. The goal is to enable rapid set-up, efficient management and effective operation of different forms of business collaborations, from the most traditionally supply chains to the most advanced and dynamic business ecosystems. Figure 7.2 shows an overview of our layered approach to enable reliable and flexible formation of collaboration groups: (i) the *Service Layer* provides the technical infrastructure to semantically describe and host Web services in order to enable cross-organizational collaborations; (ii) the *Interaction Layer* provides the means of Web service-based human interactions; e.g., allows actors to communicate and collaborate with others using dedicated services from the bottom layer; (iii) the *Monitoring Layer*, observes interactions collected from various sources (i.e., interaction services); and (iv) the *Discovery Layer* discovers social relations gathered through mining of interactions and profile properties, and supports group formation based on evaluating network links.

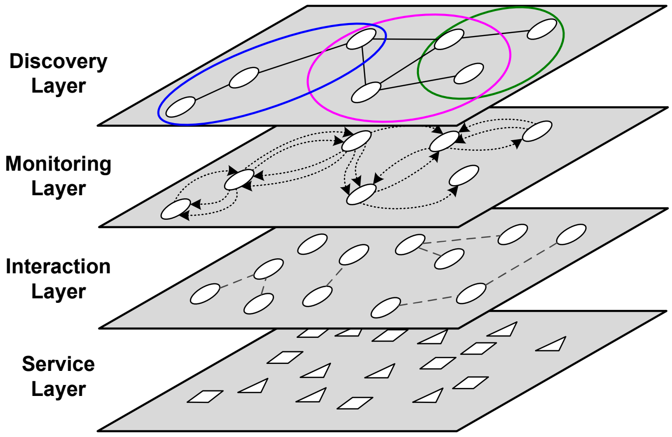


Figure 7.2: Model for social overlay networks.

Semantic Web Service Infrastructure

In order to realize the vision of cross-organizational collaboration and interoperability, various multi-national research projects, such as within the EU Seventh Framework Program[1], are conducted. The COIN project[2], where our contributions of this chapter are embedded, aims at developing a basic platform for future Web based cross-organizational collaborations. In the following, we discuss the architectural model of semantically-enriched social OESs and outline utilized major concepts on each layer.

The COIN project aims at providing an open, self-adaptive integrative solution for *Enterprise Interoperability* and *Enterprise Collaboration*. Service orientation is a well-suited and

[1] EU FP7: `http://cordis.europa.eu/fp7`

[2] COIN project: `http://www.coin-ip.eu`

widely adopted concept in collaboration scenarios, therefore, COIN utilizes state of the art SOA concepts, including Semantic Web technologies and Software-as-a-Service (SaaS) models (see [Gold et al., 2004] for more details). With respect to Enterprise Collaboration, COIN supports numerous features that focus on product development, production planning and manufacturing, and project management in networks of enterprises. As a fundamental aspect, human interactions exist in all forms and phases of virtual organizations and play a major role in the success of collaborations within open enterprise networks. Therefore, understanding human interactions and providing advanced support for efficient and effective interactions, is one of the key objectives in COIN's Enterprise Collaboration research track.

The COIN Framework consists of (i) the Social Network and Collaboration Platform (SCP) that provides fundamental features that are required in (nearly) every collaboration scenario, and (ii) a Semantic Web Service Infrastructure (SSI) that allows extensions with services following the SaaS model from third party providers. The SCP is designed for and tightly coupled to a community portal that provides an effective way to configure and personalize the SCP for specific end-users by providing customized services and tools. Single sign-on- and security mechanisms span services and tools across layers. The SSI relies on Semantic Web technologies, implemented by the Web Service Modeling eXecution environment (WSMX)[3] [Haller et al., 2005] and is utilized to discover, bind, compose, and use third-party services at run time. Because of its extensibility and configurability, the COIN platform can be applied in a wide variety of different collaboration scenarios, ranging from traditional production planning to social campaigning and interest group formations in professional virtual communities. For enabling context-aware interactions, the following baseline components are of major interest (i) user data, including skills and interest profiles, (ii) context data, such as current ongoing activities and user preferences, (iii) integrated baseline services for communication and coordination (e.g., e-mail notifications, and instant messengers), (iv) the SCP as the platform to host extended human interaction services.

Human Interaction Layer

Open enterprise systems that allow to form virtual organizations pose additional challenges to human interaction support. Typically such virtual organizations are temporary alliances that form and dissolve again. Various actors from different physical organizations are involved collaborating and working on joint activities. Figure 7.3 shows a semantic representation (i.e., an ontology) of utilized concepts, grouped in communication, coordination and collaboration entities.

Various artifacts need to be created in order to integrate common WSDL-based Web services into the Semantic Web infrastructure of WSMX [Zaremba and Vitvar, 2008]. We provide a basic description that acts as the underlying basis for the rest of this chapter as follows:

- *Enterprise Collaboration Ontology:* A collection of predefined semantic concepts establishes data interoperability through transformation, mediation, and reasoning. As depicted by Figure 7.3 the basic enterprise collaboration entities and their relations are well defined in a baseline ontology.

[3] Web Service Modeling eXecution environment: `http://www.wsmx.org`

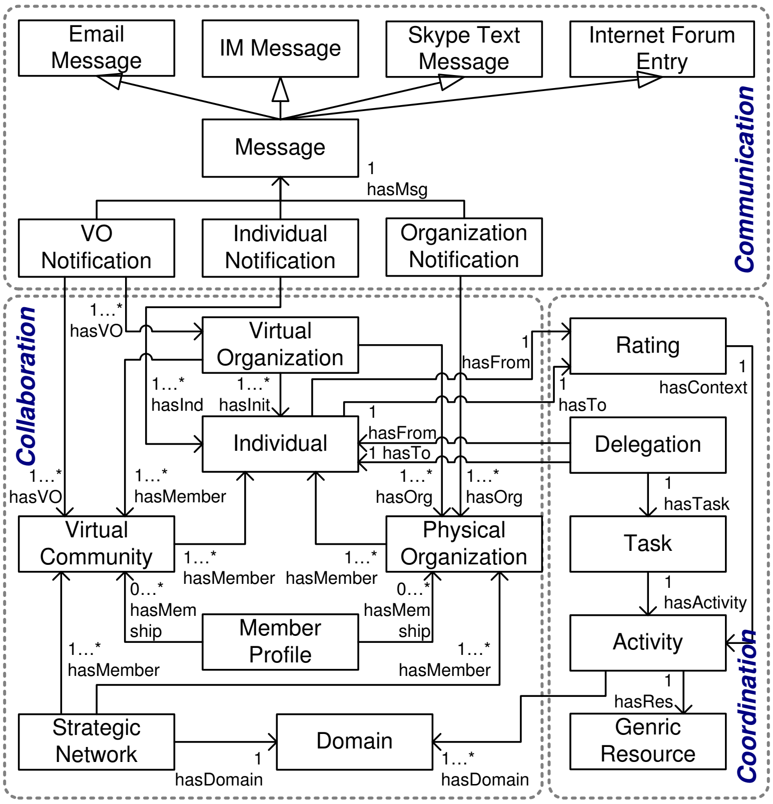


Figure 7.3: Enterprise collaboration ontology.

- *Semantic Goals:* A client specifies the objective to be achieved in terms of a goal [Stollberg and Norton, 2007], and the system resolves this by automated detection, composition, and execution of Web services. This concept allows dynamic discovery based on functional as well as non-functional properties, and advanced composability of services and service instances respectively.

- *Grounding Descriptions:* Since WSMX functionalities operate on semantic descriptions of messages, non-semantic messages require transformations to semantic representations and vice versa (i.e., lowering and lifting scripts).

- *Semantically-enriched WSDL Interface:* Data types used by Web service interfaces (WSDL) need to be linked to corresponding grounding scripts that mediate data between standard SOAP messages and semantic goals (RDF).

We utilize Semantic Web technologies to cope with inherent dynamics of open enterprise systems and to keep the environment manageable. In particular, we use the WSMX [Haller et al., 2005] platform to enable:

- *Cross-Organizational Abstraction.* Since members from various domains and organizations need to interact, we use Semantic Web Services as an abstraction from organizational structures in order to distribute communication facilities. Typically members of virtual communities use their organizations' resources and infrastructure; Web services resolve the need (semantic goal) of interaction to actual SOAP requests and additionally mediate between differing ontological concepts.

- *Context-aware Interaction Channel Selection.* Selecting appropriate communication, coordination, and collaboration service does not only depend on functional needs, but also on contextual constraints. For instance, the delivery of a message (described by a semantic goal) can be achieved through e-mail services, instant messaging, or postings in Internet forums. The appropriate channel can be selected based on user data (location, privacy rules) and messages (priority, size).

Monitoring Layer

As discussed in detail in the last chapters, interactions are observed and collected to determine social relations. We designed the system to manage relations by evaluating occurring interactions and therefore, unburden network participants – at least partly – from managing their relations manually. Logging invocations of collaboration services is the basis for advanced interaction analysis, and allows to infer social relations that are described by objectively measured metrics, such as average response times, availability, or reciprocity.

Formally, a virtual community is a special kind of social network $G = (N, E)$, where the single actors participate to perform activities. A community is modeled – as defined before – as a directed graph, where nodes N represent the actors that are connected through edges E. A directed edge from actor u to v is denoted as e_{uv}. Furthermore, activities A are a fundamental part of our model, where an activity $a \in A$ is used to include a set of participants. Thus, in short, activities describe the collaboration boundaries. Network members interact in scope of particular activities (i.e., to reach certain goals). Interactions are collected to determine (i) the center of interest of single network members by evaluating the frequency of used keywords [Schall and Dustdar, 2010; Skopik et al., 2010a], and (ii) the strength of a social relation by determining the similarity of the center of interests [Skopik et al., 2010f]. Since these techniques have been extensively discussed in previous work, we do not present a detailed description here.

Discovering Relevant Social Networks

In our framework, an actor has several *passive* links, modeled as FOAF relations, that express business/personal contacts (typically emerged from previous collaborations), but not describing that interactions are performed along these links. An actor can *activate* these links by initiating a new collaboration, e.g., setting up a joint activity. However, due to resource constraints, members can only participate in a limited amount of concurrent activities, and thus, the number of simultaneously active links is limited. Hence, collaboration partners are discovered and selected carefully, considering required effort and received benefit. Direct relations are established to create a typical social network. Since single members usually build up strong relations to only a small amount of partners, reliable information flows through collaborative networks, such as exchanging expertise and interest profiles, are limited. Thus, the discovery layer allows actors to exchange business contacts by sharing and propagating (parts of) profiles over intermediate nodes. Each actor's connectivity to other community members is determined by issuing keyword-based queries [Schall and Skopik, 2010] denoted by the query context Q. The query context is described by a pool of keywords (e.g., describing certain expertise areas) picked from global taxonomies. Using logged interaction data (and additional manual ratings) the link weight from one actor to another is calculated using a *social trust* metric that is discussed in detail in the next section.

7.3 Social Network Management

This section discusses a framework enabling distributed profile management in large-scale Web-based open enterprise systems. Profiles are shared among members and evaluated to discover potential collaboration opportunities based on interest similarities, coverage of expertise needs, project participation, and organizational memberships.

Architectural Overview and Design

Since information sharing with mostly unknown individuals in large-scale environments is a delicate matter, our framework applies common security standards to encrypt sensitive information and therefore, enables selective sharing of information. We adopt one of the most popular encryption concepts, in particular *public key infrastructure* (PKI) [Adams and Lloyd, 1999] for that purpose.

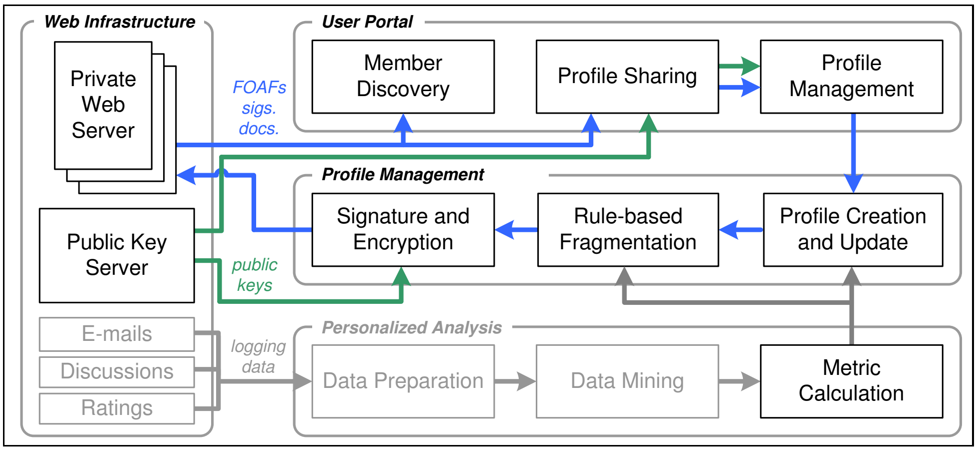


Figure 7.4: Architecture supporting discovery in self-managed social networks of open enterprise systems.

The fundamental architecture of our framework is depicted in Figure 7.4. Basically, the left side consists of globally available components, such as various Web servers owned by individuals and organizations, public key servers, and collaboration tools hosted in a semantic Web services environment, including e-mail infrastructures, discussions forums, and rating platforms. The right side comprises distributed components that are replicated for each user (and groups of users forming closed communities respectively) to manage their profiles from their personal point of view. The architecture consists of the following three layers: (i) *Personalized Analysis* enables data aggregation from collaboration tools and data mining to determine collaboration relations. Basically, the strength of social relations is inferred by calculated various interaction and behavior metrics from mining e-mail data or Internet forum entries [Schall and Dustdar, 2010; Skopik et al., 2010a]. (ii) *Profile Management* includes features to semi-automatically create and update FOAF profiles with calculated metrics. Profiles are encrypted and valid signatures created so that only close collaboration partners can decrypt and use them for discovering actors. (iii) the *User Portal* hosts tools to discover potential partners and sharing and managing personal profiles.

Emergence of Social Relations and Trust

We argue that trust and reputation mechanisms are key to the success of open dynamic service-oriented environments. However, trust is emerging based on evidence, i.e., interaction behavior. Interactions, for example, may be categorized in terms of success (e.g., failed or finished) and importance. Therefore, a key aspect of our approach is the monitoring and analysis of interactions to automatically determine trust. We argue that in large-scale SOA-based systems, only automatic trust determination is feasible. In particular, manually assigned ratings are time-intensive and suffer from several drawbacks, such as unfairness, discrimination or low incentives for humans to provide trust ratings.

We employ the usual definition of social trust as explained before. In particular, we focus on metrics describing interest similarity $isim$ and reciprocity $recpr$ (see Chapter 4). In contrast to more sophisticated rule-based approaches (see [Skopik, 2010]) that interpret metrics in terms of trust and its scope, here our focus is on social overlay networks enabled through semantic technologies. Hence, a simple arithmetic weighting[4] is feasible to create the underlying trust network and to demonstrate these overlay concepts. The actual strength (weight w respectively) of a social trust relation is determined by normalizing, combining and weighting ism and $recpr$ metrics whenever a discovery process is started, i.e., a query issued. While $isim$ is a globally valid metric, $recpr$ is bound to distinct contexts Q (e.g., expertise areas). In particular, interactions bound to all activities whose description match at least one of the query keywords issued for discovering neighbor nodes are considered when calculating $recpr$. Currently we employ flat keyword-based matching only, however for more advanced ontology matching techniques see [Castano et al., 2006; Euzenat and Shvaiko, 2007]. Eq. 7.1 allows for the balancing between two cases: (i) *newcomer support* versus (ii) weighting of links of *well established* actors (based on evidence). The factor α can be adjusted based on the requirements for each case. For example, by setting $\alpha = 1$, newcomer support becomes more dominant since $isim$ accounts for interest (profile) similarities. Whereas, the other case with $\alpha = 0$ puts stronger emphasis on already established links by accounting for the preference towards existing relations.

[4] Usually, we denote trust as τ. However, here we use only the notion of weight w which reflects the strength of a relation (i.e., degree of coupling between two nodes) determined through an arithmetic composition of metrics. In other words, w has not such a deep semantic meaning compared to τ.

$$w^Q(u,v) = \alpha \cdot isim(u,v) + (1-\alpha) \cdot recpr^Q(u,v) \tag{7.1}$$

TBAC - Trust based Access Control

Trust Bases Access Control (TBAC) (as similarly introduced in [Tran et al., 2005]) supports the discovery of collaboration partners and subsequently the formation of groups and networks in open enterprise systems using distributed profile information. The main idea is to allow actors to access the profiles of other network members based on the strength of social relations, e.g., social trust. In other words, only trustworthy partners are allowed to access, in particular read, someone's personal profile information. Key principles of the proposed approach are:

- *Self-managed Distributed Profiles.* Actors manage their personal profiles in a distributed manner, i.e., profiles are fully under control of the respective actors.

- *Public and Private Scopes.* Some profile information may be available public, for instance, expertise area and basic contact details in order to discover new collaboration partners. However, access to sensitive information, e.g., private contact details and friend relations, is restricted.

- *Social Trust-based Access Control.* Access to private fragments of profiles is granted based on strengths of social relations. For instance, close collaboration partners can read larger parts of an actor's profile. Social trust relies on interactions and an update of personal relations can be triggered by actors using logged information from the SOA infrastructure. Note, only logged interactions with personal involvement are used.

- *Public Key Infrastructure (PKI).* PKI is the means to enable public and private profile scopes and to address privacy concerns in open distributed environments.

Transitive Access. As in the real world, information is not only shared between direct neighbors, but can traverse several intermediate nodes. Using this approach allows sharing of profiles along trusted paths even if actors are not directly connected in the social network. This spreading of information relies on the principle of recommendation and propagation of trust respectively [Guha et al., 2004]. Since all involved parties are connected with a strong trust path, privacy is still maintained. Transitive access is an important concept to overcome inherent limitations of trust based discovery only.

7.4 Implementation Details

This section deals with the specification and implementation of the proposed social overlay model to realize dynamic discovery in semantically-enriched collaborative open enterprise systems.

Adaptive Distributed Profile Management

The mainly applied techniques are the Friend-Of-A-Friend (FOAF) [Brickley and Miller, 2010] ontology, Public Key Infrastructure, in particular GnuPG[5], and Web-Of-Trust (WoT)[6] schemas.

Friend-Of-A-Friend Profile Management

Various concepts and protocols have been proposed to manage open social and collaborative networks in a distributed manner. The Friend-Of-A-Friend (FOAF) concept is one of the most popular ones on the Web. It allows to model user properties, interests and relations with a well-known ontology. We apply FOAF to facilitate the discovery process used to find potential collaboration partners; for instance, u discovers v and w and acts as a mediator because of common interests, projects, and expertise, and is thus an appropriate partner to link network members.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:foaf="http://xmlns.com/foaf/0.1/"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:wot="http://xmlns.com/wot/0.1/"
<foaf:Person rdf:ID="me">
 <foaf:name>Florian Skopik</foaf:name>
 <foaf:nick>florian</foaf:nick>
 <foaf:mbox_sha1sum>a4b378...</foaf:mbox_sha1sum>
 <wot:haskey rdf:nodeID="KeyFS" />
 <foaf:interest rdf:resource="http://..." />
 <foaf:currentProject>
  <foaf:Project>
   <dc:title>Implementation Module X</dc:title>
   <dc:description>WS, programming, java</dc:description>
   <dc:identifier rdf:resource="http://.../activity#4539"/>
  </foaf:Project>
 </foaf:currentProject>
 <foaf:knows>
  <foaf:Person>
   <foaf:mbox_sha1sum>1a4578...</foaf:mbox_sha1sum>
   <foaf:name>Daniel Schall</foaf:name>
  </foaf:Person>
 </foaf:knows>
</foaf:Person>
</rdf:RDF>
```

Listing 7.1: Example of public FOAF file.

Listing 7.1 shows a simplified example of a public FOAF profile, containing basic personal properties (`name`, `nick`, `interest`) and social relations (`knows`). The Web of Trust (WoT) RDF ontology is used to integrate concept of a public key infrastructure into FOAF profiles, as demonstrated in Listing 7.2. The property `haskey` links a public key (`pubkeyAddress`), `hex_id`, and `fingerprint` to a `person`. Furthermore, a person's private key is used to sign the own FOAF profile and therefore, to guarantee for integrity and authenticity. Notice, the only guarantee regarding authenticity is that the FOAF signer is owner of the registered mail account that has been used to create the key pair. Access to parts of a FOAF document may be restricted to certain users (whose public keys are used to encrypt those parts). We utilize this concept for (i) private information, such as private phone numbers or chat accounts that can only be decrypted and used by close neighbors (connected via `knows`), and (ii) personal ratings

[5] GnuPG: `http://www.gnupg.org`

[6] Web of Trust: `http://xmlns.com/wot/0.1/`

that are given either *explicitly* (manually) or *implicitly* (through data mining of e-mail logs, instant messaging (IM) logs, or Internet forums). We understand privacy as a major concern when applying mining techniques; hence, mining of metrics is performed from each actor's perspective (or at least limited to certain groups of experts). This means that data is not stored centrally but managed on the client side and private servers.

```
<!-- restricted part of FOAF profile -->
<rdfs:seeAlso>
 <foaf:Document rdf:about="http://.../foaf-private.rdf.asc">
  <wot:encryptedTo>
   <wot:PubKey wot:hex_id="34c5a421b" />
  </wot:encryptedTo>
 </foaf:Document>
</rdfs:seeAlso>

<!-- digital signature for this file -->
<rdf:Description rdf:about="">
 <wot:assurance rdf:resource="foaf.rdf.asc" />
</rdf:Description>

<!-- public key of the owner/signer of this file -->
<wot:PubKey rdf:nodeID="KeyFS">
 <wot:hex_id>3756EA0B</wot:hex_id>
 <wot:length>1024</wot:length>
 <wot:fingerprint>03f4...</wot:fingerprint>
 <wot:pubkeyAddress rdf:resource="http://.../key.asc"/>
 <wot:identity>
   <wot:User>
     <foaf:name>Florian Skopik</foaf:name>
     <foaf:mbox_sha1sum>a4b378...</foaf:mbox_sha1sum>
   </wot:User>
 </wot:identity>
</wot:PubKey>
```

Listing 7.2: Signing FOAFs (`wot:assurance`) and linking content (`rdfs:seeAlso`).

