FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Investigation of Techniques for Collaboration in Task-based Crowdsourcing

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Stefan Vallaster

Matrikelnummer 0525239

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Schahram Dustdar
Mitwirkung: Dr. Benjamin Satzger

Wien, 07.05.2012

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Investigation of Techniques for Collaboration in Task-based Crowdsourcing

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Stefan Vallaster
Registration Number 0525239

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Dr. Schahram Dustdar
Assistance: Dr. Benjamin Satzger

Vienna, 07.05.2012 (Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Stefan Vallaster
Weyringergasse 19/2/3+4, 1040 Wien

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

(Ort, Datum)	(Unterschrift Verfasser)

Danksagung

Das Einreichen dieser *Master's Thesis* markiert den Abschluss einer wunderschönen Zeit mit vielen Freiheiten und wenig Pflichten, dem sogenannten Studentenleben.

Ich möchte mich bei jenen Menschen bedanken, die mich in all den Jahren des Studiums begleitet, unterstützt und motiviert haben. Besonderer Dank gilt dabei meinen lieben Eltern, ohne deren Unterstützung ein Studium nicht möglich gewesen wäre. Darüber hinaus möchte ich mich bei meiner ganzen Familie und meinen Freunden für deren Rat und Unterstützung bedanken.

Des Weiteren gilt mein Dank meinen Betreuern Univ.Prof. Dr. Schahram Dustdar und Dr. Benjamin Satzger. Die Qualität der Besprechungen, das schnelle Feedback zu etwaigen Fragen, sowie die exzellente inhaltliche Unterstützung bei der Erstellung dieser Arbeit waren für mich äußerst hilfreich und sehr beeindruckend.

Abstract

Crowdsourcing, a distributed problem-solving model, is gaining more and more interest. Enterprises around the world show interest in using crowdsourcing systems to outsource work. Right now task-based crowdsourcing systems support only a very simple model work, a simple task. In general, a simple task is seen as an atomic unit of work, which is assigned to one single worker. However, the demands of enterprises to share more complex work are evident. The integration of complex work into task-based crowdsourcing leads to a number of challenges due the fact that complex work in general cannot be split into units of work, which can be assigned to a single worker.

In this thesis we introduce different *techniques of collaboration*, based on the integration of complex work to a task-based crowdsourcing system. We model complex work as a *composite task*. A composite task has a set of sub-tasks; the sub-tasks can have dependency between each other, which show how much cooperation is needed to solve the dependent task. Besides the introduction of complex work to crowdsourcing, we introduce a *social collaboration network*. All workers are part of this collaboration network, ties between workers in this social network represent the fact how well two workers can work together. Further, we introduce two team structures, namely *static and dynamic teams*. The models of a composite task, the social collaboration network and the two team-based approaches are implemented in a task-based crowdsourcing simulation framework. We further perform an evaluation, based on the implementation of our concepts, to show the advantages and limitations of both team-based approaches. The evaluation results show significant differences between the quantity of performed tasks and the quality of the processed work depending on the team structure.

Kurzfassung

Aktuelle *Crowdsourcing* Systeme unterstützen nur ein sehr einfaches Arbeitsmodell. Das Interesse von Unternehmen auf der ganzen Welt, die höchst komplexe Arbeit mittels Crowdsourcing auslagern möchten, steigt dabei kontinuierlich. Deshalb ist es notwendig Konzepte zu entwickeln um die Integration komplexer Arbeit in Crowdsourcing Systeme zu ermöglichen. Dabei birgt die Integration von komplexer Arbeit eine Reihe von Herausforderungen, da komplexe Aufgabenstellungen im Allgemeinen nicht in Arbeitspakete teilbar sind, welche von einzelnen Personen unabhängig voneinander abgearbeitet werden können.