```
<rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/">
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
 <foaf:Person>
  <!-- mbox_sha1sum links to public FOAF profile -->
  <foaf:mbox_sha1sum>a4b378...</foaf:mbox_sha1sum>

  <!-- private contact details -->
  <foaf:mbox rdf:resource="mailto:skopik@....tuwien.ac.at"/>
  <foaf:phone>+43 xxxx xxxx</foaf:phone>

  <!-- private chat account -->
  <foaf:account>
   <foaf:OnlineAccount>
     <rdf:type rdf:resource="http://.../OnlineChatAccount" />
     <foaf:accountServiceHomepage rdf:resource="http://..../" />
     <foaf:accountName>florian_skopik</foaf:accountName>
   </foaf:OnlineAccount>
  </foaf:account>

  <!-- attach personalized ratings to known persons -->
  <foaf:knows>
   <foaf:Person>
    <foaf:mbox_sha1sum>1a4578...</foaf:mbox_sha1sum>
    <foaf:tipjar rdf:resource="http://..." rdfs:label="ratings"/>
   </foaf:Person>
  </foaf:knows>
 </foaf:Person>
</rdf:RDF>
```

Listing 7.3: Private fragment of a FOAF profile.

Listing 7.3 depicts an example of encrypted private FOAF fragments. While users decide manually which parts of their profiles are shared globally and which are restricted to neighbors only, relation metrics, e.g., derived from personal ratings, are managed automatically by the system. For that purpose, single ratings are stored in a dedicated document (`tipjar`) for each user. This document is processed by various evaluation tools and plugins that are fully under control of the users. Currently, we have three tools for (i) collecting manual ratings, (ii) analyzing Internet forums, and (iii) analyzing e-mail communication in order to assess collaboration performance of known partners and the strength of social ties based on past interactions.

Profile Sharing

The presented concepts enable the discovery of directly connected partners based on common properties, interests, ratings, and contextual constraints (such as projects), but still preserve their privacy. This means that profile owners encrypt sensitive parts of their profiles for their known neighbors, i.e., using their public keys. Since we do not only manage binary knows relations but also calculate the strengths of relations (e.g., social trust), the amount of shared information can be bound to certain strength levels. For instance, whenever one updates his profile, a rule-based system decides based on predefined link thresholds, who is allowed to read private FOAF fragments and encrypt files accordingly.

```
<!-- link encrypted document -->
<foaf:Document rdf:about="http://.../foaf47.rdf">
 <dc:title>Restricted Information</dc:title>
 <wot:assurance>
   <wot:Endorsement rdf:about="http://.../foaf47.rdf.asc">
     <dc:title>signature of friend47 private profile</dc:title>
     <wot:endorser rdf:nodeID="KeyFS"/>
   </wot:Endorsement>
 </wot:assurance>
</foaf:Document>

<!-- encryption information -->
<wot:EncryptedDocument rdf:about="http://.../foaf47.rdf.asc">
 <dc:title>friend47 private profile</dc:title>
 <wot:encryptedTo rdf:nodeID="KeyPartnerX"/>
 <wot:encrypter rdf:nodeID="KeyFS"/>
</EncryptedDocument>
```

Listing 7.4: Linking encrypted documents in FOAF.

However, single members usually build up strong relations to only a small amount of partners. That hinders the discovery process. In order to overcome that hurdle, we allow propagation of information over several intermediate hubs along strong social paths. Enabling such flows of information enables actors to discover new potential collaboration partners. Technically, we allow actors to link *private* profile information of well connected partners as personally encrypted documents to their own profile. Restricted access is the basis for personalized and reliable sharing of information. We use once more the WoT ontology to link external documents to one's FOAF profile (see excerpt in Listing 7.4). A detailed implementation perspective regarding processing of XML data is out of scope of this work, but has been investigated in detail in [Skopik et al., 2010c]. A semantically-enriched Web Service based environment allows to notify partners about updated profiles and send them links to encrypted documents. The receivers are able to validate these documents, i.e, verify the authenticity and consistency using the signer's public key and to decrypt information using their own private keys.

Semantic Service Infrastructure

WSMX (Web Service Modeling eXecution environment) [Haller et al., 2005] allows to describe and register Web services and thus, supports discovering, selecting, and invoking Web services at run-time in a semantic manner. The actual services are hosted elsewhere, but WSMX builds a semantic abstraction layer for these services by managing additionally required artifacts (as described in Section 7.2). The WSMX platform provides a WS entrypoint to submit semantic goals that need to be fulfilled and the platform itself discovers the best suitable service based on (i) functional properties (FPs), i.e., supported concepts, such as messaging; and (ii) non-functional properties (NFPs), here, contextual constraints including organizational boundaries, people's location and working context.

Registering Semantic Web Services

The first step of registering a common Web service with a WSDL interface in WSMX is to annotate appropriate lowering- and lifting scripts. These XSLT scripts enable the transformation between SOAP messages and ontological representations. Listing 7.5 shows a small excerpt of a semantically-enriched WSDL file.

```
<xs:element name="sendMessageKey"
 sawsdl:loweringSchemaMapping="SendEmailMessage-lowering.xslt">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="0" name="to" type="xs:string"/>
   <xs:element minOccurs="0" name="subject" type="xs:string"/>
   <xs:element minOccurs="0" name="body" type="xs:string"/>
   <xs:element minOccurs="0" name="key" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="sendMessageKeyResponse"
 sawsdl:liftingSchemaMapping="SendEmailMessage-lifting.xslt">
 <xs:complexType>
  <!-- details omitted -->
 </xs:complexType>
</xs:element>
```

Listing 7.5: Schema mapping annotations in WSDL.

Here, the complex data type `sendMessageKey` (and its corresponding response) have `loweringSchemaMapping` and `liftingSchemaMapping` respectively attached. Listing 7.6 shows a lowering script. Here, values of required semantic concepts to build an instance of type `sendMessageKey` are extracted from the enterprise collaboration ontology.

```
<xsl:template match="rdf:Description[rdf:type/@rdf:resource=
'http://www.coin-ip.eu/ontologies/ec#EmailServiceMessage']">
 <email:sendMessageKey>
  <xsl:for-each select="ecg:hasEmailAddress">
   <to><xsl:value-of select="."/></to>
  </xsl:for-each>
  <xsl:for-each select="ecg:hasSubject">
   <subject><xsl:value-of select="."/></subject>
  </xsl:for-each>
  <xsl:for-each select="ecg:hasContent">
   <body><xsl:value-of select="."/></body>
  </xsl:for-each>
  <xsl:for-each select="ecg:hasAuthenticationKey">
   <key><xsl:value-of select="."/></key>
  </xsl:for-each>
 </email:sendMessageKey>
</xsl:template>
</xsl:stylesheet>
```

Listing 7.6: Lowering script example.

Semantic Goal Description

Listing 7.7 shows exemplarily a goal defined in WSML[7] for sending a notification via e-mail. For that purpose, NFPs are defined (here: type of discovery), as well as pre- and postconditions for invoking a capable Web service (e.g., defined recipient and message). The block `instance emailRequest` contains the actual parameters that are lowered to a SOAP message and sent to an Email Web service.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"
namespace { _"http://www.coin-ip.eu/goals/ec#",
disc _"http://wiki.wsmx.org/index.php?title=DiscoveryOntology#",
ec _"http://www.coin-ip.eu/ontologies/ec#",
ecp _"http://www.coin-ip.eu/ontologies/ecp#"}

goal MessageGoal
 importsOntology {
  ec#EnterpriseCollaborationOntology,
  ecp#EnterpriseCollaborationProcess
 }

capability MessageGoalCap
 nonFunctionalProperties
  disc#discoveryStrategy hasValue disc#NoPreFilter
  disc#discoveryStrategy hasValue disc#HeavyweightDiscovery
 endNonFunctionalProperties

sharedVariables {?x, ?z, ?y}

precondition MessageGoalPre
 definedBy
  ?x memberOf ec#EmailMessage and
  ?z memberOf ec#Individual and
  ?y memberOf ec#Individual.

postcondition MessageGoalPost
 definedBy
  ecp#messageSent(?z, ?x, ?y).

ontology EmailRequest
 importsOntology {
  ec#EnterpriseCollaborationOntology
 }

instance emailRequest memberOf ec#EmailMessage
 hasAuthenticationKey hasValue "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxx"
 hasEmailAddress hasValue "name@infosys.tuwien.ac.at"
 hasSubject hasValue "Notification about project opportunity"
 hasContent hasValue "Dear sir, according to your profile ..."
```

Listing 7.7: Semantic goal for e-mail message service.

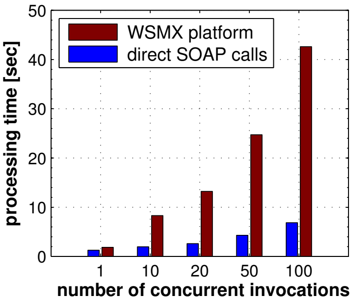
7.5 Evaluation and Discussion

This section deals with evaluation results regarding the whole system as well as discussions of essential findings. In particular, we demonstrate the performance of semantically-enriched service hosting with WSMX, discuss network formation processes using simulation, study member discovery processes through propagating distributed FOAF profiles, and discuss various design decisions with respect to PKI for FOAF.

WSMX Performance Aspects

The used WSMX setup consists of 38 different Web services, primarily communication services and document management services, 52 ontology parts (the main ontology is the enterprise col-

[7] Web service modeling language



WSMX step	time [ms]
lowering	250-400
service invoc.	> 130
lifting	150-250

single invoc.	time [ms]
SOAP (avg)	80-120
WSMX (avg)	450-750

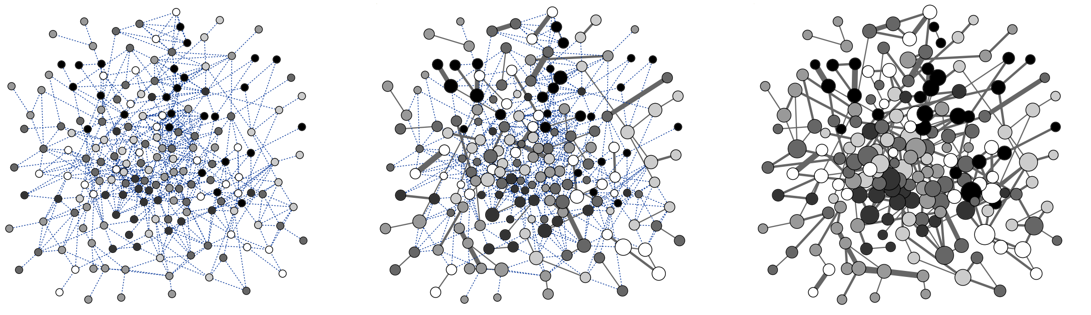
Figure 7.5: WSMX performance comparison.

laboration ontology depicted in Figure 7.3, but further ontologies of single services refine some concepts), and 13 semantic goals (e.g., sending a message with a given content to a particular person). For the following experiments, WSMX and services (implemented using Axis2[8]) are hosted on a server with Intel Xeon 3.2GHz (quad), 10GB RAM, running Tomcat 6 with Axis2 1.4.1 on Ubuntu Linux. Furthermore we perform concurrent calls from a client simulation that runs on a Pentium 4 with 2GB on Windows XP, and is connected with the server through a local 100MBit Ethernet. Figure 7.5 compares the performance of WSMX with standard SOAP calls that invoke Web services directly for different numbers of concurrent calls.. Note, the additional overhead caused by WSMX is the difference between the two results, since after processing the semantic layer also WSMX invokes a particular WS via SOAP only. In our test environment, invoking a service via WSMX compared to invoking the same service directly takes approximately 5 times longer. The additional processing time is used for lowering a request (such as the goal in Listing 7.3) to a SOAP message, and, after invoking the service, lifting the response back to the semantic level. Although WSMX adds much additional overhead to service invocation, several advantages can be taken, including, dynamic discovery and selection of best suitable service instances (depending on NFPs), and establishing real cross-enterprise interoperability through data mediation on ontological level. Note, services can be distributed over several WSMX instances to distribute load and increase performance.

Network Formation Simulation in SOA

We use a Web service testbed to simulate the interaction behavior in SOA-based communities. The purpose of the Genesis2 framework [Juszczyk and Dustdar, 2010] (in short, G2) is to support software engineers in setting up testbeds for runtime evaluation of SOA-based concepts and implementations. It allows to establish environments consisting of services, clients, registries, and other SOA components, to program the structure and behavior of the whole testbed, and to steer the execution of test cases on-the-fly. G2's most distinct feature is its ability to generate real testbed instances (instead of just performing simulations) which allows engineers to integrate these testbeds into existing SOA environments and, based on these infrastructures, to perform realistic tests at runtime.

[8] Apache Axis2: `http://ws.apache.org/axis2/`



(a) bootstrapping phase: only predicted links ($\alpha = 1$).

(b) formation: mix of predicted and emerged links ($\alpha = 0.5$).

(c) saturation phase: only emerged links ($\alpha = 0$).

Figure 7.6: Network formation process visualization.

Experiment Setup. The created test environment consists of 200 autonomous services that simulate behavior in common flexible collaboration scenarios. Each service (called actor) has an interest/expertise profile assigned, consisting of 5 to 8 distinct keywords. Profiles may partly overlap. In order to bootstrap collaborations links between actors are predicted based on profile similarities. Typically, interest similarities are a reasonable grounding for future collaboration success and emerging personal relations [Ziegler and Golbeck, 2007]. During the actual collaboration single actors interact by delegating tasks and requesting support from other members of the community; thus, in our simulation we let random members interact in fixed time intervals. Each interaction is tagged with a maximum of 3 keywords and sent to actors with matching interest profiles. We run different tests and vary the number of globally known tags, as well as the amount of occurring interactions. The results of these experiments enable us to study the formation process of typical medium scale Web-based communities. In particular we investigate the three phases of (i) bootstrapping, i.e., initiating the formation of a network; (ii) formation phase, i.e., setting up strong links between matching collaboration partners; (iii) saturation phase, i.e., cross-linking emerging small-scale communities with weak links. The aim of this experiment is to determine the effort in terms of monitoring and processing interactions until similar network structures (in the respective evolutionary phases) for different taxonomy complexities emerge. For instance, using less complex taxonomies consisting of only 10 keywords also requires less monitored interactions, since profiles and interaction contexts converge much faster than for more complex taxonomies.

Experiment Results. We study the network formation process of 200 unconnected actors for different environment setting. Depending on the complexity of the global taxonomy that determines interaction contexts, varying amounts of interactions are required in order to guarantee a feasible inference of social relations based on interest similarities. We let actors pick tags from a global taxonomy consisting of 10/20/50 keywords according to their interest profiles in order to annotate their interactions, e.g., express the expertise areas of support requests. In order to bootstrap a network formation process (see Figure 7.6(a)) links are predicted only (see dashed lines) based on actor profile overlaps [Skopik et al., 2010f]. Utilizing measured interaction metrics (here reciprocity cf. Eq. 4.5), social links are established based on evidence about reliable and

phase	#tags / #ia	network metrics
bootstrapping	10 / 0	$nc = 200$, $nn = 0(7.62)$, $nd = 0$
	20 / 0	$nc = 200$, $nn = 0(5.56)$, $nd = 0$
	50 / 0	$nc = 200$, $nn = 0(1.13)$, $nd = 0$
formation	10 / 1 000	$nc = 99$, $nn = 1.12$, $nd = 0.006$
	20 / 3 000	$nc = 109$, $nn = 0.84$, $nd = 0.005$
	50 / 5 000	$nc = 101$, $nn = 0.98$, $nd = 0.005$
saturation	10 / 5 000	$nc = 4$, $nn = 3.15$, $nd = 0.017$
	20 / 15 000	$nc = 7$, $nn = 2.65$, $nd = 0.014$
	50 / 25 000	$nc = 5$, $nn = 2.89$, $nd = 0.016$

Table 7.1: Characteristic metrics of a social overlay network in different evolutionary phases of a formation process.

dependable collaboration behavior. Note, the color of the nodes represent their (static) expertise areas, while their sizes reflect their degree of connectivity in the network. Figure 7.6(b) shows a network where most members have found at least one trustworthy (e.g., in terms of reciprocity) collaboration partner. Such social links are reflected by solid lines whereas their strengths reflect the level of cooperation. Still, most relations are predicted only (dashed lines). Finally, after a sufficient amount of interactions has been collected to reliably infer relations, a network consisting only of evidence-based relations is maintained in the saturation phase (Figure 7.6(c)).

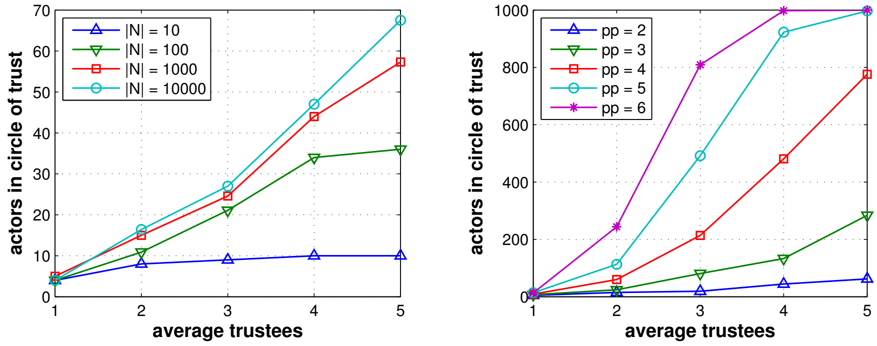
We repeat this experiment to find out typically emerging network structures for varying taxonomy complexities (number of tags #tags) and different amounts of interactions (#ia). Table 7.1 reveals the details. The metrics are (i) number of connected components (nc), (ii) average number of network neighbors (nn), and (iii) network density (nd). Although an optimal connection is hard to determine, these graph metrics deem to be appropriate indicators [Romesburg, 2004] to describe and compare network structures. Note, the values in brackets in the bootstrapping phase denote the given metrics if predicted links are treated as evidence-based links.

Member Discovery Simulations

We create synthetic networks with fixed amounts of nodes and power-law distributed edges [Reka and Barabási, 2002] to evaluate the effects of propagating profile information. This means, encrypted parts of a FOAF profile are shared over multiple hops even between unconnected members, if there is a strong trust path between them. This concept of propagation [Guha et al., 2004] enables users to extend their *circles of trust* (i.e., all members that can be reached over a strong trust path without exceeding an upper limit of hops) and to discover previously unknown members therein. The complexity of a graph is described by the average outdegree of a node in the long tail of the degree distribution; in other words, the average number of trusted neighbors (trustees) for the bottom 90% of members. We pick random nodes from this set and run experiments for each one until we get stable average results.

The first set of experiments[9] investigates the average size of the *circle of trust*, depending

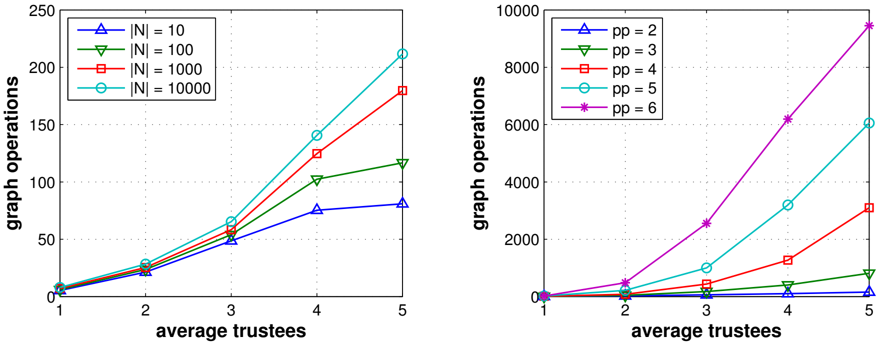
[9] Notice, these experiments are inspired by [Skopik et al., 2010b] since the basic settings and assumptions are



(a) varying network size $|N|$.

(b) varying prop. path pp.

Figure 7.7: Size of the circle of trust.



(a) varying network size $|N|$.

(b) varying prop. path pp.

Figure 7.8: Required graph operations.

on the number of trustees for different network sizes N and propagation path lengths pp. For that purpose profiles of all neighbors of specified nodes in the network are retrieved recursively until the whole circle is discovered. Figure 7.7 show that for highly cross-linked graphs (i.e., $avgtrustees > 2$), only short pps (max. 3 or 4 hops) are feasible. Otherwise, virtually all members are in one's *circle of trust*. A second set of experiments highlights the computational complexity of determining the *circle of trust*. While the size of the network does not considerably influence the number of required graph operations from each actor's perspective (at least for small pp), increasing pp in highly cross-linked graphs leads to exponential costs (Figure 7.8). Graph operations include retrieving referenced nodes and edges, as well as neighbors, predecessors and successors in the network model. Each of these operations means that finally distributed FOAF profiles need to be queried and retrieved from the Web.

similar to the use case and environment here.

Processing Encrypted FOAF Profiles

We shortly discuss the complexity and required steps to enable the discovery of collaboration partners based on FOAF profile sharing using the security concepts discussed in this chapter. For that purpose, we distinguish between three different operations: (i) publishing profiles, (ii) discovering neighbors, i.e., retrieve their (encrypted) profiles, (iii) transitive discovery, i.e., propagation of profile information over one hop. Table 7.2 summarizes complexities in terms of *number of retrieved documents* (i.e., public/private FOAF fragments, signatures, public/private key files) and *number of required steps* (i.e., file retrieval, encryption, decryption, file update, file upload). Note, we do not measure absolute performance of the proposed profile management approach, because this heavily depends on the hosting environment and IT infrastructure. Symbol n denotes the number of direct neighbors; p the number of distinct private FOAF fragments.

operation	**#retrieved docs**	**#steps**
FOAF profile publishing	$3 + n$	$3 + 3p + n$
neighbor discovery	$(2 + p) \cdot n$	$(3 + p) \cdot n$
transitive discovery	$2 + 2p + n + pn$	$3 + 3p + n + pn$

Table 7.2: Comparison of profile management ops.

FOAF Profile Publishing. Updating an actor's own profile consists of profile retrieval and update of already existing public/private profile fragments, signing the public fragment with own private key, retrieving the neighbors' public keys, encrypting private fragments individually for strongly connected (e.g., trusted) neighbors, publish public and private fragments on the Web.

Neighbor Discovery. This operation discovers directly connected actors by evaluating their profiles, e.g., interests, project participation, organizational memberships. Evaluating neighbor profiles includes for each single neighbor to retrieve the public profile and public key to validate the signature, retrieval of linked private fragments, decryption of data with own private key.

Transitive Discovery. Transitive profile sharing enables the discovery of unconnected community members. For that purpose intermediate nodes mediate information by retrieving (encrypted) profiles from neighbors, and re-encrypt them for their own (trusted) neighbors. In particular the following steps are performed: retrieve published public/private FOAF fragments of one neighbor, get public key to verify signature, decrypt private fragment with own private key, get public key of other neighbor(s), re-encrypt private fragment, attach this fragment to own FOAF profile, re-sign and re-encrypt own FOAF fragments; optionally, notify interested neighbors about third-party profiles.

7.6 Conclusion

In this chapter, we discussed the application of social network concepts in cross-enterprise collaboration scenarios from a semantics perspective. While creating dynamic profiles and flexibly discovering people and services is frequently used in typical recommender systems and on social platforms, the application in enterprise scenarios in form of overlay networks is a novelty. Especially, the combination with Semantic Web methodologies, such as Web services, taxonomy-based context management and SOA to achieve data and service interoperability is a new aspect. We proposed an approach to support human collaboration in different domains and organizations in a seamless manner; not only from s social perspective, but also from a technical one.

CHAPTER 8

Information Flows Through Strategic Social Link Establishment

Social networks have emerged from niche existence to a mass phenomenon. Nowadays, their fundamental concepts, such as managing personal contacts and sharing profile information, are increasingly harnessed for businesses in professional environments. Similar to service-oriented networks, they allow flexible discovery on demand and loose coupling of participants. Establishing social links facilitates cooperation and enables selective sharing of information. Intuitively, one shares more information with his connected neighbors and less or even none with unrelated individuals. Today, information is one of the most important and valuable goods in business networks. Being informed about ongoing collaborations and upcoming trends is a key success factor. Thus, in professional networks, participants aim at strategically establishing connections to enable reliable information flows. In this chapter, we especially highlight an opportunistic model that let mediators connect actually unrelated actors in order to benefit from information mediation.

8.1 Introduction

The Web has evolved from a distributed document repository to an interactive medium in which people actively share and disseminate information. Parts of this evolution is often referred to as Web 2.0 and characterized by the emergence of knowledge sharing communities. The way people interact on the Web, especially in professional environments, is changing once more. Service-oriented computing takes off on its triumphal course to permit even human-centric business platforms (such as Amazon Mechanical Turk). Web services enable loosely-coupled cross-organizational collaborations, and are the ideal means to realize flexible discovery and binding of interaction partners. In such service-oriented collaboration environments, participants shape the availability of information and services.

Social networks typically emerge freely and independently without restricted paths and boundaries. Research has shown that the resulting social network structures allow for relatively short paths of information propagation. For example, ‘six-degrees of separation’ [Guare, 1990] refers to the idea that each node in a freely emerged people network can be reached by propagating items of information via six hops. 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. In order to take advantage of social preferences, we propose social network principles to overcome limited information flows in collaborative environments. Particularly, in *service-oriented professional communities*, actors perform activities by interacting with other community members, e.g. to inquire information, exchange ideas, and delegate requests. Over time, actors establish social connections to their collaboration partners [Camarinha-Matos and Afsarmanesh, 2006; Skopik et al., 2010a]. Information are propagated along these links. Typically people share more information with well-known partners who proved their reliability earlier; and less or even none with unknown third parties. Hence, to motivate two unconnected users to exchange information, and thus, enable reliable information flows in networks, they need a mutually known intermediate actor.

According to the structural holes theory [Burt, 1992], people show the tendency to position themselves in networks as such *interaction mediators*. Their main incentives are to get valuable insights in ongoing collaboration and others’ work. Mediating interactions between network members further allows to influence and control partners and to build up distinct reputation and high visibility levels. However, acting as mediator also requires free capacities, e.g., in terms of time and effort, and since resources are usually limited it is a top priority to carefully decide about mediation targets.

In this chapter we deal with the following contributions:

- *Information Mediation Model.* We design an analytical model that explains the fundamental concepts of opportunistic information flows. Our model introduces the notion of *utility* in service-oriented professional communities.

- *Social Link Establishment.* We formalize the incentives and motivation of human behavior regarding social link establishment in virtual communities established upon interaction analysis.

- *Evaluation and Discussion.* We evaluate the proposed model and its application for virtual communities, and derive general findings for designing applications for socially-enhanced service-oriented environments. Also, we discuss the application of and integration with available social network standards for open environments.

8.2 Social Information Mediation

We discuss a layered *social information mediation model* and outline utilized major concepts on each layer.

Overview

Basically, an actor has several *passive links*, such as FOAF [Brickley and Miller, 2010] relations, that just express business/personal contacts (typically emerged from previous collaborations), but no interactions are performed along these links. An actor can activate these links (*active links*) by initiating a new collaboration, e.g., setting up a joint activity. However, due to resource constraints, members can only participate in a limited amount of concurrent activities, and thus, the number of simultaneously active links is limited. Hence, collaboration partners are selected carefully, considering required effort and received utility.

Figure 8.1 shows an overview of our layered approach. On the bottom layer, interactions are observed and collected to determine social relations. We designed the system to manage relations by evaluating occurring interactions and therefore, unburden network participants – at least partly – from managing their relations manually [Skopik et al., 2010a]. On the medium layer, direct relations are established to create a typical social network. Since single members usually build up strong relations to only a small amount of partners, reliable information flows through collaborative networks are limited. Therefore, on the top layer, social mediation is applied.

We have studied the bottom and middle layer, as depicted by Figure 8.1, already in our previous work. We briefly outline the main principles of these layers to highlight the novelty of the proposed utility model.

- **Interaction Layer.** A professional virtual community (PVC) is a special kind of social network $G = (V, E)^1$, where the single actors participate to perform activities $a \in A$.

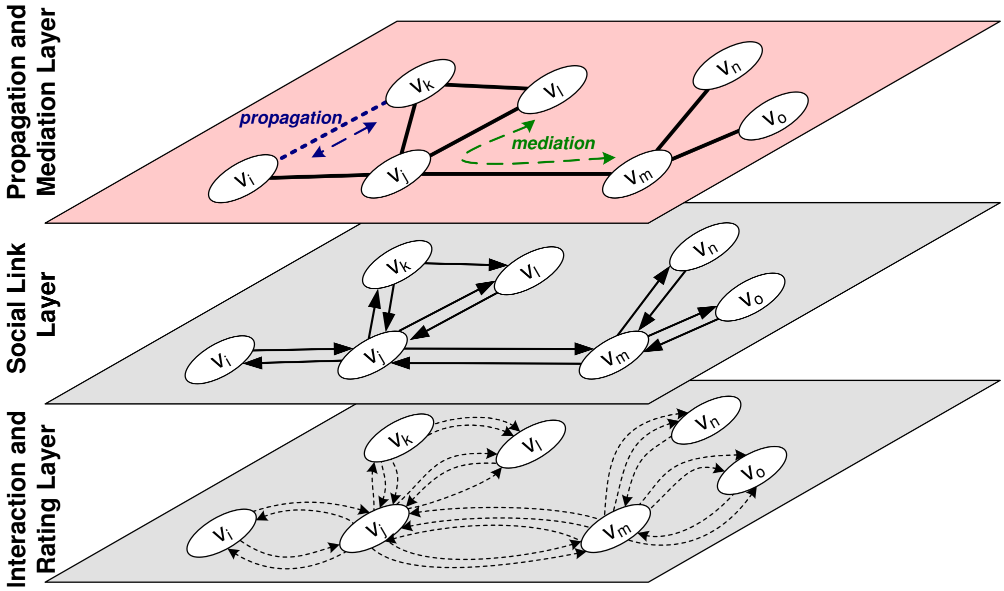


Figure 8.1: Model for social information mediation.

[1] Notice, in the previous chapters, we have introduced all required concepts to build socio-computational crowd

Here, a directed edge from actor v_i to v_j is denoted as e_{ij}. As before, network members interact in scope of particular activities to reach certain goals. Interactions are collected using techniques from previous chapters to determine (i) the center of interest of single network members by evaluating the frequency of used keywords [Schall and Dustdar, 2010; Skopik et al., 2010a], and (ii) the strength of a social relation by determining the similarity of the center of interests [Skopik et al., 2010a].

- **Social Link Layer.** By issuing keyword-based queries, each actor's connectivity to other community members in a particular query context Q is determined [Schall and Skopik, 2010]. The query context is described by a pool of keywords (e.g., describing certain expertise areas) picked from global taxonomies. Using logged interaction data and manual ratings the link strength from one actor v_i to another v_j in context of Q is calculated. For that purpose, various metrics, such as availability of actors, average ratings, responsiveness and interest similarities can be calculated dynamically.

Information Mediation

Typically, network members share information along emerging direct relations. Sharing with known partners is beneficial for both, the information provider who can spread information but still knows the boundaries of potential receivers, and the receiver who directly knows the provider and can decide on the trustworthiness of delivered information. However, each network member does usually interact with only a small amount of distinct partners compared to the overall size of a large-scale system. Thus, information, such as announcements and invitations, cannot be reliably and also widely spread at the same time.

In order to compensate that issue, several propagation models have been proposed to establish artificial connections in networks that are inferred from existing ones [Guha et al., 2004]. The most common concept is *direct propagation*. Here, a node v_i has a strong relation to v_j which is tightly linked to v_k (see Figure 8.1). Propagation means that, although v_i and v_k have never personally interacted with each other, a synthetic social relation can be introduced using the existing network structure. For instance, if these social links are considered as trust relations [Skopik et al., 2010a] one could say, because v_i trusts v_j and v_j trusts v_k, there is a high probability that v_i can also trust v_k; e.g., due to their similar habits, attitudes, and values.

However, sharing information along synthetic links, has several implications and potential disadvantages for the involved actors. Typically, the information owner v_i shares information with the actually unknown sink v_k, v_k receives information from an unknown source v_i, and v_j initially supports establishing a connection between v_i and v_k but is than bypassed and cannot control the actual information propagation over that supported link. Furthermore, there may be situations, where v_i does not want to reveal its identity to unknown network members (here: v_k), however, still wants to distribute information. Hence, additionally to these propagation models, we further apply the concept of active *information mediation*.

systems (avatars, self-managing social links, query mechanisms, and extended FOAF models) as motivated in the introduction. Thus, from now on, we understand the users not as service nodes in a service-oriented system only (cf. [Skopik, 2010]), but as social nodes in a socio-computational system. In order to reflect this change of understanding, we denote nodes with V instead of N.

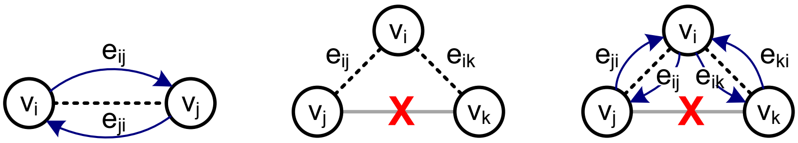
8.3 Utility-based Mediation Model

We begin with the basic mechanisms describing motivation and incentives for building relations on social Web platforms.

Link Establishment

Information mediation assumes that actors actively bridge isolated components in a social network. They mediate information to facilitate collaborations between actually unconnected network members. The most fundamental questions, however, are (i) what is a mediator's motivation for doing that and (ii) how do they select their interaction partners?

Reciprocity. Ultimately, each actor participates in the network for a certain reason, e.g., for improving its reputation, alleviating the access to information, or becoming a key player. In other words, actors aim at improving visibility and impact in the community. Intuitively, actors favor well-known partners for collaboration and information exchange over any unknown ones (social networking concept); i.e., partners who already proved their reliable and dependable behavior. Mutual social relations are required to maximize the probability of successful future interactions. Thus, our model utilizes the concept of *reciprocity* (Figure 8.2(a)).



(a) Reciprocity (b) Structural hole. (c) Joint Model.

Figure 8.2: Social theory models.

The reciprocity theory [Falk and Fischbacher, 2006] explains the motivation for reliable interaction behavior of actors with a mutual give and take relationship. The more reciprocal relations an actor has, the higher is its probability to find a reliable collaboration partner and thus, the higher is its benefit obtained from the participation in the network. The utility U_R for an actor v_i from using existing reciprocal relations can be formulated as given in Eq. 8.1. Basically, this value is composed of the sum of weighted (w) mutually strong ties; i.e., there exist equally weighted directed edges e_{ij} and e_{ji} between v_i and v_j as shown in Figure 8.2(a). Hence, the utility value of node v_i increases for each reciprocal relationship that it maintains. Utilized relations are determined by a query context Q to determine the utility in a *certain* expertise area.

$$U_R^Q(v_i) = \sum_{j=0}^{|V|-1} w_{ij}^Q w_{ji}^Q \tag{8.1}$$

Our model uses two metrics to determine link weights w (Eq. 8.2): (i) interest similarity $isim$ [Skopik et al., 2010a], calculated from the similarity of interaction contexts (determined through tags), and (ii) the average mean of assigned ratings $avgr$. While $isim$ is a globally valid metric, $avgr$ is bound to distinct expertise areas. In particular, all activities whose description match at least one of the query keywords are considered. The function $match(Q, a)$ determines the matching (within the range $[0, 1]$) of an activity a to a query Q. Whether $isim$ or $avgr$ has more impact on the final link weight w_{ij}^{Q} can be configured through a globally valid weight $\alpha \in [0, 1]$. Currently we employ flat keyword-based matching only, however for more advanced ontology matching techniques see [Castano et al., 2006].

$$w_{ij}^{Q} = \alpha \cdot isim_{ij} + (1 - \alpha) \cdot \frac{\sum_{a \in A} avgr_{ij} \cdot match(Q, a)}{\sum_{a \in A} match(Q, a)} \tag{8.2}$$

Structural Holes. Another social theory based on self-interest is the concept of structural holes [Burt, 1992]. This theory states that there are actors who actively position themselves in beneficial positions within a community network. Such an actor filling a structural hole connects two previously unconnected (or at least loosely coupled) actors, and gains direct advantage by doing that. As depicted in Figure 8.2(b), v_i establishes an indirect link between the actually unconnected nodes v_j and v_k via edges e_{ij} and e_{ik}. In general, actors such as v_i collect larger amounts of contacts and represent the main hubs of information exchange [Kleinberg et al., 2008]. Therefore, they are able to collect information on ongoing collaborations and can also exercise greater control on connected members. In the presented information mediation use case, the utility U_{SH} for v_i results from being able to control the communication of linked actors and collect exchanged information.

$$U_{SH}^{Q}(v_i) = \sum_{j=0}^{|V|-1} w_{ij}^{Q} \sum_{k=0}^{|V|-1} w_{ik}^{Q}(1 - w_{jk}^{Q}) \tag{8.3}$$

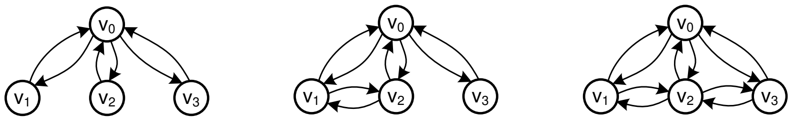
In an opportunistic network, the motivation of each actor is to maximize its utility. According to aforementioned concepts, well-known reliable actors are predominantly picked and a mutual social relation is required to maximize the probability of successful future interactions. Each additional mediation role poses additional effort to the affected actor. However, each actor's mediation capacity is limited, thus the strategic positioning in the network is still of paramount importance.

Joint Model. Combining the concept of reciprocity and the structural holes theory, the overall utility value U in context of query Q is calculated as given in Eq. 8.4. Basically, the joint model has the same form as the structural hole model (Figure 8.2(c)), however, considers directed (and ideally balanced) relations between single actors. In case of social information mediation needs the utility value of each actor is evaluated. Thus, by periodically setting up new mediation activities and releasing unprofitable mediation partners (i.e, mediation targets) the utility value obtained from the network can be optimized, i.e., increased.

$$U^{Q}(v_i) = \sum_{j=0}^{|V|-1} w_{ij}^{Q} w_{ji}^{Q} \sum_{k=0}^{|V|-1} w_{ik}^{Q} w_{ki}^{Q}(1 - w_{jk}^{Q})(1 - w_{kj}^{Q}) \tag{8.4}$$

Mediation Cases

We study various cases of mediation needs as shown in Figure 8.3 to demonstrate the application of the proposed mediation algorithm. For those illustrative examples we neglect contextual constraints Q and thus assume that connections are established with a weight $w = 1$. Furthermore, assuming $w = 1$ for the following examples better visualize the impact of certain links in the network.



(a) full. (b) partial. (c) limited.

Figure 8.3: Various mediation needs for v_0.

In particular, we investigate the obtained utility of a certain v_0 in simple networks consisting of only four nodes (Figure 8.3). Here, $U(v_0)$ can be calculated as given in Eq. 8.5. Note, self connections are implicitly weighted with $w = 1$.

$$U(v_0) = \sum_{j=0}^{3} w_{0j} w_{j0} \sum_{k=0}^{3} w_{0k} w_{k0} (1 - w_{jk})(1 - w_{kj}) \tag{8.5}$$

Actually, four different cases of mediation needs exist from an algorithmic perspective (Figure 8.3 depicts three of them): (i) *Full Mediation* means that v_0 is encouraged to mediate information between all of its neighbors because none of them are interconnected; (ii) *Partial Mediation* means that there are some mediation needs that can be fulfilled exclusively, e.g., between v_2 and v_3 in Figure 8.3(b); (iii) *Limited Mediation* is the case when no mediation is required exclusively, because of available alternatives, e.g., mediation between v_1 and v_3 can be performed by v_0 and v_2 in Figure 8.3(c); (iv) *No Mediation* is the case, when there are neither exclusive nor alternative mediation needs, i.e., all neighbors of v_0 are directly connected with each other.

case	$\mathbf{U_{j=0}}$	$\mathbf{U_{j=1}}$	$\mathbf{U_{j=2}}$	$\mathbf{U_{j=3}}$	$\mathbf{U(v_0)}$
full	0	2	2	2	6
partial	0	1	1	2	4
limited	0	1	0	1	2
none	0	0	0	0	0

Table 8.1: $U(v_0)$ for different mediation cases.

Table 8.1 shows the results of utility computation for node v_0 in all of the mentioned cases. The first sum in Eq. 8.5 (with index j) is split in single subresults shown for $j = 0 \dots 3$. The last column shows the utility result $U(v_0)$. Typically, every node that is exclusively connected to v_0, i.e., not connected to another one of v_0's neighbors, adds a utility that is equal to $|N_{v_0}| - 1$, where

Algorithm 2 Utility optimization algorithm.

```
Input: context $Q$, node $v$
Global: activities $A$, network $G = (V, E)$
$V^Q$ ←passiveNeighbors($v, Q$)                ▷ passive neighbors with free capacities
$V^A$ ←activeNeighbors($v, A, Q$)              ▷ actively collaborating neighbors
$U^Q$ ←calcUtility($v, V^A, Q$)                ▷ current utility in context $Q$
$U^Q_{max} \leftarrow U^Q$
                                               ▷ try to replace single active members with passive ones
$V^Q_d$ ←determineDeLinkCandidates($v, V^A$)   ▷ active low-utility partners
for each $v^Q_d \in V^Q_d$ do                  ▷ for each partner that can be replaced
    for each $v^Q_i \in V^Q$ do                ▷ try all passive partners
        $V^A_a \leftarrow V^A \backslash \{v^Q_d\} \cup \{v^Q_i\}$   ▷ modified temporary active set
        $U^Q$ ←calcUtility($v, V^A_a, Q$)      ▷ alternative utility
        if $U^Q > U^Q_{max}$ then              ▷ apply changes if higher utility
            $U^Q_{max} \leftarrow U^Q$
            $V^Q \leftarrow V^Q \backslash \{v^Q_i\}$   ▷ remove candidate from passive set
            $V^A \leftarrow V^A_a$             ▷ make temporary set permanently active
```

$|N_{v_0}|$ is the number of v_0's neighbors in the network. In other words, v_0 has the opportunity to mediate between this exclusively connected node and all other neighbors. If that exclusive nodes becomes connected to one of v_0's neighbors, the obtained utility is reduced by 1 (see partial mediation case). Basically, supporting the emergence of connections between neighbors of v_0 reduces its utility in the first place. However, allowing members to establish links may be reciprocated by affected actors, and thus, extend the number of known network members.

Utility Optimization Algorithm

Each actor in the network periodically attempts to improve its current utility by replacing active collaboration partners that provide only low utility with known but currently passive network members. That procedure is described by Algorithm 2. First of all, when a node v executes this algorithm for a given context Q (e.g., expertise area), passive neighbors V^Q and active neighbors V^A are determined. Using the set V^A, v's current utility U^Q can be calculated. This value becomes the initial maximum value. Then a list of de-link candidates V^Q_d, e.g., actually unreliable collaboration partners or members of low utility, is determined. Now, the algorithm iterates through this set and tries to replace each single de-link candidate v^Q_d with one of the passive neighbors $v^Q_i \in V^Q$. If the utility value increases with this modification, the sets of active and passive neighbors are adapted accordingly. Notice, the assumption is that node v is in fact able to set up a collaboration (i.e., activity) with v^Q_i until the next update cycle. Otherwise, this replacement does not increase v's utility and the current v^Q_i would become a de-link candidate in the next update period.

8.4 Evaluation and Discussion

In this section we outline the application of our approach and derive findings by studying various simulations.

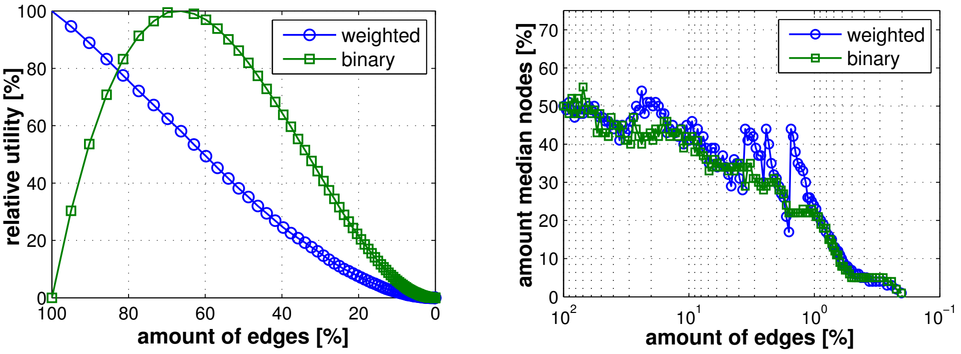
Experiment Background

Resilience (robustness) of complex networks can be studied by analyzing the effects of node and edge removal. Social networks, as studied here, are one class of a complex networks. Other examples include the WWW, power grids, or biological networks. *Percolation analysis* [Stauffer and Aharony, 1994] can be used in complex networks to understand the effects of node/edge removal; for example, to study the attack resilience of networks [Albert et al., 2000]. Here we take this approach to analyze the *utility evaluation* of mediators in social network under different conditions. The question we attempt to answer is how a mediator's utility is influenced by decreasing connectivity of the network. Based on gradually removed edges, we attempt to understand the effects of our utility metric. We expect that the utility of a mediator is particularly high if a network is not well-connected (see metrics such as average path length, density). Our approach is as follows:

- Take the initial network and calculate v's utility.
- Remove edges between nodes that are *highly similar*, thereby introducing potential 'gaps' and mediation opportunities. Repeat the calculation of a node's (mediator) utility.
- Remove edges between nodes that are *less similar* to study the impact of profile similarity (mediation between similar nodes versus mediation between nodes with different interests).

Discussion and Findings

Bootstrapping Collaboration Networks. First, we outline a bootstrapping mechanism for social and collaborative networks. Our approach heavily relies on monitored interactions and collected data. For instance, the intensity, i.e., weight, of social relations is determined through various metrics – see Eq. 8.2. In particular we utilize *interest similarity* ($isim$) to connect people with similar expertise (derived from manually defined profiles and tagging data), and *average rating* ($avgr$) for evidence-based relations based on previous collaborations. The parameter α controls which one of the parameters receive higher attention and influences the final weight more (Eq. 8.2). We distinguish between three cases: (i) *Bootstrapping Phase* ($\alpha = 1$): here, only $isim$ is used which describes a certain interest overlap between two people's profiles. Thus, potentially all nodes are connected to all others, independent from whether they collaborated in the past. Relations described by $isim$ only, can be seen as a prediction of successful future interactions. (ii) *Formation Phase* ($\alpha \in \,]0,1[\,$): here, we use a mix of $isim$ and collected $avgr$ to determine the weight. In other words, as members begin to interact, predictions based on $isim$ lose importance while assigned ratings due to reliable interaction behavior mainly influence a link's weight. (iii) Saturation Phase ($\alpha = 0$): after a while, rating-based structures emerge,



(a) utility.