Wir nutzen verschiedene *Formen der Kollaboration* um die Integration von komplexer Arbeit in Crowdsourcing Systeme zu ermöglichen. Wir modellieren komplexe Arbeit als sogenannte *Composite Tasks*. Ein Composite Task besteht aus einer Reihe von Teilaufgaben, welche Abhängigkeiten untereinander haben können. Diese Abhängigkeiten zeigen das Maß an Zusammenarbeit, welches nötig ist, um voneinander abhängige Teilaufgaben zu lösen. Ergänzend stellen wir ein Modell für ein *Soziales Kollaborationsnetzwerk* vor. Alle Arbeiter sind Teil dieses Netzwerks. Die Beziehungen zwischen Arbeitern im sozialen Netzwerk zeigen dabei, wie gut diese miteinander zusammenarbeiten können. Darauf aufbauend führen wir zwei Team-Strukturen, nämlich *statische und dynamische Teams*, ein. Unsere vorgestellten Konzepte werden in eine Simula- tionsumgebung für Crowdsourcing implementiert um eine Evaluierung zu ermöglichen. Wir führen eine Reihe von Auswertungen durch um Vor- und Nachteile der beiden Teamstrukturen aufzuzeigen. Die Ergebnisse der Evaluierung zeigen signifikante Unterschiede in der Anzahl der erfolgreich abgearbeiteten Aufgaben und der Qualität der erbrachten Arbeit in Abhängigkeit der Teamstruktur.

Contents

List of Algorithms

List of Figures

Listings

List of Tables

CHAPTER 1

Introduction

Crowdsourcing, a term coined by Jeff Howe in 2006 [19], is a problem-solving model based on outsourcing work in a distributed way via an open call to a network of people, the so called crowd. Since its beginning around the millennia crowdsourcing is gaining more and more popularity throughout companies around the world [41]. A more comprehensive and thorough definition provided by [10] is as follows:

'Crowdsourcing is a type of participative online activity in which an individual, an institution, a non-profit organization, or company proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a task. The undertaking of the task, of variable complexity and modularity, and in which the crowd should participate bringing their work, money, knowledge and/or experience, always entails mutual benefit. The user will receive the satisfaction of a given type of need, be it economic, social recognition, self-esteem, or the development of individual skills, while the crowdsourcer will obtain and utilize to their advantage that what the user has brought to the venture, whose form will depend on the type of activity undertaken.'

Getting a more practical understanding of crowdsourcing, we will try to give a simple example according to the definitions stated before. As can be seen, there are two types of actors within the model of crowdsourcing. Crowdsourcers, also known as requesters, and workers. A requester has a problem, e.g. a small task to sort pictures according to some criteria, which needs to be solved. The requester publishes this specific problem to an unknown crowd of workers. One or more of the workers will make the attempt to solve the problem and will send the problem's solution back to the requester. As crowdsourcing is used as a business model in most cases a small kind of revenue, e.g. money or prestige, is offered to the worker for successfully solving the stated problem.

After defining crowdsourcing and giving a simple example on how crowdsourcing works, the remainder of this chapter is divided into three parts. Beginning with a selective overview of the history of crowdsourcing, we continue with some examples for state-of-the-art crowdsourcing platforms. The next section will elaborate on the problems current crowdsourcing systems are facing. We state the approaches used right now and explain the motivation behind this master's thesis. The last section gives an organizational overview of the remaining chapters of this thesis.

1.1 History

To get a better understanding on the topic of crowdsourcing we will give a selective overview of the predecessors of modern crowdsourcing. These predecessors can be found as early as 1714, when the British government offered a price of £20.000 for solving 'The Longitude Problem', which was eventually solved by John Harrison, a worker-class carpenter [18].
As described in The Surgeon of Crowthorne [46], in the second half of the 19th century an open call was made for contributions to index the words of the English language. The input of this open call led to one of the first versions of the well known Oxford English Directory.
In the mid of the 20th century an international architectural competition for the Sydney Opera House was started [37], which lead to 233 suggestions from around the world.

The examples, given before, show that the idea behind crowdsourcing is in fact a very old one. However only the success of the web in its current technological stage made it possible to get quick and easy access to a broad audience of workers. Hence the web provides the foundations to make state-of-the-art crowdsourcing systems possible. This is the reason why after the millennia modern crowdsourcing systems, described to some extend in [3] and [41], have been populated throughout the internet. Three of the most well known crowdsourcing platforms available today are explained below.