(b) node median.

Figure 8.4: Utility and number of median nodes for varying edge saturations and weighting models (binary v.s. weighted edges).

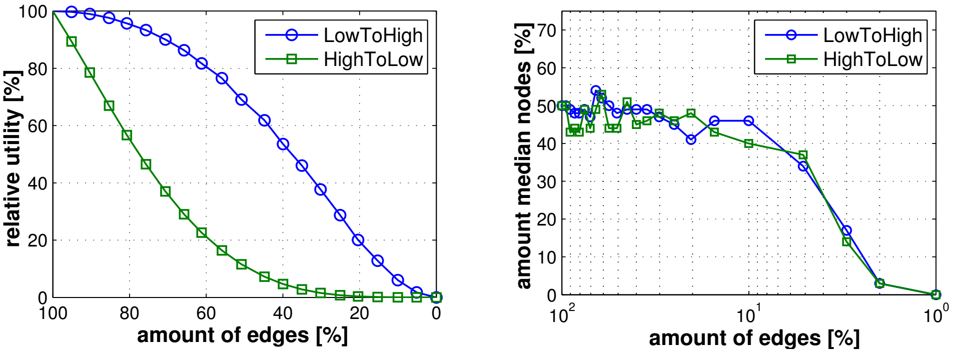
which means that only $avgr$ is used to determine a link's weight. However, these structures are in a constant flux and change due to periodic re-evaluation of personal utilities.

We use above mechanisms to create a synthetic collaboration network $G = (V, E)$ with $|V| = 100$ for our further studies. In the beginning, we assume that all nodes are connected to all others described by similarity relations ($isim$ uniformly distributed in $[0, 1]$) resulting in $|E| = \frac{1}{2} \cdot |V|(|V| - 1) = 499\,500$. Then, we utilize the *preferential attachment model* of Barabasi and Albert [Reka and Barabási, 2002] to let rating-based graphs emerge with power-law distributed degrees. These structures are the basis for a realistic scenario and edge weight distribution among members. As typcial for scale-free collaboration networks, we assume that 80% of interactions take place between 20% of the most active users.

Simple Percolation Analysis. Figure 8.4(a) depicts the aggregated utility (the percentaged amount of the maximum possible utility) of all nodes in the network when gradually decreasing the number of edges in the network. We start with a fully saturated graph (amount of edges is 100%). Two experiments show the difference between (i) binary relations only, i.e, weight = $\{0|1\}$, and (ii) weighted relations with weight = $[0, 1]$. In the binary case, the utility rises when edges are removed, because at the same time mediation opportunities emerge. At an amount of $\approx 65\%$ the received utility is at a maximum. In the weighted case, the utility decreases strictly monotonic from the beginning, since there are always mediation opportunities between neighboring nodes, even if they are directly connected (except the case where mediation targets are directly connected with an edge weight = 1). Figure 8.4(b) demonstrates the amount of nodes that receive a utility above the median. Both cases produce similar results, i.e, removing 90% of edges still results in a fair distribution with 45% to 50% of nodes having an utility above the median.

We expect that the weights of relations have major influence on the received utility of a node. Thus, we apply percolation strategies where (i) lowly weighted edges are removed first ($LowToHigh$), and (ii) highly weighted edges are removed first ($HighToLow$). Figure 8.5 shows again the aggregated utility of nodes and the amount of nodes receiving utility above the median value. Obviously lowly weighted edges have less influence on the aggregated utility

(utility drops slower) compared to highly weighted edges (utility drops faster). This is of particular interest, because one might assume that removing highly weighted edges first discovers new and better mediation opportunities. In fact, this is the case but also highly weighted connections to mediation targets are removed at the same time which results in lower average utility. The importance of weak links (even compared to strong ones) have been extensively studied by [Granovetter, 1973].



(a) utility.

(b) node median.

Figure 8.5: Utility and number of median nodes for varying edge saturations and different percolation strategies (LowToHigh v.s. HighToLow).

Bounded Percolation Analysis. Until now, we neglected the costs for maintaining relations. But in reality, keeping relations active results in high effort for serving requests and mediating information. Thus, usually only a small amount of all known partners are involved in active collaborations at the same time. We define that a node v has only a limited amount of resources to cope with *costs* for serving partners. We further define the *benefit-cost factor* $\zeta(v) = \frac{U(v)}{costs(v)}$ that is the basis to decide whether node v should maintain or release a partner. In other words ζ is the basis to determine how much effort v should invest and thus its re-linking behavior.

Typically, actors can cope with a predefined cost level, i.e., they have a limited amount of credits or resources (e.g., time) which they can invest. Thus, we argue that in most cases they attempt to keep $\zeta \rightarrow max$. Figure 8.6 visualizes the relative utility (amount from the highest possible utility when neglecting occurring costs) in a gradually disconnecting graph (decreasing amount of edges). For $\zeta \rightarrow max$, this experiment applies a cost function that allows a single member to collaborate with a maximum of 5 neighbors at the same time. Notice, in the beginning utility typically rises with decreasing amounts of edges when partners become served exclusively. Furthermore, only a limited number of neighbors can be served when accounting for costs. So, this curve has a shape similar to the binary weighted model in Figure 8.4(a) (binary edge weighting).

However, in certain cases actors may try to obtain a utility that is higher than in the case $\zeta \rightarrow max$. That means, they decide to invest comparatively more effort in order to obtain (even only slightly) higher utilities. Figure 8.6 shows the results for $\zeta \rightarrow \{max, \frac{max}{5}, \frac{max}{10}, \frac{max}{50}\}$. Note, we simplify the model and assume $min \approx \frac{max}{50}$; i.e., the worst (lowest) ζ factor means that costs are (nearly) neglected and each user is able to invest 50 times more resources than in

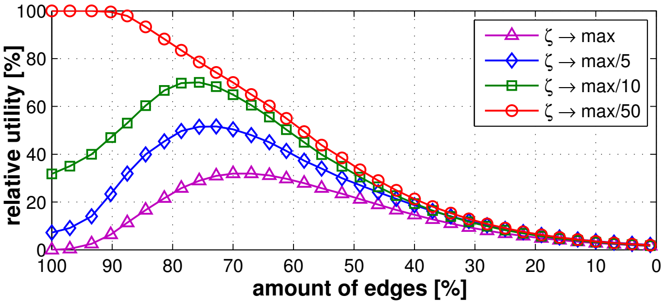
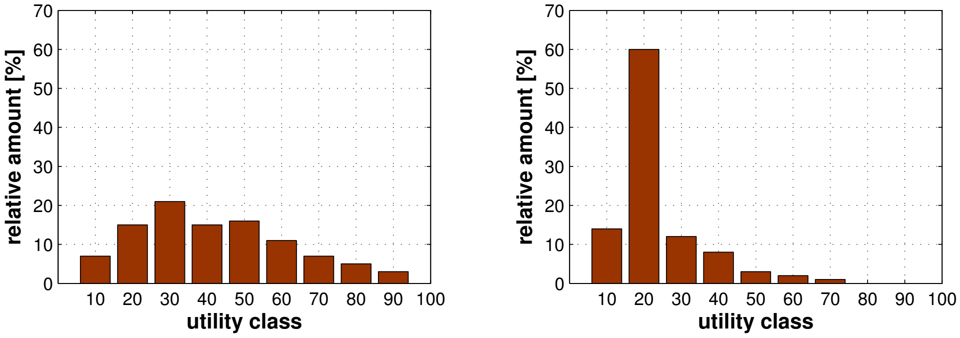


Figure 8.6: Utility for varying benefit-cost factors ζ.

the optimal case. As expected, the more effort is invested to maintain relations, the higher utility can be obtained. However, the amount of additional utility is comparatively low. For instance, if the network is still connected with 65% of the initial edges, then $U_{\zeta \to max} \approx 0.5 \cdot U_{\zeta \to max/50}$, i.e., the utility value can be doubled by coping with approximately 50 times higher costs.

Utility Distribution. We take the data from the previous experiments and study the case for an amount of edges of 65% (maximum utility for $\zeta \to max$) in greater detail. Until now, we just studied the aggregated utility in the whole network (which possesses similar characteristics as the *average* utility of a single user depending on the amount of edges in the network). Here, we group members with respect to their obtained utility and highlight the distribution of actors in the single classes. Figure 8.7(b) depcits the relative amount of members from the whole population in distinct utility classes (reflecting users obtaining more than 10,20,30... percent of the maximum possible utility value).

Figure 8.7 shows the results. In particular, when (nearly) neglecting costs ($\zeta \to \frac{max}{50}$) the distribution spans most utility classes. In other words, there are approximately as many actors who receive 20% of the maximum value as members with 50%. That distribution seems to be unfair. However, when accounting for costs, actors are not able to claim as many neighbors as possible (based on their structural connectivity) but need to carefully select their collaboratin



(a) $\zeta \to \frac{max}{50}$ (costs neglected).

(b) $\zeta \to max$ (costs kept low).

Figure 8.7: Node distribution in different utility classes.

partners. So many partners are served exclusively and most actors are able to find a small amount of valuable partners. Here, approximately 60% of members are located in the 20% utility class. This demonstrates that applying the benefit-cost factor ζ seems to be a feasible concept to 'balance' networks, i.e., to distribute gained utility in a fair manner among community members.

8.5 Conclusion

In this chapter we introduced an opportunistic approach that models the formation process in open collaborative networks on the Web. We discussed and utilized two well-known theories, reciprocity and structural holes, and introduced a combined utility-based model. Several simulations demonstrated the applicability of this model and its basic properties.

CHAPTER 9

Discovering and Managing Member Compositions in Crowds

Crowdsourcing is an increasingly used model to outsource certain tasks to be carried out by external experts on the Web. Especially when lacking experience or expertise with certain task types, crowdsourcing offers a convenient way to receive instant support. In this Section, we introduce an in-house enterprise crowdsourcing model, which leverages the crowdsourcing concept and transfers it to traditional organizations. Here, a company's staff is considered a crowd that – besides its regularly assigned tasks – can also receive tasks from colleagues from other departments and across hierarchical structures. The aim is to offer instant support and utilize free capacities throughout a large organization more efficiently. In our work, we describe this concept and supporting mechanisms in context of an agile software development use case. However, in contrast to usually crowdsourced microtasks, complex software architectures usually consist of tens and hundreds of connected modules that can be potentially crowdsourced. These technical dependencies between modules require active coordination and interactions between crowd members that process the single artifacts. Hence, technical dependencies of artifacts result in social dependencies of *collaborating crowd members* that create them. In order to efficiently discover *member compositions* based on artifact dependencies, we introduce an indexing and discovery approach based on subgraph matching. Typically, assigning tasks to well-rehearsed teams results in more reliable task processing, faster results, and higher quality of work. We evaluate our approach in terms of system scalability and overall applicability by mining and analyzing the popular SourceForge community. We show that our approach of member composition discovery is feasible in terms of scalability and quality of discovery results. Our findings deliver important input for the design and implementation of supporting information systems for future large-scale collaboration platforms.

9.1 Introduction

The collaboration landscape has changed dramatically over the last years by enabling users to shape the Web and availability of information. While in the past collaborations were bounded to intra-organizational collaborations using company-specific platforms, and also limited to messaging tools such as e-mail, it is nowadays possible to utilize the knowledge of an immense number of people participating in collaborations on the Web. The shift toward the Web 2.0 allows people to write blogs about their activities, share knowledge in forums, write Wiki pages, and utilize social platforms to stay in touch with other people. Task-based platforms for human computation and crowdsourcing, enable access to the manpower of thousands of people on demand by creating human-tasks that are processed by the crowd. Human-tasks include activities such as designing, creating, and testing products, voting for best results, or organizing information. This paradigm is increasingly utilized by today's companies to enable scalable distributed software development by outsourcing tasks to external experts when lacking particular in-house expertise or development time.

A wide variety of software development processes, models, and managing techniques have been proposed in the last decades [Ghezzi et al., 2002]. They emerged from the need for structuring and managing work in large-scale teams. Though these models have led to accepted standards for software development approaches, they typically focus on task and artifact dependencies but widely neglect social aspects of software development. We argue that both technical *and* social dependencies are of paramount importance, especially for today's agile paradigms. **Technical dependencies** are mainly induced by artifact dependencies, e.g., software modules, being created and developed. Various methods exist to identify these dependencies, such as *call-graphs* that show links of components during run-time. Once relations between artifacts, e.g., software components and modules of a planned system, have been identified, we use the concept of flexible, collaborative activities to describe work to be performed, i.e., artifacts to be created in order to build the designed system. Thus, activity dependencies reflect artifact dependencies. By dividing work in separate pieces, activities can be distributed across crowd members. However, as a consequence, initially decomposed work eventually needs to be composed and integrated again to obtain the final result. This process requires substantial coordination effort among crowd members, i.e., artifact creators. Due to this collaborative aspect it deems to be more beneficial to pick crowd members that are familiar with each other's working style. Managing these **social dependencies** is typically not covered by today's crowdsourcing platforms that hardly support interactions between single members. Moreover, crowd members typically need to interact frequently if they work on related artifacts in order to make sure that their work is aligned and will finally fit together. Even rigidly defined interfaces between separately developed software modules usually cannot avoid this inherent need for communication. As described by Conway's Law [Conway, 1968] decades ago, *because software modules interact, this creates a similar need of interaction among software developers* [Trainer et al., 2005].

The Principle of Collaborative Enterprise Crowdsourcing

In this work, we discuss the foundational model for a collaborative enterprise crowdsourcing environment. Essentially, this model takes the usual concept of crowdsourcing on the Web and applies it to an enterprise collaboration context. The basic properties of this environment are that (i) *preselected experts* (ii) can be *flexibly involved* in *ongoing work* by outsourcing them generally encapsulated tasks, however (iii) still giving them the means to coordinate their work (*interaction tool support*). Especially the last point (active coordination and collaboration between crowd members) is a strong requirement to deal with complex tasks within enterprises compared to crowdsourcing microtasks on the Web.

The notable point here is that crowdsourcing is driven by company staff who are all hired experts and who collaborate with each other in context of the crowdsourced tasks. However, it is not the intention to replace existing working collaboration models but to establish this collaborative crowdsourcing model in parallel to traditional models in order to better utilize company resources. Open Source Software (OSS) development shares some of these properties with the envisioned environment, especially that interactions between team members happen largely online, participation of people is (often) more flexible, work fully distributed, and participants are only loosely coupled to management. We argue that these properties are vital to an enterprise environment in which – besides regular work – instant help and support can be flexibly gathered, even across organizational boundaries and hierarchical structures. Of course, in order not to render existing management structures useless, this model is applied only for a small fraction of the work time, i.e., indeed for instant help and support and tasks being carried out in free cycles only.

Research Challenges

The overall research challenge is to design an infrastructure that supports the described environment. Previous research – which is referred to, however not in scope of this work – tackles the following challenges:

- Flexible integration of humans in collaborative environments using service-oriented concepts, such as *Human-Provided Services (HPS)* [Schall and Dustdar, 2010; Schall et al., 2008b; Skopik et al., 2011d]
- Dynamic creation of trust networks using collaboration monitoring techniques [Skopik et al., 2010a, 2012b], which allows us to automatically detect social relations between crowd members.

The current Section deals with challenges on top of a flexible trust-based collaboration network, especially dealing with the following concerns:

- Indexing approach for social compositions utilizing emerging trust networks.
- Query and discovery mechanisms using subgraph matching.

Contributions

In this work, we describe an approach to manage socio-technical dependencies in collaborative crowdsourcing environments. Here we introduce *socio-computational crowds* based on the concept of human computation (e.g., see [Gentry et al., 2005]) that was established in the context of crowdsourcing. Traditional crowdsourcing environments lack to a great extent collaborative aspects. Our approach leverages social networks to support the collaborative processing of crowdsourced tasks. For that purpose we focus on enabling interactions and establishing social relations in crowdsourcing environments, as well as on the management of reliable crowd member compositions. In particular, this work deals with the following contributions:

- *Socio-Computational Crowdsourcing Fundamentals.* We motivate the need for interactions between crowd members that usually act only isolated on today's platforms. Our approach utilizes service-oriented computing paradigms to build a loosely coupled crowd community of experts within large-scale enterprises.

- *Dependency Management Approach.* We highlight a conceptual model to enable sophisticated management of social dependencies in aforementioned collaborative environments. In detail, we discuss the implementation of a feature-based discovery model that enables the efficient search for crowd member compositions.

- *Structure and Dynamics of the SourceForge Community.* We study real community data and extract common properties that build the basis for further evaluations of our approach in terms of scalability and applicability. Furthermore, we discuss the limits of our approach.

9.2 Basic Building Blocks of Enterprise Crowdsourcing

We start with a motivating scenario that introduces basic concepts. Furthermore, we discuss foundational building blocks, that enable seamless human participation in service-oriented architectures (SOA) and account for social implications.

Agile Software Development Processes

Let us consider a software development project that is executed in an agile manner [Martin, 2002]. Such projects do not follow a top-down approach where all requirements are gathered upfront. In agile software engineering, typically all engineering cycles are performed iteratively. Each iteration consist of (i) requirements analysis, (ii) design, (iii) implementation, (iv) integration, and (v) testing. Many of today's software companies have adopted this methodology since it allows to consider emerging customer requirements. For example, a customer may request new features during the development of a software product. These new requirements are then evaluated in a new development iteration.

Here we consider a large scale software development project comprising various team members including project managers, requirements engineers, software architects, developers, and testers. We assume that the overall project is structured roughly in three essential phases.

1. The initial requirements are gathered, basic technologies are evaluated and crucial design decision are made by a small core team. The core team usually consists of senior people (e.g., chief architect, senior developers) that define the most essential features of the

software. For example, suppose the team needs to design and implement an extensible framework that offers an API, tools, libraries, etc.

2. The next step is the implementation of the core features and the API, which is done by the senior developers. At this stage, the team is still kept small and typically geographically collocated because people need to collaborate and discuss extensively. This phase may already be performed according to the agile process where all before mentioned phases are part of an iteration. Depending on the size, resources, and duration of the project, an iteration may take several weeks. Also, we assume that certain automation in the development process already takes place such as automatic builds, generation of test reports, check of test coverage, etc.

3. In the third phase of the project we assume a transition of the centralized team to a distributed team. At this point, the core features of the framework have already been defined, documented, and implemented. Team members located in different departments or company sites have the ability to use and extend the core framework. This can be accomplished by designing and implementing components using the framework's API. Again, certain steps of the process are automated such as executing batch scripts, copying files, deployment of software components, or testing of individual components. However, we assume that various features are implemented and tested by distributed team members where certain steps require tight collaboration between members. For example, the tester may report a problem to the developers who in turn need to discuss changes in the source code.

In the following we discuss a use case scenario with the focus on socio-computational crowds embedded in distributed collaborations.

Use Case

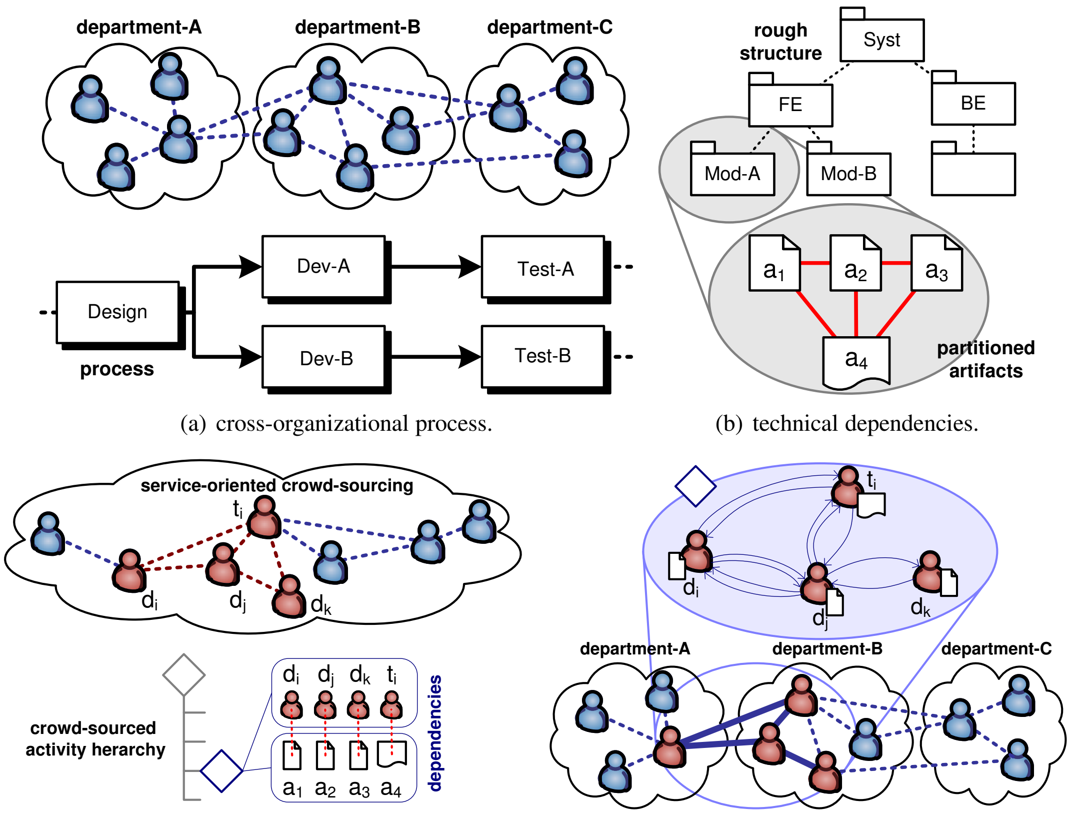
Basic Setting. A motivating scenario, including involved actors and artifacts, is depicted in Figure 9.1. Here, in Figure 9.1(a) people in geographically distributed departments of a large-scale enterprise collaboratively participate in a software engineering project. In particular, the basic steps and responsibilities are modeled in a process that spans numerous organizational units. Using today's modern Web 2.0 approaches and service-oriented architectures [Schroth and Janner, 2007] people have all tools at hand to flexibly collaborate on the Web. In our scenario, employees of a multi-national organization are connected through a social trust network (reflected by the dotted lines between people).

Single departments take over the responsibility for particular tasks of the global process. We assume that this company applies an agile software development approach, where artifacts are designed, implemented, tested and subsequently refined in short iterative cycles [Martin, 2002]. After creating a rough overview, e.g., a UML package view of the software framework that is going to be developed, each module (e.g., see *Mod-A* in Figure 9.1(b)) is partitioned in its (atomic) artifacts that can be processed by individuals. For instance, a software module is decomposed in three classes or submodules a_1, a_2, and a_3. Each of these artifacts is going to be created by a software developer and accompanied by a software tester who creates test cases and a final report a_4.

After technical dependencies have been thoroughly identified (see Figure 9.1(b)), artifact structures are mapped onto matching social structures. In other words, tightly coupled technical artifacts typically require a lot of coordination and integration effort, thus, can be best performed by people that know each other's working style from previous collaborations (according to *Conway's Law* [Conway, 1968]).

Several research challenges arise when assigning responsibilities for creating technical artifacts to people:

- What if for given technical artifact compositions, there are no matching social structures, i.e., well-proven team compositions, in the responsible department?



(a) cross-organizational process.

(b) technical dependencies.

(c) social dependencies.

(d) collaborative crowdsourcing.

Figure 9.1: Motivating scenario: (a) geographically distributed departments execute a cross-organizational software development process; (b) a rough package view and detailed artifact partitions capture technical dependencies; (c) technical relations enable the discovery of matching social compositions and creation of crowdsourcing activities; (d) interactions during collaborations approve and update registered social networks.

- What if the responsible department has no free capacities while other departments still have?
- What if there are only matching social structures across the borders of numerous organizational units?

Enterprise Crowdsourcing. Today, large-scale enterprises are facing the challenge of effective management and exploitation of the employees' knowledge and resources. Usually, expertise in numerous fields is available but often this knowledge is neither discovered nor captured somewhere. Since decades, researchers invent models and approaches to overcome that issue [O'Leary, 1998]. Enterprise crowdsourcing [Vukovic, 2009] follows a different path. Here, employees are encouraged to actively participate in a *private crowd environment*, where they offer their skills and expertise to other departments of the company. On the one side, rare expertise can be discovered and, on the other side, free capacities in one department can be used to tackle peak loads in other departments by **outsourcing especially non-critical activities**. Thus, enterprise crowdsourcing deems to be an elegant new paradigm to harness people's capabilities in a flexible and far more effective manner compared to rather static traditional cross-department collaborations. Service-orientation is the ideal means to realize such private crowds (i.e., not open to the public), because members can be dynamically discovered, are loosely coupled and thus composed at run-time, and flexibly assigned to activities.

In our use case, a majority of the employees participate in the described *socio-computational crowd environment* as shown in Figure 9.1(c). In contrast to the widely used notion of crowdsourcing, we do not use this environment to outsource tasks to single individuals only. We rather outsource compositions of problems, e.g., the creation of technical artifacts having interdependencies, to compositions of crowd members. Therefore, one major challenge is to identify reliable social compositions, i.e., groups of crowd members that have proven their reliable and successful collaboration behavior before. Once identified (as highlighted in Figure 9.1(c)), a collaboration activity is created and artifacts a_1 to a_4 (or templates respectively), as well as crowd members d_i, d_j, d_k, and t_i assigned.

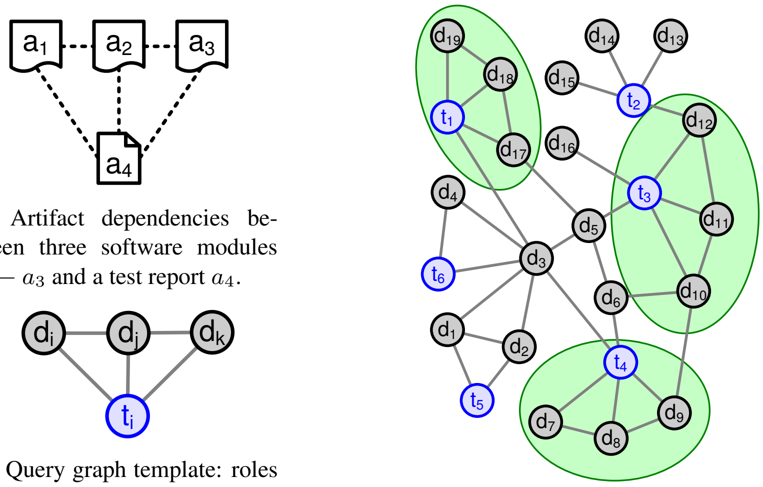
Finally, these members create, modify, and extend the required (or given) artifacts. This activity requires an extensive amount of interactions (Figure 9.1(d)) to coordinate work, align artifacts, and ensure a smooth integration of software modules later on. Today, a wide range of service-oriented communication, coordination, and collaboration tools are available for crowd members. Furthermore, since interactions are performed through these tools, they can be observed and even analyzed. Thus, valuable information about real collaboration behavior and spirit can be obtained and used to approve and update the social trust network between crowd members. This network is the basis for an effective future discovery of reliable crowd member compositions.

9.3 Feature-based Discovery Model

The basic aim of our approach is to discover reliable actors for given activity templates created from artifact structures. Typically, in a collaborative environment, activities are performed by compositions of actors. Such compositions, e.g., teams, consist of actors with potentially different roles having social dependencies. For instance an activity might demand for three software developers specifying and implementing software modules, and a software tester who assesses the created artifacts. The fundamental challenge is to find one (or even more) compositions of actors who can perform this activity. Therefore, this group should have the following properties: (i) the capabilities of humans match the required roles, and (ii) social relations follow artifact relations to enable reliable communication and coordination between dependent actors.

Feature-based Search

From an analytical point of view, this problem is described by the *concept of induced subgraph isomorphism* [Eppstein, 1999]. In order to measure the substructure similarity between a target graph (here: distinct parts of a social network) and a query graph (here: a template describing required actor composition properties), different models [Papadopoulos and Manolopoulos, 1999; Yan et al., 2005, 2006] have been proposed; in particular (i) physical property-based, (ii) feature-based, and (iii) structure-based. In this section we describe a feature-based approach since it allows to introduce some degree of fuzziness in the discovery process and is not as complex to compute as structure-based models; and thus, fit better to large-scale networks. In short, commonly searched elementary features of subgraphs are extracted, for instance, the number of software developers in a team, the degree of cross-links between them, or their field of experience and expertise. Whether a subgraph in the social network matches a query graph is determined by the number of matching features. Given that similarity definition, each frequently



(a) Artifact dependencies between three software modules $a_1 - a_3$ and a test report a_4.

(b) Query graph template: roles and social relations between three software **(d)**evelopers and one **(t)**ester.

(c) Discovering query graph structures in a large-scale crowd network (consisting of **(d)**evelopers and **(t)**esters.

Figure 9.2: Feature-based search approach: (a) from artifact dependencies (b) a social graph query is inferred and (c) matching subgraphs are discovered.

active team (i.e., highly cross-linked subgraph in a social network) is represented by a feature vector, where its single components represent the frequency of common predefined features. The distance between the query graph and a potential match in the social network is measured by the distance between corresponding feature vectors.

Figure 9.2 depicts an illustrative example. In Figure 9.2(a) technical dependencies between three concrete software modules $a_1 - a_3$ and a test case a_4 are identified. These artifacts and their dependencies are mapped to a social structure consisting of software developers and a tester in Figure 9.2(b). Using this generic subgraph template, appropriate instances are discovered in the large-scale social network as highlighted in Figure 9.2(c). Notice, in this example, software modules are decomposed in segments so that each of them can be processed by exactly one crowd member.

Approach Outline

The feature-based composition model is applied in context of other concepts to keep track of the dynamics in flexible socio-computational crowd environments. The whole approach depicted in Figure 9.3 is described as follows:

1. *Social Network Definition:* Single crowd members register their personal profiles, consisting of interests, expertise and usual roles; but also their relations to well-known collaboration partners (e.g., FOAF fragments including `knows`-relations). Through semantically reasoning over FOAF snippets a large-scale network model is created and periodically updated.

2. *Online Monitoring:* Since members communicate over Web services, all SOAP interactions can be logged and analyzed. Metrics, such as interaction frequency and density, describe the strength of predefined social relations between collaborating members. This information is utilized to confirm registered profiles; for instance, to discover defined but not approved relations.

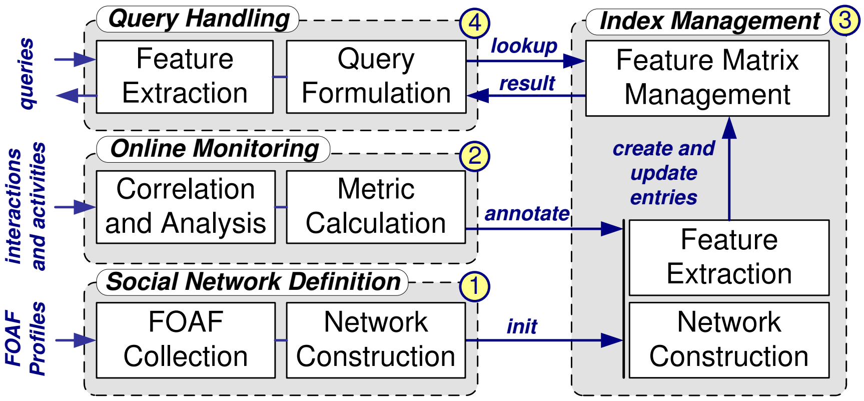


Figure 9.3: Approach to feature-based discovery: (1) network definition, (2) network annotation, (3) feature index management, and (4) query handling.

3. *Index Management:* Periodically analyzing member interactions and performed activities enables the identification of regularly used collaboration patterns (i.e., frequently applied compositions of actors) with commonly requested features.

4. *Query Handling:* Whenever a query is issued, features of the query graph are extracted (e.g., number of roles, degree of cross-linkage) and an approximate query is created based on features only.

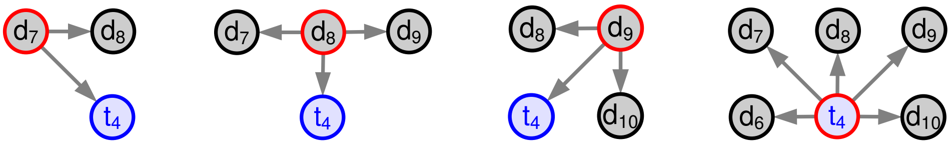
Detailed Formulation

We proceed with the definition of an analytical model that supports the discovery of social compositions considering artifact dependencies.

Social Network Definition

Crowd members define their individual FOAF profiles where each one represents a single graph fragment $G_F = (V_F, E_F)$ of the social network. In particular crowd members define which other members they `know`, i.e., collaborate with. For instance, members d_7, d_8, d_9, and t_4 define structures as shown in Figure 9.4. Of course, they only know their direct neighbors, but by reasoning over these RDFstructures, the whole graph can be constructed (being a part of the network depicted in Figure 9.2(c)). We define the whole social network G to be composed of single fragments as given in Eq. 9.1. Notice, manually declared relations are the basis for determining which interactions need to be monitored and analyzed to confirm social structures and enrich relations with expressive interaction behavior metrics.

$$G = (V, E) = (\bigcup_{\forall G_F} V_F, \bigcup_{\forall G_F} E_F) \tag{9.1}$$



(a) d_7's view (b) d_8's view (c) d_9's view (d) t_4's view

Figure 9.4: Aggregating individual FOAF profiles allows the construction of large-scale social networks.

Interaction Mining and Social Trust Inference

Since we apply SOAP interceptors and access layers for Web service calls, we are able to capture directed interactions I that carry some payload p, such as support requests and work delegations, between pairs of crowd members (u, v) in context of an activity a (from a set of activities A). These interactions are realized as standard SOAP calls as explained in detail in our previous work [Skopik et al., 2010a].

$$I = \{(u, v, a, p) \,|\, u, v \in V, a \in A\} \tag{9.2}$$

This enables us to annotate FOAF `knows`-relations with evidence-based interaction metrics $m_i(u, v)$. Common metrics are, for instance, request reciprocity, availability, interaction intensity, frequency and uniformity [Skopik et al., 2011a]. These metrics that characterize the behavior of people can be interpreted in terms of trustworthiness and dependability [Mui et al., 2002; Skopik et al., 2010a]. We exemplarily define the following ones[1]:

Reciprocity $recpr$. A typical social behavior metric is reciprocity [Mui et al., 2002] that here reflects the ratio between obtained and provided support in a community. Let $I_{REQ}(u, v)$ be the set of u's sent support requests to v, and $I_{RES}(u, v)$ the set of u's provided responses to v's requests. Then we define reciprocity in $[-1, 1]$ as in Eq. 9.3; hence, 0 reflects a balanced relation of mutual give and take.

$$recpr(u, v) = \frac{|I_{RES}(u, v)| - |I_{REQ}(u, v)|}{|I_{RES}(u, v)| + |I_{REQ}(u, v)|} \tag{9.3}$$

Availability $avail$. This metric describes u's availability for v's requests, i.e. the amount of answered requests. The result of Eq. 9.4 is a value in $[0, 1]$.

$$avail(u, v) = 1 - \frac{|I_{REQ}(v, u)| - |I_{RES}(u, v)|}{|I_{REQ}(v, u)|} \tag{9.4}$$

Responsiveness $resp$. This metric (cf. Eq. 9.5) describes the response behavior of a crowd member. In particular in today's highly dynamic businesses fast responses on support requests are a key success factor. Here, we calculate the average of response times as a measure for someone's commitment. For that purpose, the average time span t between single interactions $\imath$, i.e., requests and corresponding responses, is determined.

$$resp(u, v) = \frac{\sum_{I_{RES}} t(\imath_{REQ}(u, v)) - t(\imath_{RES}(u, v))}{|I_{RES}(u, v)|} \tag{9.5}$$

Social Trust τ. Finally, these metrics are interpreted and aggregated to reflect the strength of FOAF relations by one score. For that purpose either an arithmetical approach that simply weights normalized metrics, or a rule-based approach that truly interprets metric values (i.e., the trustworthiness of interaction behavior) according to a given rule-base is applied [Skopik et al., 2010a]. This operation is represented by $\otimes$ in Eq. 9.6.

$$\tau(u, v) = \langle (recpr(u, v), avail(u, v), resp(u, v)), \otimes \rangle \tag{9.6}$$

Index Management

The basic aim is to identify frequently occurring actor compositions, e.g., teams, collaborating in context of activities. For that purpose an indexing algorithm uses two data sources: (i)

[1] Two of them have been defined in a similar manner in Chapter 4. We explain them here again for the sake of completeness, since all three are used in the evaluation section of this chapter.

interaction logs I from the monitoring facilities to discover strong social connections and dependencies, and (ii) *activity structures A* to identify member compositions working in the same context. Since crowd members interact in context of activities, the indexing algorithm traverses a list of finished activities and analyzes interactions between assigned members. Hence, social network structures are identified relying on evidence through interaction mining, rather than manually defined connections from FOAF profiles only.

We harness an index structure, referred to as the feature-graph matrix M [Yan et al., 2006], to facilitate the feature-based registration of actor compositions. Each row of the matrix M corresponds to a registered subgraph G_i of the social network, while each column corresponds to a feature f_j being indexed. Each entry x_{ij} records the number of the embeddings (values respectively) of a specific feature f_j in the registered subgraph G_i (Table 9.1).

	$\mathbf{f_A}$	$\mathbf{f_B}$	$\mathbf{f_C}$	$\dots$	$\mathbf{f_j}$
$\mathbf{G_1}$	x_{1A}	x_{1B}	x_{1C}	$\dots$	x_{1j}
$\mathbf{G_2}$	x_{2A}	x_{2B}	x_{2C}	$\dots$	x_{2j}
$\mathbf{G_3}$	x_{3A}	x_{3B}	x_{3C}	$\dots$	x_{3j}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\mathbf{G_i}$	x_{iA}	x_{iB}	x_{iC}	$\dots$	x_{ij}

Table 9.1: Feature-graph matrix index.

Subgraph Features. One of the most challenging parts in our approach is the definition of meaningful and significant features. For our given software engineering scenario, we defined the features given in Table 9.2 to describe significant properties of social subgraphs and compositions of crowd members.

feature name	feature description
num_nodes	Number of crowd members in the subgraph
num_{role}	Number of {role}, e.g., developers, testers
num_links	Number of links in the subgraph
avg_nodedeg	Average degree of nodes (distribution of links)
num_hubs	Number of nodes linked to all other nodes
avg_trust	Average trust score between members
maturity	Number of activities that approved the composition

Table 9.2: Crowd features for software development scenarios.

Index Updates. The index requires frequent updates to reflect the real social networks of highly dynamic collaboration environments. Thus, whenever an activity is finished, involved members and their relations are analyzed and either a new subgraph G_i added to the index, or an existing one (consisting of the same members) is updated in terms of changed features. Indexing algorithms are discussed in the next section in greater detail.

Approximate Discovery and Query Relaxation

In order to discover appropriate crowd members to process a set of technical artifacts, queries on the social network, in detail on the feature graph index, are issued. Typically a composition of technical artifacts G_A, as shown in Figure 9.2(a), is given; for instance a class diagram, entity-relationship diagram or higher-level dependency model. This input is mapped to a social dependency graph (Figure 9.2(b)) using a scenario-specific bijective mapping between user roles and artifact types, i.e., different types of technical artifacts require distinct roles of crowd members. For instance, a software module requires a software developer, or a test report demands for a tester. Furthermore, the type (implements, uses, calls, etc.) and degree of coupling between technical artifacts determine the required strength of social relations between assigned crowd members; for instance, the development of tightly coupled modules results in higher coordination effort and thus requires distinct social trust τ.

Algorithm 3 shows the details. First, for each technical artifact $x \in V_A$ of a particular type ($x.Type$) a template for a crowd member with a corresponding role ($u.Involvement.Role$) is created (Line 3). Then, for all existing dependencies from $x \in V_A$ to any $y \in V_A$, a social trust value τ between corresponding actors $u, v \in V_Q$ of the social query grah G_Q is set (Line 6).

Algorithm 3 Build query graph from technical dependencies.

```
1:  **Input:** artifact dependencies $G_A = (V_A, E_A)$
2:  **Output:** social dependency query graph $G_Q = (V_Q, E_Q)$
3:  for each $x \in V_A$ do                ▷ create one social node per technical artifact
4:      $u \leftarrow$ createNodeTemplate($x.Type$)
5:      addNode($V_Q$, $u$)
6:  for each $x \in V_A$ do      ▷ link social nodes according to technical artifact dependencies
7:      $u \leftarrow$ getCorrespondingNodeTemplate($V_Q$,$x$)
8:      for each $y \in V_A$ do
9:          $v \leftarrow$ getCorrespondingNodeTemplate($V_Q$,$y$)
10:         $\tau(u, v) \leftarrow$ mapFromTechnicalDependencyType($x$,$y$)
11: return $G_Q$
```

Once, the query graph has been constructed, the features that characterize the required social composition need to be extracted. Using the metrics of Table 9.2 and the query graph given in Figure 9.2(b), one would get the features $num\_nodes = 4$, $num\_d = 3$, $num\_t = 1$, $num\_links = 5$, $avg\_nodedeg = 2.5$, $num\_hubs = 1$ (here, avg_trust and $maturity$ are omitted). A query that is basically an *ordered set* of these criteria, is issued by comparing these query graph properties with the feature index. This is an approximate querying mechanism; i.e., because features of the query graph match features of the index graphs does not mean that query and result graphs are structurally identical. However, the more features match the higher is the probability that resulting graphs match the query.

In case no matching subgraph is found, two mechanisms can be applied:

- *Query Relaxation [Yan et al., 2006]:* is a mechanism that subsequently removes less important features from the query until matching results are found; for instance, num_nodes

must hold in order to have a workforce of appropriate size, but the distribution among roles $num\_\{roles\}$ is relaxed.

- *Subgraph Fusion:* if no single subgraph satisfies the query, a result can be constructed out of two subgraphs, e.g., construct a larger workforce composed of two smaller teams. In that case, overlapping nodes between these two subgraphs ensure proper interlinks in the final result. A heuristic is required to realize this feature, which is out of scope of this work.

Notice, our approach focuses only on the discovery of appropriate and approved social compositions for a given problem. However, further negotiation with actors are required prior to starting a collaboration, e.g., accounting for their free capacities, personal constraints, and rewarding.

Activity-based Indexing Algorithm

The further presented indexing algorithm (Algorithm 4) basically operates on a list of finished activities. First, for each activity the list of involved members is extracted from the activity structure (Line 3). Since members are sometimes officially involved in activities but actually inactive (since other actors take over their responsibilities), we remove all crowd members whose amount of performed actions is less than ϑ_{inv} (Line 4). Then, mutual knows-relations from

Algorithm 4 Basic periodic index update.

```
 1: Input: list of activities $A$, index matrix $M$
 2: for each $a \in A$ do                          ▷ extract one social graph per finished activity
 3:     $G_i \leftarrow$ createGraph($a.InvolvedMembers$)
 4:     for each $u \in V_i$ do                    ▷ remove officially involved but actually inactive members
 5:         if $|a.actions(u)|$ / $|a.actions(any)| < \vartheta_{inv}$ then
 6:             removeNode($G_i$,$u$)
 7:     for each $u \in V_i$ do
 8:         for each $v \in V_i$ do
 9:             if $\exists$ knows$(u, v) \wedge \exists$ knows$(v, u)$ then
10:                 $\tau(u, v, )$ = updateMetrics($u, v, E_i$)      ▷ annotate strength of social ties
11:                 if $\tau(u, v, ) > \vartheta_\tau$ then
12:                     $E_i = E_i \cup \{e(u, v)\}$
13:     addOriginActivity($G_i, a.id$)             ▷ further annotations to $G_i$
14:     setUpdateTimestamp($G_i$)
15:     $F \leftarrow$ extractFeatures($G_i, featurelist$)      ▷ subgraph registration in index $M$
16:     /*... apply alternative template-based filtering here (see next) */
17:     if $G_i \in M$ then
18:         updateIndexEntry($G_i, F, M$)
19:     else
20:         createIndexEntry($G_i, F, M$)
```

FOAF profiles determine potential social dependencies in context of the finished activity. However, if these social relations actually have been relevant, i.e., interactions along these relations have been performed, needs to be verified by metrics gathered from interaction mining. Thus, if social trust $\tau(u, v)$ is above a certain threshold ϑ_τ, this relation is considered important for the success of this collaboration. Otherwise, i.e., no interactions occurred between u and v or trust is low, the relation is not considered for the social subgraph G_i (Line 10). Afterwards, beginning with Line 13, G_i is 'tagged' with two further properties: (i) its origin(s) (activity ids), and (ii) a timestamp to capture the up-to-dateness of this composition. Finally values of predefined features ($featurelist$) are extracted that describe this graph (see Table 9.2) and either a new entry in the index matrix created or an existing entry updated (Line 15). An entry already exists, if it contains exactly the same members (identified by their `uris`) and connections between them. This means that there are no two subgraphs with identical nodes/edges in the index.

Template Mechanism. If features of recognized social compositions largely differ (e.g., in terms of roles and trust values) than numerous unique compositions are registered. In order to avoid that phenomenon (i.e., to be able to provide various alternatives to a given search query), subgraphs can be registered according to predefined common *social graph templates* (Algorithm 5). That means the features of a social subgraph G_i are tested against predefined templates, i.e., compositions that frequently occur or whose features have been recognized as highly requested (e.g., from query log analysis [Radlinski and Joachims, 2005]) while further properties are neglected.