- Wikipedia, which is [11] according to Alexa [1] within the top 10 webpages around the world, was founded in 2001. Wikipedia, a plattform for crowdwisdom, is an online encyclopaedia. Each article in Wikipedia is created, checked and rated by the users visiting the webpage.

- One very popular and successful crowdsourcing system founded in the first half of 2000, is iStockphoto.com [24]. iStockphoto.com is a platform where everyone can buy royalty-free photographs. The pictures get uploaded and categorized by registered photographers, which represent the crowd. To become a photographer you have to go through a registration process, described in [3]. Clients can buy the images. For each image bought, the photograph receives a small amount of money, the bigger share of the selling price belongs to iStockphoto.com.

- Another well known crowdsourcing system, launched in 2005, is Amazon's Mechanical Turk (AMT) [20]. AMT, a micro-task aware crowdsourcing system, is offering a platform where everyone can publish human intelligence tasks, so called HITs. Some of the HITs require a certain qualification level, which has to be specified by the worker, during task creation. AMT's workers solve HITs, which have been created beforehand, according to their qualification level, and gain income by doing so.

1.2 Motivation

Like in AMT's case, state-of-the-art task-based crowdsourcing systems only allow the creation and population of micro-tasks. Micro-tasks are *atomic tasks*, which cannot be

split up. They are meant to be solved by one single worker. As can easily be seen, this model for work does not meet all the demands introduced by the increasing interests of enterprises around the world to solve more complex work with crowdsourcing systems. Thereby the need to integrate forms of complex work into task-based crowdsourcing systems can clearly be seen.
Another problem arises with the introduction of complex work. As complex work cannot be solved by a single individual, the means of splitting the work into manageable tasks have to be defined. Three approaches can be made to address this issue.

- The requester himself has to do a conclusive partitioning and has to split the complex work into tasks. This approach seems to be infeasible, as it cannot be applied to every kind of complex work in general, moreover it is even not sure if the requester has the knowledge to split the task.
- A second approach would be to let the complex work be analysed by the crowd itself, this approach was proposed in [25]. Still the same problem as before remains, it might not be possible to find an adequate set of subtasks, which each can be performed by a single individual.
- When partitioning complex work is not possible the crowdsourcing platform could support mechanisms for collaborative structures, so that the complex work can be solved by a group of individuals.

The third approach leads to another limitation of crowdsourcing systems. There is no support for workers communicating with each other. There is neither support for techniques for collaboration, nor is the social network, formed by the crowd itself, used to solve problems with more focus on the distributed knowledge represented by the crowd. This means that workers have no chance to solve problems together, which can clearly be seen as a reason why only micro-tasks are supported right now.

The limitations of currently implemented crowdsourcing systems described before, make crowdsourcing an interesting, challenging and important research area in the field of computer science and are the basis for the motivation behind this master's thesis.

1.3 Contribution

In this thesis the limitations discussed before will be addressed. To overcome the issue of not supporting the social network, the crowd is explicitly defined as a social collaboration network. The system is aware of the social network and can take advantage of this fact. We propose a solution for integrating a model for complex work into a task-aware crowdsourcing system. The focus is on the investigation to find suitable techniques-of-collaboration in the context of task-based crowdsourcing systems. The contribution itself is split in the following parts:

- Besides supporting micro-tasks, we introduce the model of a *composite task* to support complex work. A composite task itself has a finite set of subtasks, which are allowed to have dependencies among each other, therefore the composite task is represented with the help of an undirected labelled graph.
- Two different techniques of collaboration, namely static and dynamic teams will be defined. Both approaches towards introducing teams to task-based crowdsourcing will be based on the crowd, which is structured as a social collaboration network as mentioned before.
- An algorithm for partitioning composite tasks within a team, based on the qualification of the workers within the team, is presented.
- Both structures for collaboration, namely static and dynamic teams, are evaluated and the results are compared to see advantages and limitations of the chosen approach. For the evaluation the proposed changes to task-based crowdsourcing systems will be implemented into a framework for simulating crowdsourcing systems.