Algorithm 5 Template-based filtering for index updates.

```
1: Input: list of graph templates $T$, social subgraph $G_i$
2: $F \leftarrow$ extractFeatures($G_i, featurelist$)
3: for each $t \in T$ do                                  ▷ check each registered template
4:     $featurematch \leftarrow 0$
5:     for each $f \in F$ do
6:         if match($t, f$) then                            ▷ single feature match
7:             $featurematch \leftarrow featurematch + 1$
8:     if $featurematch/numFeatures(t) \approx 1$ then
9:         /* add/update $G_i$ in $M$ */          ▷ $G_i$ matches (nearly) all template feature
```

Algorithm 5 shows the basic principle. First, all features F of a social subgraph G_i are extracted. Then, these features are tested against a list of predefined temples T that characterize the desired features (Line 5). The algorithm counts the number of matching features of a subgraph G_i and template $t \in T$. If, finally, most features of a template t are covered by G_i, this subgraph is added to the index (Line 8). Otherwise, the recognized composition is not of interest and skipped.

Pruning. Of course, social compositions of crowd members that do not frequently perform activities need to be removed from the index. For that purpose, we apply a self-pruning index mechanism [Goodman, 2002] (Algorithm 6) that recognizes whether the last update of an index entry due to a finished activity is older than a predefined threshold (ϑ_{age}). In that case, the outdated G_i is removed (Line 3). However, *mature* and long-term settled compositions that were involved in an extraordinary amount of activities (ϑ_{act}) may remain in the index (Line 4).

Algorithm 6 Self-pruning index mechanism.

```
1: Input: index matrix $M$
2: for each $G_i \in M$ do                              ▷ inspect each registered subgraph
                                                ▷ if composition not used recently (aged out)
3:     if currentTime - getUpdateTimestamp($G_i$) $> \vartheta_{age}$ then
                                                ▷ ... and if no long-term settled composition
4:         if getNumOriginActivities($G_i$) $< \vartheta_{act}$ then
5:             deleteIndexEntry($G_i, M$)                         ▷ remove $G_i$ from index
```

9.4 Evaluation and Discussion

Since we have not applied our approach and implementation in large-scale environments but like to evaluate its feasibility and applicability (e.g., of the indexing algorithm), we need to utilize either synthetic data or data sets from other domains. In order to set up a realistic setting, we study the properties of free and open source software development communities [Crowston and Howison, 2005; Howison et al., 2006], such as the *SourceForge Research Data Archive* (SRDA) [Van Antwerp and Madey, 2008] to create a synthetic collaboration network graph consisting of users with certain roles having interrelations and being involved in sets of activities.

SRDA Data Set Properties

The SourceForge[2] platform provides a free management and development infrastructure for open source projects, including CVS repositories, mailing lists, discussion forums, and task management and tracking – just to name a few. The SRDA consists of collected data from the SourceForge community. We analyze SRDA tables[3] from January 2011 regarding (i) *project tasks and structures*: `project_assigned_to` and `project_task`, (ii) *discussion forums:* `forum` and `forum_group_list`, (iii) *artifacts:* `artifact`, `artifact_message` and `artifact_category`. Furthermore, we correlate user actions and user roles using prior work from [Christley and Madey, 2007b]. Here, user actions also include CVS operations. Analyzing and harnessing real data, combined with synthetic data under feasible assumptions allows us to create a realistic scenario of large-scale crowd-sourced software development.

Activity Structures

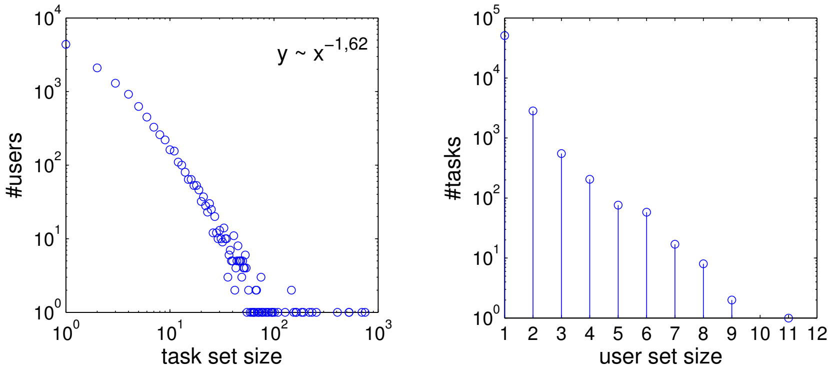
Analyzing SRDA enables us to create a realistic setting in terms of activity sizes (i.e., number of involved members[4]), number of activities where one user is involved at the same time, as well as the distribution of activities among projects. We create one activity in our system for each single

[2] SourceForge: `http://sourceforge.net`

[3] SRDA Online Wiki: `http://srda.cse.nd.edu/mediawiki`

[4] Notice, we removed all anonymous users (with id=100) from the data set, because there is no way to distinguish between different anonymous users and this hinders us to create a realistic social network later.

task in the data set. In order to show the feasibility of this approach, we study the properties of tasks and show some characteristics in Figure 9.5.



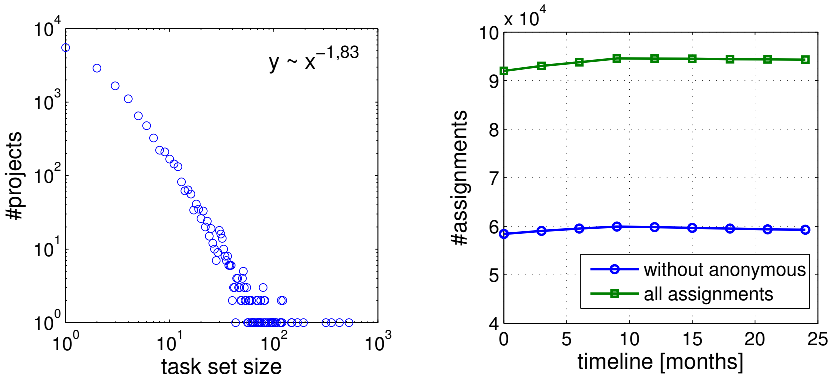
(a) user effort.

(b) task size.

Figure 9.5: Community structures on task level: (a) user effort in terms of involved tasks; (b) task size in terms of involved users.

In detail, Figure 9.5(a) visualizes the typical number of tasks (task set size) where a single user is involved. From 11 915 users, 4 369 are involved in only one task, 2 098 in 2 tasks, and 1 297 in 3 tasks. There are around 45 users who are involved in more than 50 tasks. The rest is distributed as shown in the figure. Figure 9.5(b) shows a different perspective, in particular the number of tasks having a particular user set size (i.e., team) assigned. Obviously, most tasks (50 758 of 54 500) are performed by single users only, while there is no task where more than 11 users are involved. Due to that reason, we further analyze team structures on project levels by aggregating tasks.

Figure 9.6(a) shows the distribution of tasks on project level. From 14 285 analyzed projects,



(a) project size.

(b) temporal evolution.

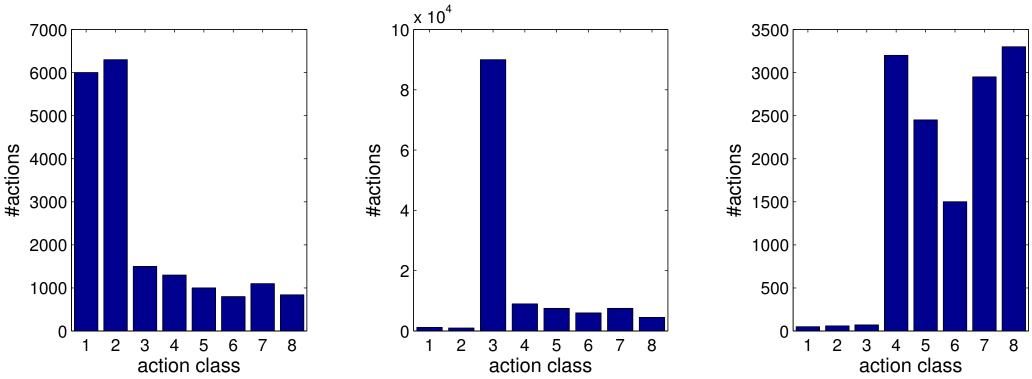
Figure 9.6: Community structures on project level: (a) project size in terms of number of tasks; (b) temporal evolution of created task assignments in a 18 months timeframe.

there are 5 518 that consist of only one task, 2 901 with 2 tasks, and 1 669 with 3 tasks. On the other side, the largest project consists of 527 tasks. Figure 9.6(b) demonstrates that the number of newly created task assignments over an observed timeframe of 18 months remains nearly constant at around 59 000 and 94 000 (including assignments to anonymous users) respectively.

We conclude that for around 12 000 users, we will create around 4 000 activity entries (corresponding to tasks having more than 1 user in SRDA) consisting of 2 to 11 users, and assigned to 500 projects (having more than one task with more than one involved user) with a distribution as discussed before.

Role Mining

Implicit user roles describe a user's typical collaboration behavior and focus based on performed actions. Roles are an inherent concept of our composition discovery approach. However, since user roles are *not* explicitly set in SourceForge (neither statically in user profiles nor dynamically at project setup), we need mechanisms to identify user roles by analyzing the SRDA data set. For that purpose we study experiments first performed by [Christley and Madey, 2007b] (and described in greater detail in [Christley and Madey, 2007a]). In short, users (of the largest and most active projects) are clustered based on actions they perform on the SourceForge platform. Here, we only study a small subset of available actions, in particular, (1) create a new forum message, (2) create a follow up forum message, (3) modify a project (adding/removing members, changing permissions etc.), (4) check out source code from CVS repository, (5) add source code to CVS repository, (6) remove code, (7) modify code, and (8) update code. There are much more actions available[5], including tracking bugs, releasing files and submitting patches, feature requests, artifacts, todos etc. A clustering approach is used to identify similarities of action distributions among users. Finally, around 40 member custers are identified (depending on the thresholds in the clustering algorithm [Christley and Madey, 2007b]), which describe around 7 different roles. The two largest clusters, containing software users - who check out software and



(a) sw user. (b) project admin. (c) sw developer

Figure 9.7: Action distribution in clusters of various user roles.

[5] [Christley and Madey, 2007b] identified 29 different action types that enable user role mining.

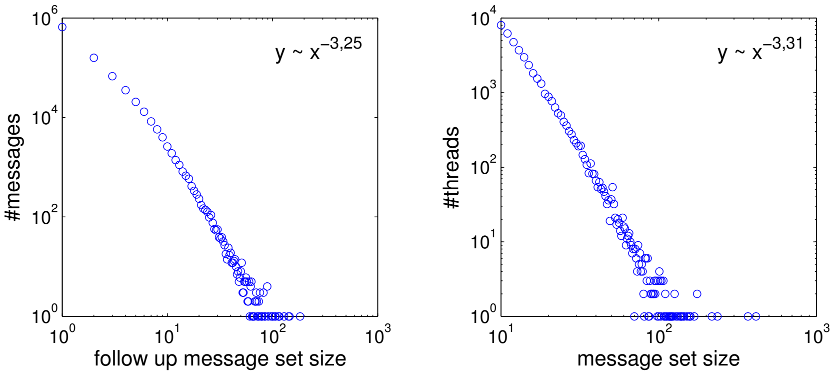
discuss bugs and features - and project managers, contain around 75% of all users. However, there are clearly other roles, such as pure software developers, task managers and bug reporters [Christley and Madey, 2007a]. Figure 9.7 visualizes action distributions (i.e., the number of performed actions by all users in the corresponding cluster) for the three major roles software user, project administrator, and software developer.

We conclude that in our crowdsourcing environment around 41% of members will be software users (i.e., general experts that drive the further development by testing prototpyes and discussing issues in forums), 34% project administrators, and 17% pure developers. The rest (8%) does not fit into these roles.

Interaction Data

Interactions on SourceForge may take place over various channels. However, not all of them can be easily captured or are included in the SRDA. For instance, while subscriptions to mailing lists are part of the data set, actually sent e-mails are not. Therefore, we utilize two different interaction channels: (i) forum messages, and (ii) artifact messages.

Threaded Forum. The forum allows to discuss missing features, bugs or further development in a hierarchical manner. That means to every message a so-called follow-up message can be posted. The data set consists of 3 167 605 captured messages (and 5 191 629 if including posts by anonymous users respectively). We count 494 302 distinct forum posters. Figure 9.8 shows some basic characteristic of the forum. In particular, Figure 9.8(a) visualizes the distribution of the set size of follow-up messages on the same hierarchical layer; in other words, the number of posts attracted by one particular message. Figure 9.8(b) shows thread sizes by aggregating all messages on all levels beginning with the top message in one message tree. From 985 838 messages that have at least one follow-up (i.e., not being an unanswered post), 11 857 have equal or more than 10 follow-ups, from that set 915 messages have more than 25 follow-ups. Regarding the thread size, we investigated thread structures with at least 10 messages in order to



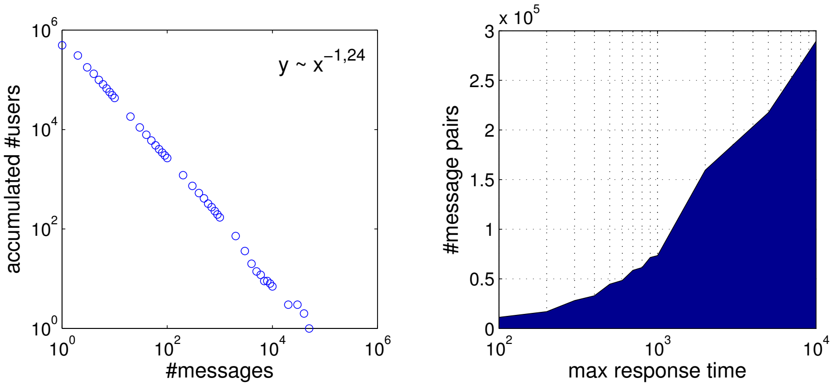
(a) follow up messages.

(b) thread size.

Figure 9.8: Forum structure: (a) number of follow up posts to one message; (b) typical number of messages in one thread.

capture serious discussion effort. There are 40 830 threads that fulfill this criterion. From this set 3 887 threads consist of more than 25 messages, 509 have more than 50 messages, and 125 more than 75. The largest thread consists of 414 messages (without anonymous posts). We conclude that in SourceForge discussions are typically focused, i.e., a message does not have dozens of follow-ups, however, deep thread structures emerge, e.g., due to controversy on certain issues.

Figure 9.9 provides information about general user behavior. In detail, Figure 9.9(a) shows the number of posts per user. For better readability[6] users are aggregated on the y-axis. That means, the figure shows on the y-axis the number of users who posted more than the given amount of messages on the x-axis. There are 43 302 users who posted more than 10 messages, 6 001 with more than 50 messages, and 2 679 users with more than 100 messages. Thus, serious analysis is only possible with a small fraction of the whole user base. We furthermore investigated response times in Internet forums. Since we do not know which message is a question and which one a comment, we simply capture the differences of timestamps between each message and its follow-up (iff there is one). Figure 9.9(b) reflects the responsiveness of forum posters by categorizing message pairs according to their posting times. Response times are given in seconds. Note the logarithmic scale.



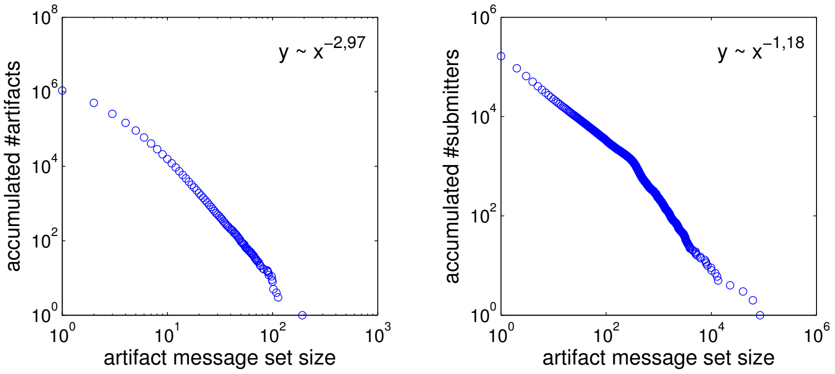
(a) user effort.

(b) response time (seconds).

Figure 9.9: User involvement in forums: (a) number of messages posted by users; (b) typical response times of users (up to 10 000 seconds).

Artifact Messages. A second source of interaction data are artifact messages. SourceForge users can ‘attach’ messages and comments to artifacts of various types. Overall, there are 1 076 517 artifacts where at least one message is assigned in the studied time span. In sum, 164 429 distinct users submitted 2 305 702 messages (2 991 274 if including artifact messages by anonymous users). Figure 9.10(a) shows the accumulated number of artifacts for various message set sizes. As shown in this cumulative diagram, there are around 15 654 artifacts with equal to or more than 10 messages attached (thus, really collaboratively processed artifacts), and 96 artifacts with equal to or more than 50 messages. More than 50% of artifacts have only one message assigned.

[6] the number of posts per user highly varies from 1 to 90 071



(a) amount of message assignments.

(b) submitter effort.

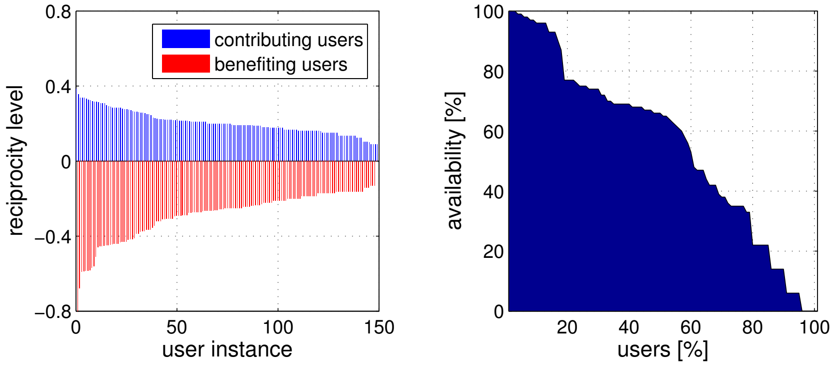
Figure 9.10: Artifact message and submitter distributions.

Figure 9.10(b) displays a cumulative submitter perspective; i.e., the number of message submitters on the y-axis that submitted more messages than the message set size given on the x-axis. For instance, from the full set of 164 429 distinct submitters, 22 214 submitted equal or more than 10 artifact messages, 3 306 users more than 100 messages, and 225 users more than 1 000 messages. Approximately 43% of all users submitted only a single artifact message. Since artifact messages are sparsely distributed over a large set of artifacts, we assume an artifact message as a kind of point-to-multi-point communication, where each single message addresses all users who submitted messages to the same artifact. This approach, instead of a point-to-point model, enables us to infer still meaningful metrics that quantify users' relations in our proposed socio-computational crowdsourcing model.

Collaborative Crowd Environment Setup

After extensively analyzing a real Web-based large-scale software development environment, we create a synthetic crowdsourcing environment that reflects attributes from SRDA in terms of (i) *structural properties* such as number and sizes of activities, and (ii) *dynamic interactions* such as exchanged messages and performed task assignments. Furthermore, we assign the roles of software developers, software users (aka testers) and project administrators to the crowd members using exactly the same distribution as given in the SRDA data set. The following section deals with metric calculation as defined in Section 9.3. In particular, we demonstrate the calculation of *reciprocity*, *availability* and *responsiveness* based on SRDA data.

Reciprocity. The SourceForge data set offers two valuable properties to calculate reciprocity, i.e., the amount of obtained support from the community compared to the amount of provided support. First, tasks are created by one person and may be assigned to another one. In total, there are 11 915 users involved in task processing; 10 613 of them assign tasks to others than themselves. In sum, 94 308 task assignments have been captured. By removing assignments from anonymous users and also self-assignments, only 15 207 tasks remain to be analyzed. Finally, there are 5941 unique user pairs, i.e., a mapping from one assigner to another assignee.



(a) reciprocity distribution.

(b) availability distribution.

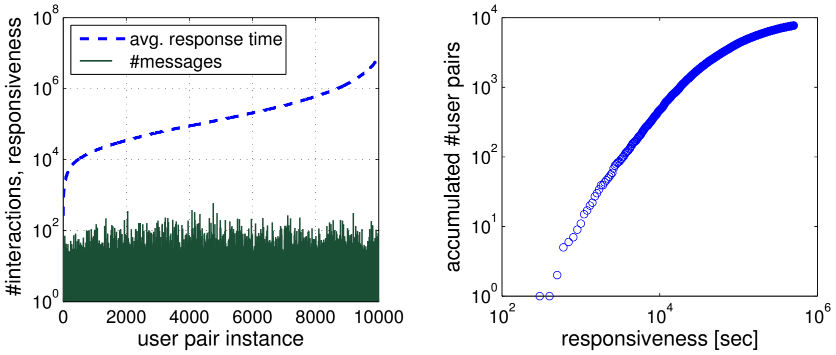
Figure 9.11: Metric distribution: (a) *reciprocity*, (b) *availability*.

From that amount, 3151 distinct users assign tasks to 5335 other users.

A second source of information is the Internet forum. Our assumption is that a top post, i.e., the first post in a thread, is usually a question or support request (except announcements which however are mostly unreplied). All further replies are attempts to address this request. Thus, the poster of the first message obtains some help from the community, while repliers provide some support. Using the forum mining algorithm from [Skopik et al., 2009b][7], we calculate each user's contribution score and subsequently reciprocity. If a user obtains more support from the community, i.e., posts many top messages but replies less or assigns many tasks but processes only a few, than this value is negative. Figure 9.11(a) shows the distribution of reciprocity for top-150 contributing and top-150 benefiting users. Notice, project administrators have negative reciprocity by nature (e.g., it is part of their role to assign tasks to others). Thus, we calculate reciprocity values only between members with the same roles, in particular, software users, project administrators, and software developers. Overall, there are few users who benefit very much, while on the other side the majority of users contribute a little. Accumulating all reciprocity values of all users results in ≈ 0.

Availability. Users submit artifacts of various types to the SourceForge platform. However, they can be assigned to and finally closed by other users. Thus, an artifacts ownership and relation to users is described by a triple $\langle submitted\_by, assigned\_to, closed\_by \rangle$. In the studied time span, there are 422 443 artifacts that have not been submitted by, assigned to, or closed by anonymous users (in sum, there are 2 292 054 artifacts on the platform). From that set, 301 095 artifacts are submitted by and assigned to two different users. We assume that artifacts who are submitted and closed by the same user ($\approx 11.9\%$) are processed successfully (at least if the status is set to closed), and thus the assigned user was available to process a given task. In case the

[7] Notice, since there are no frequently discussing distinct user pairs, we calculate reciprocity not for personal relations between two particular users, but for users with respect to the whole community. This further reduces the computational complexity from around $O(n^2)$ – one potential link between every pair of nodes – to $O(n)$. However, in our proposed socio-computational crowd environment, interactions would not be performed in a public manner but addressed directly to receivers, thus, metrics would be calculated for personalized links.



(a) responsiveness v.s. #messages.

(b) responsiveness distribution.