1.4 Organization

The remainder of the thesis is organized as follows:

- Chapter 2 gives an overview of state-of-the-art task-based crowdsourcing systems, social networks and collaboration networks, besides that the focus will be on related work in the area of supporting complex tasks in crowdsourcing systems, collaborative- and social-computing and the team formation problem.

- Chapter 3 starts by introducing a generic model for task-based crowdsourcing. Based on this model, we introduce the concepts for our social collaboration network. On top of the social network we will describe in detail static and dynamic teams. Right after defining teams, our concept for supporting complex-work, namely the composite task, is explained in detail.
- Implementing the concepts and models explained in chapter 3 into a framework for simulating crowdsourcing are the focus of chapter 4.
- Chapter 5 will explain the configuration used to perform an evaluation of the concepts, which were implemented in chapter 4, and will discuss the results of the data, gathered by the evaluation.
- Chapter 6, a summary with suggestions on future work, will complete this master's thesis.

CHAPTER 2

Literature Overview

This chapter is twofold. The first part gives an overview of the state-of-the-art concerning crowdsourcing systems, social networks and collaboration networks. The second part focuses on related work in the area of social computing, the team formation problem. Furthermore extensions to micro-task aware crowdsourcing are presented and two frameworks for simulating crowdsourcing are introduced.

2.1 State-of-the-art

Crowdsourcing is a very versatile area, different platforms have been set up throughout the last decade. The platforms diversity reaches from games, over crowd-wisdom platforms, to micro-task aware marketplaces as partly described in [3] and categorized into four distinctive groups in [44]. All crowdsourcing platforms support the interaction between the platform and users and dynamic web-based applications, which can be accessed by the users via a web browser. We split our overview of state-of-the-art crowdsourcing systems into two sections. First we give an overview of the most popular market-based crowdsourcing systems, then we will give an overview of crowdsourcing systems without the need of using a marketplace.

2.1.1 Market-based Crowdsourcing

Market-based crowdsourcing systems use a marketplace as the interaction point between requesters and workers. Usually these systems offer a salary as incentives for workers to solve tasks. AMT, a crowdsourcing system launched in 2005, offers workers the possibility to solve a huge number and variety of 'Human Intelligence Tasks' (HIT). Workers receive a small amount of money for the work they have performed, according to [44] the worker's salary is in most cases less than 5 cents USD. HITs can be submitted by requesters. Requesters can define and create certain qualifications, a worker needs, to be able to solve a HIT. To obtain a qualification, in most cases a worker has just to select it. However there are also certain qualifications for which a worker has to perform a test before s/he can obtain the qualification. A worker is able to access and select HITs he wants to solve by simply using a web browser. On the requester side AMT uses *Web Services*. Therefore AMT uses a *Service-oriented Architecture (SOA)* [17]. In AMT's case a HIT can be solved exclusively by one worker. Therefore workers are not in a direct competition with each other. Taskcn [38], described in [47] as one of the biggest task-based crowdsourcing systems in China, uses a more competitive approach. There, workers compete with each other directly, by simultaneously submitting solutions for the same tasks. As analyzed in [47], this leads to an environment, which attracts people with a certain amount of expertise. Due to the high rate of competition requesters try to make tasks as simple as possible, so that a broader range of solutions is submitted by workers. Another highly competitive crowdsourcing platform, which offers remarkable high salaries in comparison to platforms like AMT or Taskcn is Innocentive [21]. Innocentive is a crowdsourcing system which provides the capabilities for companies to outsource research and development. Innocentive offers a wide range of different types of challenges a company can create. The challenges are split into three categories, namely Brainstorm Challenge, Premium Challenge and Grand Challenge. Depending on the challenge's category the salary offered to workers ranges from 500 USD to more than 100.000 USD per challenge.

2.1.2 Non Market-based Crowdsourcing

Besides the very popular market-based crowdsourcing systems, presented in the last section, various other systems exist, where workers and requesters are not communicating with each other via a market. Wikipedia [11], a plattform for crowdwisdom, is an online encyclopaedia. Each article in Wikipedia is created, edited and rated by the users visiting the web page. The crowd of users accessing the Wikipedia share a dual role, while most visitors are seeking for information of a specific topic, others take there time to help and correct existing articles or to write brand new ones. To guarantee a certain amount of quality Wikipedia introduces ratings for articles. Besides that, Wikipedia supports policies and techniques to counter vandalism. This is accomplished by promoting trusted users to editors. Those editors are there to guarantee the quality of certain articles, by checking the changes submitted by normal users. Other non market-based crowdsourcing systems including Yahoo Answers [23] or Stackoverflow [22] focus on providing a web page where users can ask questions, which are then answered by other users. Unlike market-based crowdsourcing systems non-monetary incentives are given to users answering questions. These incentives typically include badges and a system of reward points. Whereas Yahoo Answers focuses on providing a platform to ask and answer generic questions, Stackoverflow calls itself a site for answering and asking questions around the topic of programming.

After giving an overview of market-based and non market-based crowdsourcing systems the next sections focus on social networks and collaboration networks.

2.1.3 Social Networks

Social networks are formed by group of people. Each individual represents a node. As people are acquainted with each other, there exists a number of ties between the nodes of the social network. Sociologists around the world have studied social networks in detail. The work presented in [31] analyses the average path length between two individuals in a social network. It shows that the path length in comparison to the size of a social network is very low (e.g. is the path length between two American citizens six hops). The author of [15] separate the links between individuals of social networks into strong and weak ties and argues that strong ties are tightly clustered. Lately online social networks are gaining more and more popularity. Research of these kind of social networks has shown that online social networks share the same properties as their offline pendants [32]. Milgram [31] determines that social networks are structured with a 'small world' property. Research in [14] shows that social networks contain community structures. Altogether over the last decades research in the area of networks has determined certain properties. These properties are shared among social networks and include the following:

1. **Power-law degree distribution**: As explained in [32] social networks with a power-law distribution have nodes, where the nodes probability to have a degree of k is proportional to $k^{-\lambda}$. This holds for a large k and a $\lambda > 1$. λ is called the *power-law coefficient*.

2. **Scale-free property**: The scale-free property is a property of power-law distributed networks. It describes the characteristic that nodes with a high degree are connected to other nodes with a high degree. A detailed explanation of the scale-free property can be found in [29].

3. **Small-world property**: The small-world property was first described in [31]. A network, having a small-world property, can be characterized as having an small average path length in comparison to networks size. That means that two individuals, being part of a social network, are connected with each other through a small number of intermediaries.

4. **Network-transitivity/ Clustering**: Network-transitivity describes the property, that if two nodes A and B have both a connection to node C then the probability of node A and B being connected to each other is heightened.

2.1.4 Collaboration Networks

Collaboration networks are a specialized kind of social networks. In collaboration networks the ties between the individuals are used to quantify the collaboration. Different

models to represent collaboration networks have been discussed in [33], [29]. Each of the models adds a value, indicating the collaboration, as a label (or weight) to a tie between two nodes. The results presented in [33] confirm that collaboration networks share the same properties as social networks in general do.
We explain the papers stated before as part of the next section, which is focusing on related work in the area of social computing, team formation and extensions to micro-task aware crowdsourcing systems.

2.2 Related Work

This section focuses on work in the research area of task-based crowdsourcing, human-based and social computing.

2.2.1 Social Computing

A research area in particular related to social networks and collaboration between individuals is social computing, which is described in [42]. The authors of [8] propose a novel approach, *Social Compute Unit (SCU)*, to integrate social computing into modern workflow systems [5] and systems for business process management [39]. A SCU is a virtual construct. Computing within an SCU is performed by humans, brought together through a social network. According to the authors a SCU utilizes aspects for programmability, compute power and elasticity.