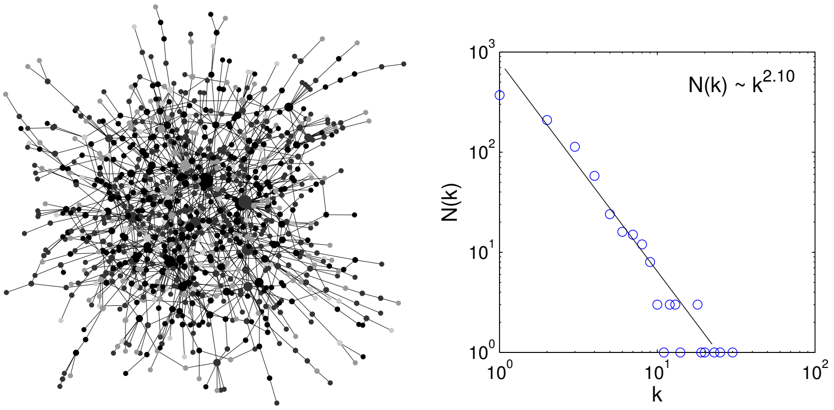
Figure 9.12: Metric distribution: (a) *responsiveness v.s. number of messages*, (b) *average response time*.

submitter and closing user are two different persons ($\approx 70.3\%$), the assigned user was available for collaboration if s/he assigned at least one message to the corresponding artifact. For the rest, we consider artifacts with state closed or pending as success, others as failed[8]. Figure 9.11(b) depicts the *minimum* availability in percent for a certain amount of users (in percent from the whole population). Note, we only considered a small user base of 250 users, since for larger populations availability could not be calculated seriously. There is simply not enough data to prove availability with smaller amounts of interactions. Basically, in this figure there are two buckles, one where availability drops from around 95% to around 75% and another one where the same happens from around 65% to 40%. This effect seems to be caused by highly varying numbers of captured interactions (i.e., artifact assignments). We conclude that for the availability metric, a larger pool of personal interaction is required to calculate stable and reliable values.

Responsiveness. In order to calculate this metric for SourceForge community members, we utilize once more the Internet forum. In contrast to reciprocity calculation, here we can partly use posts from anonymous users too. In particular, we consider how fast distinct users reply to anonymously posted messages. We proceed as follows: First, all final answers (i.e., unreplied follow-up posts) from anonymous users are removed from the data set. Then, we determine unique[9] user pairs, count how often they replied to each other's posts, and study those, who had at least 10 interactions. Doing so, there remain 10 189 user pairs of which 1 780 had at least 25 interaction, 491 at least 50 interactions, and 131 equal or more than 100 interactions. The number of posts and the average of response times do virtually not correlate (Pearson correlation coefficient of -0.024). Figure 9.12(a) shows average response time values and corresponding number of exchanged messages. Figure 9.12(b) deals in more detail with the responsiveness values of the fastest replying users. Notice, here we record user pairs. Thus, there are 11 user pairs who have an average response time below 1 000 seconds ($\approx 17min$), 471 with response

[8] Of course, we cannot prove that these assumptions are correct in all cases, however, given the massive amount of data we argue that the trend of handling data this way is feasible.

[9] Posts from anonymous users are assigned to *one* virtual user. This method does not distort the measurement since we calculate average values only (and do not sum up contributions such as for reciprocity).



(a) graph view.

(b) degree distribution.

Figure 9.13: Created social trust network (reduced view for 1 000 nodes).

times below 10 000 seconds ($\approx 2.8hrs$), and 4 307 below 100 000 seconds (slightly above one day). We do not use other data sources for responsiveness calculation, such as task assignments, since processing times (and thus response times) highly vary according to task complexity.

Social Trust. Various approaches exist to infer social trust values from captured behavior metrics. Here we use normalization and weighted average, i.e., metric values are normalized, e.g., to fit the interval [0,1] and are than combined with predefined[10] weights/impact. Alternatively, more sophisticated approaches, including rule-based aggregation and fuzzy set theory [Skopik et al., 2010a] can be applied. The final outcome is a scale-free social trust network, as visualized in Figure 9.13. The graph's properties regarding degree distribution and connectivity match attributes of common collaborative communities as investigated by [Reka and Barabási, 2002]. In particular, degree distributions of such networks follow power laws with degree exponents between 2.1 and 2.5; here, for our network created from SourceForge data we calculated an exponent of 2.10, thus matching overall expectations.

Experiments and Results

The basic aims of these experiments are (i) to demonstrate the *feasibility* of our dependency management approach, and (ii) to measure the *scalability* and performance of the prototype implementation.

Scale of Social Network Management

The first step in our evaluation approach is to *create a synthetic network that has realistic properties* (in terms of size, node degree, member roles, activity involvements etc.) extracted from real community data as discussed before. For that purpose we use the following model:

1. Create 12 000 user instances (**nodes**).

[10] all three metrics ($recpr$, $avail$, $resp$) use same weights, i.e., $\frac{1}{3}$

2. Assign **roles**: software users (41%), project admins (34%), software developers (17%), undefined (8%).
3. Create list of **activities** with 4 000 entries.
4. Create **links** between users according to SourceForge data set, i.e., distribution of link strength between users of same roles.
5. Partition users in **groups** (subgraphs) consisting of 2 to 11 nodes; $\approx \frac{1}{3}$ having 2-3 users, $\approx \frac{1}{3}$ having 4-6 users, $\approx \frac{1}{3}$ having > 7 users.
6. **Assign groups** to activities.
7. Structure activities in **projects**.
8. Create **FOAF profiles** that are processed by our system.

Properties of the resulting graph are summarized in Table 9.3. Metric definitions follow common standards as further explained by the utilized software tool *Network Analyzer*[11]. This overview shows the complexity[12] of typical networks that our algorithms will have to cope with.

In order to utilize the created graph[13] for the evaluation of our prototype implementation for managing and discovering member compositions, we create a FOAF profile[14] for every single user, containing his/her name (`lastname`), role (`Group`), relations to collaboration partners (`knows`) extended with a trust value, and activity involvement (`currentProject` and `pastProject` respectively). This process is supported by the *Jena Semantic Web Framework*[15].

social network metric	**value**
number of nodes	12 000
number of edges	23 976
connected components	1
clustering coefficient	0.001
network radius	6
network diameter	9
network centralization	0.016
characteristic path length	5.381
avg. number of neighbors	3.996

Table 9.3: Complexity of created network.

[11] Network Analyzer: `http://med.bioinf.mpi-inf.mpg.de/netanalyzer/`

[12] Notice, the mentioned software tool required 7 969 seconds to calculate the given properties on a Pentium D with 3.0 GHz.

[13] Here we use the *Java Universal Network/Graph Framework (JUNG)* available at `http://jung.sourceforge.net`.

[14] The related FOAF concept is highlighted with a `typewriter` font.

[15] JENA: `http://jena.sourceforge.net`

Basic Graph Construction and Indexing

We discuss performance aspects of our proposed indexing approach as defined by Algorithm 4. Notice, this algorithm operated on top of the previously created synthetic collaboration network. The performance of interaction log analysis, metric calculations and trust inference is not in scope of this work but has been extensively studied in [Skopik et al., 2010a]. Here we traverse a list of 4 000 activities and index social compositions by determining the predefined features[16] of Table 9.2.

We run the indexing algorithm two times: (i) *Activity-centric indexing*: applies the algorithm as defined where for each finished activity the features of the corresponding social composition are extracted. (ii) *Node-centric indexing*: refers to the creation of one virtual activity per user and adding all node's neighbors in advance. In fact, here the surrounding social network of each node (and from its individual perspective), potentially emerged from numerous activity involvements, is indexed, rather than social compositions from a third person's view.

For **activity-centric indexing** the result are 4 000 social compositions (one per activity), which are grouped according to equality of features. Figure 9.14(a) depicts the size of clusters and their distribution. Here, the 25 most common compositions are clustered and labeled (A-Y), while further 268 of 4 000 compositions do not fit into these schemes (lable Z). Table 9.4 shows the top-5 occurring social compositions and some relevant features, i.e., num_{role}[17], num_links, avg_nodedeg, avg_trust[18]. Notice, that smaller compositions occur of course more often.

For **node-centric indexing** results are depicted in Figure 9.14(b). In this case, a higher amount of activities (i.e., 12 000 matching to the number of nodes) is analyzed. The results show compositions (A'-Z') from each user's point of view (as given, for instance, in Figure 9.4). In other words, the results are the decompositions as reflected by the single FOAF profiles of network members. Since social structures from a node's perspective include only direct relations and hence are not as complex[19] as in the previous case (again cf. Figure 9.4), only 256 (of 12 000) social compositions do not fit into one of the created 25 clusters. This situation would

type	num_{role}	num_links	avg_nodedeg	avg_trust	count
A	2 su	1	1	[0.25,0.5[	446
B	1 su, 1 pa	1	1	[0.25,0.5[	398
C	2 su	1	1	[0.5,0.75[	347
D	2 su, 1 pa	2	1.5	[0.25,0.5[	302
E	1 su, 1 sd, 1 pa	2	1.5	[0.5,0.75[	289

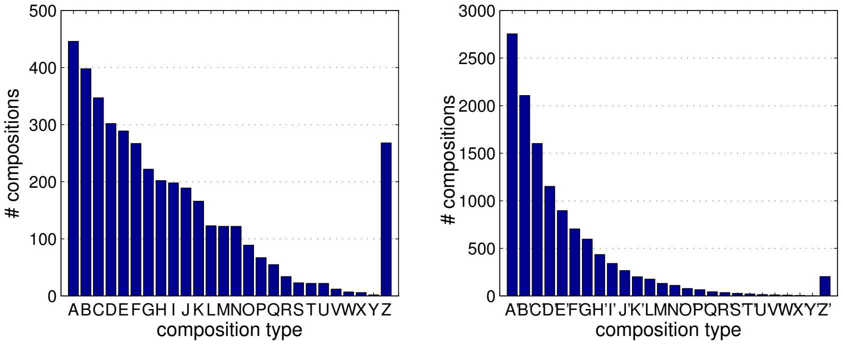
Table 9.4: Most frequent compositions (in sum $\approx$ 44.5% of all compositions).

[16] Notice, we skip the calculation of the *maturity* feature since social compositions are rarely reapplied in SourceForge and thus in our synthetic model. However, we argue that when using our system from the beginning, this feature will have major impact, especially when querying for well-trained and frequently applied social compositions

[17] su=software user, sd=software developer, pa=project admin

[18] measured in intervals, otherwise compositions could not be grouped

[19] Notice, here the average degree of a node is 3.996.



(a) activity-centric indexing.

(b) node-centric indexing.

Figure 9.14: Composition detection frequency for different indexing approaches.

significantly change, if we did not only consider a node's direct relations, but also neighbors of neighbors (a kind of recommendation mechanism.)

We discuss the **performance** of these approaches from an abstract perspective, in terms of number of invocations of core services (infrastructure discussed later), and number of database accesses, because real run-time performance varies depending on numerous impacts, such as network latency, processor speed and load, and memory consumption. Table 9.5 compares the effort of applying the two discussed indexing mechanisms. In general, activity-centric indexing (cf. count(i)) results in less but larger (partly unique) compositions compared to node-centric indexing (cf. count(ii)) which produces more, but less complex social compositions.

After performing this set of experiments with basic indexing, we motivate the application of template-based indexing, as discussed earlier in this work, which allows to:

- group social compositions more efficiently, e.g., group actually different but in terms of features quite similar compositions.
- better support the discovery of frequently requested compositions (e.g., matching to often issued queries)

	measurement	**count (i)**	**count (ii)**
infrastructure	#ActivityServiceAccesses	4 000	12 000
	#UserDBAccesses	59 818	47 592
	#SocialTrustNetworkAccesses	23 976	23 976
	#FOAFProfileAccesses (public)	12 000	12 000
	#RegistryDBAccesses	25 888	51 684
data	#CompositionsRegistered	25 888	51 684
	#IndexNetworkCoverage (percent)	93.3%	98.3%

Table 9.5: Performance in terms of number of calls and resulting index complexity for (i) activity-centric indexing and (ii) node-centric indexing.

- cut the long tail of the distribution in Figure 9.14 which provides little value to most users but causes significant management overhead (e.g., size of index)
- recognize sub-subgraphs, e.g., a social composition which performed a set of activities could be split into two frequently requested compositions and thus registered multiple times.

Template-based Composition Indexing

We pick the most recognized subgraph types A-J (see top-five in Table 9.4) as templates and re-run indexing Algorithm 4, but additionally apply Algorithm 5. We configure the indexing process to categorize recognized subgraphs by comparing the four features $num\_\{role\}$, num_links, $avg\_nodedeg$, and avg_trust as used in Table 9.4 before. Figure 9.15 visualizes the results. Given our data set, we can recognize around 11% of social compositions with only one template, around 44.5% with 5 templates, and already 71.4% with 10 templates. We tested template-based indexing with up to 25 different templates with which we can categorize 93.3% of all occuring social compositions.

Submatching. Until now, we categorized subgraphs G_i (extracted from activities) which exactly match a given template's t features; e.g., $num\_links(G_i) = num\_links(t)$. Now, we further evaluate so called *submatches*. Here, we test each G_i against all available templates in order to recognize if a given composition fulfills *at least* a template's features; e.g., $num\_links(G_i) \geq num\_links(t)$. As a result the more complex a recognized social composition is the more often it can be decomposed in simpler submatches and registered multiple times. Figure 9.16 shows the results[20] for this experiment. In general, 2 825 social compositions (from 4 000 activities) are registered only once (because those consist only of 2 nodes and therefore cannot match further relevant subgraphs); 497 compositions ($\approx$ 12.5%) are registered twice. Interestingly, the number of social compositions for higher number of registrations is not

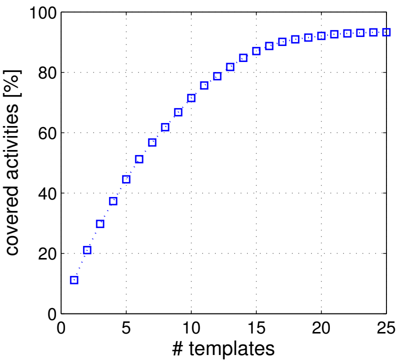


Figure 9.15: Amount (in percent) of covered activities compared to number of applied templates.

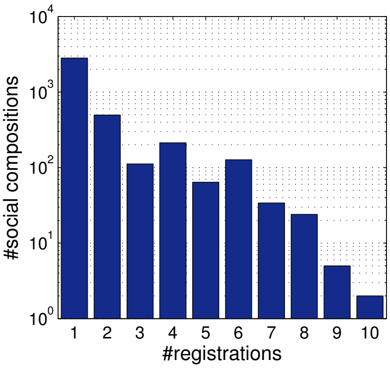
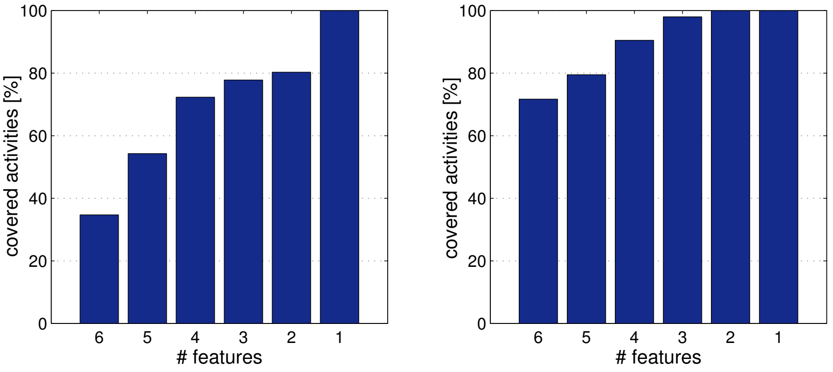


Figure 9.16: Number of social compositions that match multiple templates (leading to several registrations).

[20] Notice the logarithmic scale on the y-axis.



(a) exact matching.

(b) submatching.

Figure 9.17: Amount of covered activities depending on number of compared template features.

decreasing monotonic. For instance, there are more social compositions that are registered four times than three times. The reason is, that there are numerous compositions consisting of three nodes (u, v, w), which can be additionally decomposed in three subgraphs consisting of only two nodes ((u, v), (u, w), (v, w)). Thus, in sum, a three-node-composition here is registered four times as long as other features (such as avg_trust) do match either.

Template Complexity. Finally, we investigate the impact of template structures and complexity on the indexing process. The question is, how does the amount of social compositions, which is recognized by the indexing process, change for differently complex template definitions. For that purpose we set the number of utilized templates to 10 (fixed) and just vary the number of impacting features. The whole feature list consists of (in this order): num_nodes, $num\_\{role\}$, num_links, $avg\_nodedeg$, num_hubs, and avg_trust. We begin with testing each social composition against each single template by considering all features. Then, we remove the last feature in the feature list (beginning with avg_trust) and run the indexing process again. Figure 9.17 visualizes the results for indexing with exact matches and indexing with submatches. As expected, the more features are tested the lower is the amount of matching subgraphs. Furthermore, submatching leads to at least partly indexed structures of more complex social compositions as well as to multiple registrations. Therefore, even larger social compositions become indexed to some extent and the number of covered activities (and their corresponding social compositions respectively) in Figure 9.17(b) does not increase as sharply as in Figure 9.17(a). A careful *tradeoff between accuracy* of indexed compositions *and amount of covered activities* by applying a fixed number of templates (here: 10) must be considered. For instance, when indexing by exact matching the num_nodes feature, 10 templates are sufficient to cover 100% of activities (matching $num\_nodes = [2, 11]$), because there are no corresponding social compositions with more than 11 users (compare Figurer 9.5(b)). However, in that case one would only be able to query for social compositions with an appropriate size but neglecting other features, such as roles and trust links.

Composition Discovery and Query Relaxation

For the final tests we create a reference index through template-based indexing with exact matching and using 25 templates which consider *all* features as defined in Table 9.2. This index covers 87% of activities and their corresponding social compositions respectively. Furthermore we define three test queries $\mathbf{Q_1}$, $\mathbf{Q_2}$, and $\mathbf{Q_3}$. Query $\mathbf{Q_1}$ is designed to deliver a large amount of results, i.e., querying only for a pair of software users (su) with an average trust relation. Query $\mathbf{Q_2}$ looks up three users, with one hub node to which two other nodes are connected. Query $\mathbf{Q_3}$ defines the search for three users of different roles which are well interconnected. All three query definitions use features num_nodes, $num\_\{role\}$, num_links, $avg\_nodedeg$, num_hubs, and avg_trust. Notice, $maturity$ is neglected here (symbol $*$ means any) which is not sufficiently reflected in our test data set. Table 9.6 shows the details.

feature	$\mathbf{Q_1}$	$\mathbf{Q_2}$	$\mathbf{Q_3}$
num_nodes	2	3	3
num_{roles}	2 su	2 su, 1 pa	1 su, 1 sd, 1 pa
num_links	1	2	3
avg_nodedeg	1	1.5	2
num_hubs	2	1	3
avg_trust	[0.25,0.75]	[0.25,0.75]	[0.25,0.75]
maturity	$*$	$*$	$*$

Table 9.6: Definition of test queries $\mathbf{Q_1}$, $\mathbf{Q_2}$, $\mathbf{Q_3}$.

Issuing the queries as given delivers result sets of sizes 446, 102, and 89 (Table 9.7). If the discovered social compositions are currently not available (e.g., actors are involved in other activities, index is outdated), a query relaxation mechanism can be applied in order to find further social compositions that potentially match one's needs. This mechanism gradually removes features from the query and therefore extends the result sets. Table 9.7 provides an overview of the results for the given queries. The first column describes the relaxation factor $\gamma = \frac{\#appliedFeatures}{\#allFeatures}$ ($\gamma = 1$ means all features are used in the query). Then two columns, a lower bound and an upper bound, describe the size of the result set. There are two limits because depending on which feature is removed first, the sizes of result sets vary. For instance, removing num_nodes from $\mathbf{Q_1}$ has virtually no effect, since $num\_\{roles\}$ already strictly define that two software users are required. However, removing avg_trust dramatically extends the result set (here: from 446 to 767). The last row in the table summarizes how many more results are generated when neglecting half of the query features ($\gamma = 0.5$). In general, more complex queries profit more from query relaxation because they become considerably simpler. Again, query relaxation introduces fuzziness to the query results and its usefulness heavily depends on the use case, i.e., how strictly results must match to an issued query.

Web Services-based Implementation

Service Infrastructure. From the technical (and implementation) point of view, we use a wide variety of state-of-the art Web (service) technologies. Crowd members (and even their relations)

γ	$\lfloor \mathbf{Q_1} \rfloor$	$\lceil \mathbf{Q_1} \rceil$	$\lfloor \mathbf{Q_2} \rfloor$	$\lceil \mathbf{Q_2} \rceil$	$\lfloor \mathbf{Q_3} \rfloor$	$\lceil \mathbf{Q_3} \rceil$
1	446	446	102	102	89	89
0.833	446	767	102	222	89	194
0.667	446	890	102	289	194	226
0.5	446	890	212	356	201	289
0.333	987	2825	282	472	222	312
0.167	1237	2825	385	534	250	501
increase ($\gamma = 0.5$)	≈100%		107% - 183%		126% - 225%	

Table 9.7: Query relaxation results.

are represented by a large set of individual FOAF profiles that are initially created manually for (and ideally from) each node in the social network. Once registered, these profiles, especially `knows`-relations are then automatically updated based on captured interaction data as further discussed in [Skopik et al., 2011a]. Using FOAF, we link these nodes to a list of activity identifiers that reflect respective community members' involvements in certain activity instances. Activities are managed in an external Activity Web service that implements the activity model as discussed in the beginning of this thesis and in [Schall et al., 2008a], and is hosted on Axis2[21]. Since reading single FOAF profiles for each query for determining graph structures is time-consuming, we implement a dedicated SocialTrustNetwork Web service that manages an in-memory graph model (with a MySQL backend database to guarantee persistence) and whose *cache* is frequently updated from current FOAF profiles. Together with the the single FOAF profiles, the whole technical infrastructure is hosted on an *Apache Tomcat Web Server*[22]. Potential end-users, i.e., people who query for social compositions, can utilize this infrastructure through *Java Portlets* that are hosted on a Liferay Community Server[23]. These portlets support the configuration of the index management (e.g., defining template features), and the definition of queries (e.g., entering required subgraph feature values of indexed social compositions).

Performance Issues. Our implemented prototype uses a service-oriented backend with dedicated Web services for (i) periodically processing interaction logs to infer (and update) collaboration metrics and behavior (see details in [Skopik et al., 2010a], (ii) managing activities and joint task contexts (see details in [Schall et al., 2008a]), and (iii) managing subgraphs reflecting well-proven social compositions (i.e., create, read, update, delete index entries). Web services offer the great ability to facilitate interoperability and openness, since interfaces are well documented (as WSDL) and easily usable through established frameworks that can create stubs on demand and even perform flexible invocations. However, Web service calls are time-intensive (predominantly because to complex XML processing of SOAP messages) and often cause a major bottleneck in data-intensive applications. Furthermore, processing distributed FOAF profiles and creating/updating the social network graph is a time- and resource-intensive process. Numerous optimization methods can be applied to tackle performance issues, including caching of FOAF documents, bulk-transfers of profiles if applicable, and even bypassing Web services,

[21] Apache Axis2: `http://ws.apache.org/axis2/`

[22] Apache Tomcat: `http://tomcat.apache.org/`

[23] Liferay Community Server: `http://www.liferay.com`

e.g., here granting the indexing service direct access to back-end databases. Some optimization mechanisms for social network management, including WS caching mechanisms [Skopik et al., 2010a] and selective updates of relations [Skopik et al., 2011a], have been discussed in our previous work. Here, the technical infrastructure itself is not in the focus of this work.