A SCU itself is created at request time and is disbanded after computing the assigned task. The life-cycle of SCU includes the following stages:

- Request: A SCU is requested for an specific problem domain.
- Create: A SCU is created.
- Assimilate: The SCU is introduced to the problem and the problem's domain.
- Virtualize: The SCU is installed. This includes providing a collaboration space for the members of a SCU to communicate with each other and also providing a test environment for the SCU.

- Deploy: The SCU is producing results.
- Dissolve: The SCU is dissolved and rewarded for the work, which it has performed.

An AppStore like architecture is proposed to implement the SCU. Humans can register at the system. Upon request for a SCU by a client, a *SCU Compiler* is used to create a SCU for the problem domain, which was specified by the client. To accomplish this an *assessment unit* is used for finding the right persons to form a SCU.
The work in [9] describes the *Social Routing Principle*. A principle to help to overcome the problem of integrating social networks and mobile applications like smart phone apps or web applications [45] to solve tasks based on social computing. The authors propose the mechanisms for task delegations in mobile applications, to achieve this they introduce a *Social Router*, which aim is to delegate work, by using a model for solving problems vertically across different collaborative systems, e.g. task-based crowd-sourcing systems like Amazon's Mechanical Turk. The router supports three kinds of collaborative environments, namely personal communities, context-based communities and crowd-sourcing systems. For delegating work to this systems the router has to have the knowledge on how to communicate with the systems. The authors point out that the introduction of a Social Router introduces opportunities and challenges. This challenges include the integration of trust and privacy features when delegating tasks. Besides that people have to be supported by applications to formulate a problem or task in a way that it can be translated by a Social Router.

2.2.2 Team Formation

Besides social computing, the NP-hard team formation problem, described in [7] and [28], is a very important area of research. It is the foundation for automatically forming teams based on a social network, which is implicitly introduced by social computing. The work presented in [7] describes heuristics to form teams effectively based on the structure of an underlying expert social network. Recommendations of collaboration and the number of direct interactions between experts are used to find teams effectively. Team formations are used for a task, requiring a set of skills, to be solved. The authors

propose two heuristics for performing team selection. The first heuristic is based on a genetic algorithm, which treats the problem of team formation as the problem of finding the best individual (i.e. a team), by determining the individual with the highest fitness, defined by their chromosomes (i.e. the team configuration). A chromosome is defined by a set of genes, where a gene represents a skill of an expert. The algorithm uses two steps. A crossover step to mate individuals and a mutation step, which switches the value of a gene (i.e. an expert is switched with another expert). The second heuristic is based on an algorithm for simulated annealing, there the problem of team configuration is simulated as a hot system which is being cooled down with a number of transitions until it reaches a cold state. The algorithm uses the same technique as described by the mutation step of the genetic algorithm. Each changed team configuration is compared with the current configuration, if the new configuration is worse than the current, the system cools down.
Work, presented in [28], focuses on solving the team formation problem, by minimizing the communication cost within a team. The work is based on modelling a social network, as an undirected weighted graph, where edges between people represent the communication cost. Several algorithms are proposed. The algorithms are based on the diameter communication cost within a group of individuals, which is defined by the largest shortest path within two individuals of the group, and by the costs of a minimum spanning tree, which is computed for the group of people. The costs represent the sum of weights of the edges included in the diameter or the minimum spanning tree.

2.2.3 Extensions to Micro-task Aware Crowdsourcing

Besides related work in the research areas of social computing and solutions to the team formation problem, we also present work in the area of task-based crowdsourcing, specifically we focus on the area of micro-task aware markets like AMT.
TurKit [30] is a toolkit, which makes it possible to program iterative task for AMT. The toolkit supports writing tasks as imperative programs and gives examples for iterative text improvement and a simple sorting algorithm using AMT workers.
CrowdForge, a general purpose framework for solving complex tasks within standard micro-task aware crowdsourcing systems, is proposed in [25]. The aim of the work

is to solve complex tasks within normal micro-task aware crowdsourcing systems like Amazon's Mechanical Turk. To accomplish this, the authors propose a system, which supports building a workflow for a general complex task. Building a workflow includes the following three types of sub-tasks:

- Partition task: A partitioning task, is sent to the crowdsourcing system. The purpose is to get a partition of a bigger task.
- Map task: A map task is processed by one or more workers.
- Reduce task: The aim of the reduce task is to reduce the number of results for a specific sub-task to one result.