9.5 Conclusion

Our work is motivated by the observation from related studies that there is a direct mapping between social dependencies and technical dependencies in large-scale software projects. We presented an approach that uses this knowledge to enable efficient *collaborative* crowdsourcing of software artifacts (and related activities respectively) by considering technical artifact dependencies to discover matching social compositions of crowd members. Social links are defined by members through their FOAF profiles and enriched with data gathered through an automatic interaction mining process. Using service-oriented architectures enables sophisticated interaction monitoring and thus the calculation of interaction metrics that describe collaboration behavior. We evaluated and proved our concepts using data from a real community, i.e., *SourceForge*. This approach ensures that we design our concepts and prototype implementation for scenarios having realistic properties and scale. Our work has important design implications for future frameworks and platforms supporting socio-computational crowdsourcing applications. We discussed typical properties of large-scale *Web-based* software development use cases and demonstrated the application of monitoring and mining techniques, and social composition indexing and discovery mechanisms.

We argue that our proposed approach of subgraph matching with additional query relaxation for discovering member compositions is a good tradeoff between performance and quality. Although the indexation and relaxation approach might lead to misses of best results for specific queries, the mechanisms can still quickly provide results even in large networks. This is essential if small update cycles are required (as typical for high dynamics in collaborative networks) in order to keep the index up-to-date.

CHAPTER 10

Conclusion and Future Research

What we call a *socio-computational system*, is also a *socio-technical system* in the wider sense. The notion of a socio-technical system in the area of computer science describes (according to [Fiadeiro, 2008]) *one of those software-intensive systems that involve complex interactions between software components, devices and social components (people or groups of people), not as users of the software but as integral players engaged in common tasks.* Some of the key properties of socio-technical systems are that they need to adapt to changes in order to deliver the intended services. However, social entities cannot be designed, such as software components, to comply with system rules. In particular, social components cannot be forced to adapt their (potentially malicious) behavior, but we can reconfigure the technical components of the system that they are interacting with; for instance, we do not control human behavior but aim to adapt interaction facilities (communication services) to (re-)shape their behavior.

Numerous research challenges, directions and critical objectives (see various lists in [Fiadeiro, 2008]) have been identified concerning the design of socio-technical systems. In this thesis, we studied the implementation of such systems using state-of-the-art social network approaches and Web technologies. According to [Fiadeiro, 2008], a major challenge is to define *a methodological approach for socio-technical systems where single modules are no longer strictly designed and implemented, but instead having the ability to connect during run-time and thus adapt to even complex situations.* We addressed this objective by using SOA paradigms of loose coupling, flexible discovery and run-time binding of components. Furthermore, there is the need to *develop methods, tools and theoretical foundations for socio-technical systems to control the evolution of the system; in particular, to simulate behavior of given system configurations, including their interactions.* We covered this requirement by applying the agile service hosting environment G2 [Juszczyk, 2011], which allows run-time adaptations according to system rules. Moreover, G2 also enables the simulation of complex interaction networks. A further objective is to *validate scenarios of possible configurations, define reconfiguration operations; so that systems can (self-)adapt to changes in the domain of operation.* We did this by hosting software components, so called avatars, that represent the human entity in the technical system and which can be dynamically reconfigured and adapted to shape the behavior of humans. Another point is that *social models of human components and associated notions of responsibilities, capabilities, duties, and roles* should be investigated. We applied existing social network models, such as FOAF [Brickley and Miller, 2010], and extended them to reflect human interests and capabilities. Another challenge is to invent a *mathematical model for interconnection and emergence; e.g., a framework that allow nodes to fell decisions in group formation processes, for instance, based on perceived utility and expectations.* In this thesis we introduced various group formation principles and designed analytical models describing such processes. A final objective of current research is to describe *formal models of reconfiguration for adaptability and higher level languages and mechanisms supporting that.* Here we discussed a novel social network query language that is applied in context of socio-computational systems.

Future research focuses more on the operational phase, i.e., the adaptation of the technical infrastructure at run-time to avoid performance degradations due to (human) misbehavior. Substantial effort regarding this issue has been already invested by [Psaier, 2012].

Bibliography

Adams, C. and Lloyd, S. (1999). *Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations*. Macmillan Publishing.

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

Albert, R., Jeong, H., and Barabasi, A.-L. (2000). Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382.

Allee, V. (2000). Reconfiguring the value network. *Journal of Business Strategy*, 21(4).

Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2003). *Web Services - Concepts, Architectures and Applications*. Springer.

Alonso, O., Rose, D. E., and Stewart, B. (2008). Crowdsourcing for relevance evaluation. *SIGIR Forum*, 42(2):9–15.

Altinel, M. and Franklin, M. J. (2000). Efficient filtering of xml documents for selective dissemination of information. In *International Conference on Very Large Data Bases (VLDB)*, pages 53–64. VLDB Endowment.

Artz, D. and Gil, Y. (2007). A survey of trust in computer science and the semantic web. *Journal on Web Semantics*, 5(2):58–71.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*.

Bird, C., Nagappan, N., Gall, H., Murphy, B., and Devanbu, P. T. (2009). Putting it all together: Using socio-technical networks to predict failures. In *International Symposium on Software Reliability Engineering (ISSRE)*, pages 109–119. IEEE.

Bird, C., Pattison, D., D'Souza, R., Filkov, V., and Devanbu, P. (2008). Chapels in the bazaar? latent social structure in open source projects. In *SIGSOFT Symposium on Foundations of Software Engineering*, pages 24–35. ACM.

Box, D. et al. (2004). Web services addressing (ws-addressing), w3c. http://www.w3.org/Submission/ws-addressing/.

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

Breslin, J., Passant, A., and Decker, S. (2009). Social web applications in enterprise. *The Social Semantic Web*, 48:251–267.

Brewington, B. E. and Cybenko, G. (2000). How dynamic is the web? *Computer Networks*, 33(1-6):257–276.

Brickley, D. and Miller, L. (2010). Foaf vocabulary specification 0.98. http://xmlns.com/foaf/spec/.

Bryl, V. and Giorgini, P. (2006). Self-configuring socio-technical systems: Redesign at runtime. *International Transactions on Systems Science and Applications (ITSSA)*, 2(1):31–40.

Burt, R. S. (1992). *Structural holes: The social structure of competition*. Harvard University Press.

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

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

Castano, S., Ferrara, A., and Montanelli, S. (2006). Matching ontologies in open networked systems: Techniques and applications. *Journal on Data Semantics V*, pages 25–63.

Cherns, A. (1976). The principles of sociotechnical design. *Human Relations*, 29(8):783–792.

Christley, S. and Madey, G. (2007a). Social positions at sourceforge.net. Technical report, Dept. of Computer Science and Engineering, University of Notre Dame.

Christley, S. and Madey, G. R. (2007b). Analysis of activity in the open source software development community. In *Hawaii International Conference on System Sciences (HICSS)*, page 166.

Conway, M. (1968). How do committees invent. *Datamation*, 14(4):28–31.

Cozzi, A., Farrell, S., Lau, T., Smith, B. A., Drews, C., Lin, J., Stachel, B., and Moran, T. P. (2006). Activity management as a web service. *IBM Systems Journal*, 45(4):695–712.

Crowston, K. and Howison, J. (2005). The social structure of free and open source software development. *First Monday*, 10(2).

D'Ambros, M., Gall, H., Lanza, M., and Pinzger, M. (2008). Analysing software repositories to understand software evolution. In *Software Evolution*, pages 37–67. Springer.

Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., and Pohl, K. (2008). A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*.

Diao, Y., Rizvi, S., and Franklin, M. J. (2004). Towards an internet-scale xml dissemination service. In *International Conference on Very Large Data Bases (VLDB)*, pages 612–623. VLDB Endowment.

Dobson, S. et al. (2006). A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(2):223–259.

Domingo, C., Gavaldà, R., and Watanabe, O. (2002). Adaptive sampling methods for scaling up knowledge discovery algorithms. *Data Mining and Knowledge Discovery*, 6:131–152.

Dwyer, C., Hiltz, S. R., and Passerini, K. (2007). Trust and privacy concern within social networking sites: A comparison of facebook and myspace. In *American Conference on Information Systems (AMCIS)*.

Eppstein, D. (1999). Subgraph isomorphism in planar graphs and related problems. *Journal of Graph Algorithms and Applications*, 3(3).

Euzenat, J. and Shvaiko, P. (2007). *Ontology Matching*. Springer.

Falk, A. and Fischbacher, U. (2006). A theory of reciprocity. *Games and Economic Behavior*, 54(2):293–315.

Fiadeiro, J. L. (2008). On the challenge of engineering socio-technical systems. In *Software-Intensive Systems and New Computing Paradigms*, pages 80–91. Springer.

Ford, M. et al. (2007). Web Services Human Task (WS-HumanTask), Version 1.0.

Gentry, C., Ramzan, Z., and Stubblebine, S. (2005). Secure distributed human computation. In *ACM conference on Electronic commerce*, pages 155–164. ACM.

Georgakopoulos, D. and Papazoglou, M. P., editors (2008). *Service-Oriented Computing*. MIT Press.

Ghezzi, C., Jazayeri, M., and Mandrioli, D. (2002). *Fundamentals of Software Engineering*. Prentice Hall, 2nd edition.

Giereth, M. (2005). On partial encryption of rdf-graphs. In *International Semantic Web Conference (ISWC)*, pages 308–322. Springer.

GMPG (2011). Xhtml friends network. http://gmpg.org/xfn/. Global Multimedia Protocols Group.

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

Gold, N., Knight, C., Mohan, A., and Munro, M. (2004). Understanding service-oriented software. *IEEE Software*, 21(2):71–77.

Golder, S. A. and Huberman, B. A. (2006). The structure of collaborative tagging systems. *The Journal of Information Science*.

Goodman, B. D. (2002). Accelerate your web services with caching. *IBM Advanced Internet Technology*.

Goyala, S. and Vega-Redondo, F. (2007). Structural holes in social networks. *Journal of Economic Theory*, 137(1):460–492.

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

Granovetter, M. S. (1973). The strength of weak ties. *The American Journal of Sociology*, 78(6):1360–1380.

Guare, J. (1990). *Six Degrees of Separation: A Play*. Vintage Books.

Guha, R., Kumar, R., Raghavan, P., and Tomkins, A. (2004). Propagation of trust and distrust. In *International World Wide Web Conference (WWW)*, pages 403–412.

Haller, A., Cimpian, E., Mocan, A., Oren, E., and Bussler, C. (2005). Wsmx - a semantic service-oriented architecture. In *International Conference on Web Services (ICWS)*, pages 321–328. IEEE.

Hang, C.-W. and Singh, M. P. (2010). Trust-based recommendation based on graph similarity. In *AAMAS Workshop on Trust in Agent Societies (Trust)*.

Herbsleb, J. D., Mockus, A., Finholt, T. A., and Grinter, R. E. (2001). An empirical study of global software development: Distance and speed. In *International Conference on Software Engineering (ICSE)*, pages 81–90. IEEE.

Hollenbach, J., Presbrey, J., and Berners-Lee, T. (2005). Using rdf metadata to enable access control on the social semantic web. In *International Semantic Web Conference (ISWC)*. Springer.

Howe, J. (2008). *Crowdsourcing: Why the Power of the Crowd is Driving the Future of Business*. Crown Business.

Howison, J., Inoue, K., and Crowston, K. (2006). Social dynamics of free and open source team communications. In *International Conference on Open Source Systems (OSS)*, volume 203, pages 319–330. Springer.

IBM (2005). An architectural blueprint for autonomic computing. *Whitepaper*.

Jøsang, A., Ismail, R., and Boyd, C. (2007). A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644.

Juszczyk, L. (2011). *Techniques for Automated Generation of Testbed Infrastructures for SOA*. PhD thesis, Vienna University of Technology.

Juszczyk, L. and Dustdar, S. (2010). Script-based generation of dynamic testbeds for soa. In *International Conference on Web Services (ICWS)*, pages 195–202. IEEE.

Kittur, A., Chi, E. H., and Suh, B. (2008). Crowdsourcing user studies with mechanical turk. In *Conference on Human Factors in Computing Systems (CHI)*, pages 453–456. ACM.

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

Kleinberg, J., Suri, S., Tardos, E., and Wexler, T. (2008). Strategic network formation with structural holes. *ACM Conference on Electronic Commerce*, 7(3):1–4.

Kossinets, G. and Watts, D. (2009). Origins of homophily in an evolving social network. *American Journal of Sociology*, 115(2):405–450.

Kruk, S. R., Grzonkowski, S., Gzella, A., Woroniecki, T., and Choi, H.-C. (2006). D-foaf: Distributed identity management with access rights delegation. In *Asian Semantic Web Conference (ASWC)*, pages 140–154.

Lara, R., Roman, D., Polleres, A., and Fensel, D. (2004). A conceptual comparison of wsmo and owl-s. In *European Conference on Web Services (ECOWS)*, pages 254–269. IEEE.

Liben-Nowell, D. and Kleinberg, J. (2003). The link prediction problem for social networks. In *International Conference on Information and Knowledge Management (CIKM)*, pages 556–559. ACM.

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

Malik, Z. and Bouguettaya, A. (2009). Reputation bootstrapping for trust establishment among web services. *Internet Computing*, 13(1):40–47.

Martin, R. C. (2002). *Agile Software Development, Principles, Patterns, and Practices*. Prentice-Hall, Inc.

Matsuo, Y. and Yamamoto, H. (2009). Community gravity: Measuring bidirectional effects by trust and rating on online social networks. In *International World Wide Web Conference (WWW)*, pages 751–760.

Medjahed, B. and Bouguettaya, A. (2005). A dynamic foundational architecture for semantic web services. *Distributed and Parallel Databases (DPD)*, 17(2):179–206.

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

Middleton, S. E., De Roure, D. C., and Shadbolt, N. R. (2001). Capturing knowledge of user preferences: ontologies in recommender systems. In *International Conference on Knowledge Capture (K-CAP)*, pages 100–107. ACM.

Middleton, S. E., Shadbolt, N. R., and De Roure, D. C. (2004). Ontological user profiling in recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):54–88.

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

Mui, L., Mohtashemi, M., and Halberstadt, A. (2002). A computational model of trust and reputation for e-businesses. In *Hawaii International Conference on System Sciences (HICSS)*, page 188.

Object Management Group (2008). Uml profile for modeling quality of service and fault tolerance characteristics and mechanisms specification, version 1.1.

O'Leary, D. E. (1998). Enterprise knowledge management. *IEEE Computer*, 31(3):54–61.

Osborne, M. J. and Rubinstein, A. (1999). *A Course in Game Theory*. MIT Press.

Papadopoulos, A. and Manolopoulos, Y. (1999). Structure-based similarity search with graph histograms. In *DEXA Workshop*, pages 174–178.

Petrie, C. (2010). Plenty of room outside the firm. *IEEE Internet Computing*, 14.

Psaier, H. (2012). *Behavior Management and Self-adaptation in Mixed Systems - Adaptation Models, Strategies, and Management*. PhD thesis, Vienna University of Technology.

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

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

Radlinski, F. and Joachims, T. (2005). Query chains: learning to rank from implicit feedback. In *Conference on Knowledge Discovery and Data Mining (KDD)*, pages 239–248. ACM.

Reka, A. and Barabási (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97.

Romesburg, H. C. (2004). *Cluster Analysis for Researchers*. Krieger Pub. Co.

Ronen, R. and Shmueli, O. (2009). Soql: A language for querying and creating data in social networks. In *International Conference on Data Engineering (ICDE)*, pages 1595–1602.

Sabater, J. and Sierra, C. (2002). Social regret, a reputation model based on social relations. *SIGecom Exchanges*, 3(1):44–56.

Salton, G. and Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523.

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

Schall, D., Dorn, C., Dustdar, S., and Dadduzio, I. (2008a). Viecar - enabling self-adaptive collaboration services. In *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 285–292.

Schall, D. and Dustdar, S. (2010). Dynamic context-sensitive pagerank for expertise mining. In *Social Informatics*, pages 160–175. Springer.

Schall, D., Dustdar, S., and Blake, M. B. (2010). Programming human and software-based web services. *Computer*, 43:82–85.

Schall, D. and Skopik, F. (2010). Mining and composition of emergent collectives in mixed service-oriented systems. In *Conference on Commerce and Enterprise Computing (CEC)*, pages 212–219. IEEE.

Schall, D., Skopik, F., and Dustdar, S. (2011). Bridging socially-enhanced virtual communities. In *Symposium on Applied Computing (SAC)*, pages 792–799. ACM.

Schall, D., Truong, H.-L., and Dustdar, S. (2008b). Unifying human and software services in web-scale collaborations. *IEEE Internet Computing*, 12(3):62–68.

Schroth, C. and Janner, T. (2007). Web 2.0 and soa: Converging concepts enabling the internet of services. *IT Professional*, 9(3):36–41.

Shahaf, D. and Horvitz, E. (2010). Generalized task markets for human and machine computation. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Skopik, F. (2010). *Dynamic Trust in Mixed Service-oriented Systems - Models, Algorithms, and Architectures*. PhD thesis, Vienna University of Technology.

Skopik, F., Schall, D., and Dustdar, S. (2009a). Start trusting strangers? bootstrapping and prediction of trust. In *International Conference on Web Information Systems Engineering (WISE)*, pages 275–289.

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

Skopik, F., Schall, D., and Dustdar, S. (2010b). Supporting network formation through mining under privacy constraints. In *International Symposium on Applications and the Internet (SAINT)*, pages 105–108. IEEE.

Skopik, F., Schall, D., and Dustdar, S. (2010c). Trust-based adaptation in complex service-oriented systems. In *International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE.

Skopik, F., Schall, D., and Dustdar, S. (2010d). Trusted interaction patterns in large-scale enterprise service networks. In *Euromicro Conference on on Parallel, Distributed and Network-Based Computing (PDP)*, pages 367–374.

Skopik, F., Schall, D., and Dustdar, S. (2010e). Trustworthy interaction balancing in mixed service-oriented systems. In *Symposium on Applied Computing (SAC)*, pages 801–808. ACM.

Skopik, F., Schall, D., and Dustdar, S. (2011a). Computational social network management in crowdsourcing environments. In *International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 273–282. IEEE.

Skopik, F., Schall, D., and Dustdar, S. (2011b). Managing social overlay networks in semantic open enterprise systems. In *International Conference on Web Intelligence, Mining and Semantics (WIMS)*, page 50. ACM.

Skopik, F., Schall, D., and Dustdar, S. (2011c). Opportunistic information flows through strategic social link establishment. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 308–315. IEEE.

Skopik, F., Schall, D., and Dustdar, S. (2012a). Discovering and managing social compositions in collaborative enterprise crowdsourcing systems. *International Journal of Cooperative Information Systems (IJCIS)*, 21(4):297–341.

Skopik, F., Schall, D., and Dustdar, S. (2012b). Trusted information sharing using soa-based social overlay networks. *International Journal of Computer Science and Applications (IJCSA)*, 9(1):116–151.

Skopik, F., Schall, D., Psaier, H., and Dustdar, S. (2010f). Social formation and interactions in evolving service-oriented communities. In *European Conference on Web Services (ECOWS)*, pages 27–34. IEEE.

Skopik, F., Schall, D., Psaier, H., and Dustdar, S. (2011d). Adaptive provisioning of human expertise in service-oriented systems. In *Symposium on Applied Computing (SAC)*, pages 1568–1575. ACM.

Skopik, F., Truong, H.-L., and Dustdar, S. (2009b). Trust and reputation mining in professional virtual communities. In *International Conference on Web Engineering (ICWE)*, pages 76–90. Springer.

Souza, C. d., Froehlich, J., and Dourish, P. (2005). Seeking the source: software source code as a social and technical artifact. In *International Conference on Supporting Group Work (GROUP)*, pages 197–206. ACM.

Souza, C. d., Quirk, S., Trainer, E., and Redmiles, D. F. (2007). Supporting collaborative software development through the visualization of socio-technical dependencies. In *International Conference on Supporting Group Work (GROUP)*, pages 147–156. ACM.

Stauffer, D. and Aharony, A. (1994). *Introduction to percolation theory*. CRC Press.

Stewart, O., Huerta, J. M., and Sader, M. (2009). Designing crowdsourcing community for the enterprise. In *KDD Workshop on Human Computation*, pages 50–53. ACM.

Stollberg, M. and Norton, B. (2007). A refined goal model for semantic web services. In *International Conference on Internet and Web Applications and Services (ICIW)*, pages 17–22.

Story, H., Harbulot, B., Jacobi, I., and Jones, M. (last access: 2011). Foaf+ssl: Restful authentication for the social web. http://esw.w3.org/Foaf+ssl.

Trainer, E., Quirk, S., Souza, C. d., and Redmiles, D. F. (2005). Bridging the gap between technical and social dependencies with ariadne. In *OOPSLA workshop on Eclipse technology eXchange*, pages 26–30. ACM.

Tran, H., Hitchens, M., Varadharajan, V., and Watters, P. A. (2005). A trust based access control framework for p2p file-sharing systems. In *Hawaii International Conference on System Sciences (HICSS)*.

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

Van Antwerp, M. and Madey, G. (2008). Advances in the sourceforge research data archive (srda). In *International Conference on Open Source Systems (OSS)*. Springer.

van der Aalst, W. M. P. and Song, M. (2004). Mining social networks: Uncovering interaction patterns in business processes. In *International Conference on Business Process Management (BPM)*, pages 244–260. Springer.

Vukovic, M. (2009). Crowdsourcing for enterprises. In *Congress on Services*, pages 686–692.

W3C (2008). Sparql query language for rdf. Online: http://www.w3.org/TR/rdf-sparql-query/.

Watts, D. (2003). *Six degrees: The science of a connected age*. W.W. Norton & Company.

Yan, X., Yu, P. S., and Han, J. (2005). Substructure similarity search in graph databases. In *SIGMOD Conference*, pages 766–777. ACM.

Yan, X., Zhu, F., Yu, P. S., and Han, J. (2006). Feature-based similarity search in graph structures. *ACM Transactions on Database Systems (TODS)*, 31(4):1418–1453.

Zaremba, M. and Vitvar, T. (2008). Wsmx: A solution for b2b mediation and discovery scenarios. In *ESWC*, pages 884–889.

Ziegler, C.-N. and Golbeck, J. (2007). Investigating interactions of trust and interest similarity. *Decision Support Systems*, 43(2):460–475.

APPENDIX A

CV

Please notice that the obligatory CV is provided in a separate document and added here in the printed version only.