All three types of sub-tasks can be assigned as micro-tasks to a crowdsourcing platform. The aim is to use a divide-and-conquer strategy to process the complex task. A typical approach is to send a partition task to a crowdsourcing system and to use the partitioning result to generate map tasks for each partition. As the last step the results produced by the map tasks for each partition get reduced to a final result, by using the reduce task. As pointed out by the authors, it might not be always possible, to split complex work, in a partitioning of sub-tasks which can all be solved by individuals.
Turkomatic [27], yet another framework to make it possible to process complex work with the help of predefined workflows, introduces a similar approach as before. A divide-and-conquer approach is used to process complex tasks. The authors propose a recursive algorithm and implement a visual workflow editor to support requesters in designing a workflow.
The work presented in [35] focuses on another problem which arises with the support for complex work. The authors argue that the introduction of complex work to crowdsourcing results in a significant increase of tasks, which need to be processed by the platform. Therefore the focus is not only on the quality of the results processed by workers but also on the throughput of the crowdsourcing system itself. A voting mechanism, for ensuring high quality of results and maintaining a higher throughput in comparison with typical strategies of using redundancy to keep the error rate low, is proposed. Based on the voting algorithm another algorithm is used to boost the throughput further. The

boosting algorithm is based on the assumption, that workers who produced few errors in a certain amount of time are more reliable than other workers.

2.2.4 Simulation Frameworks for task-based Crowdsourcing

As crowdsourcing itself is a novel research area not a lot of frameworks exist to simulate crowdsourcing. The work presented in [34] introduces a task-based crowdsourcing system, which supports skill evolution. Workers and requesters communicate via a marketplace with each other; direct communication is not supported. The framework uses an auction-based task assigning process, based on the usage of sealed-bid auctions. To support skill-evolution workers have a confidence value, determining the trust in the quality of solutions, which the worker delivers. Determining the confidence of workers is performed by using so called 'assessment tasks'. If a newly registered worker, which has not been assessed yet, wins a task in an auction then the system assigns the same task to another worker, who's confidence has been determined before. The results of both workers are then compared to determine the confidence of the newly assessed worker.
The simulation framework for task-based crowdsourcing proposed in [16] uses also auctions to assign tasks to workers. The framework is built on top of the agent-based modelling framework JABM [12]. The framework is highly configurable and supports the simulation of different scenarios. Besides supporting different types of auctions (e.g. Dutch auction, English auction, second price sealed-bid auction and continuous double auction) the framework also supports the configuration of different requester and worker behaviours. This is accomplished by providing several trading strategies and valuation policies, which are used by workers and requesters when participating in an auction.

CHAPTER 3

Methodology

The aim of this thesis is to overcome some limitations state-of-the-art crowdsourcing systems are facing. We want to accomplish this by introducing techniques of collaboration to task-based crowdsourcing systems. This chapter is split into three sections. The first section describes a generic model for task-based crowdsourcing. We use this model to explain the aforementioned limitations. Then the second section introduces extensions to overcome the limitations discussed before. The extensions include the introduction of a collaboration-based social network and the concept of composite task, our approach to support complex work in the context of task-based crowdsourcing systems. Based on these extensions we will introduce techniques for collaboration, manifested as static and dynamic teams, which are two different types of collaborative structures.

3.1 The Task-based Crowdsourcing Model

In chapter 2 we give an overview of state-of-the-art task-based crowdsourcing (TC) systems. Based on our observations we introduce a generic model for task-based crowdsourcing. Figure 3.1 illustrates the generic TC model. The TC model has two actors, namely *requesters*, *workers*, which interact with each other with the help of the *marketplace*. To understand the abilities and limitations of current TC systems, in the next

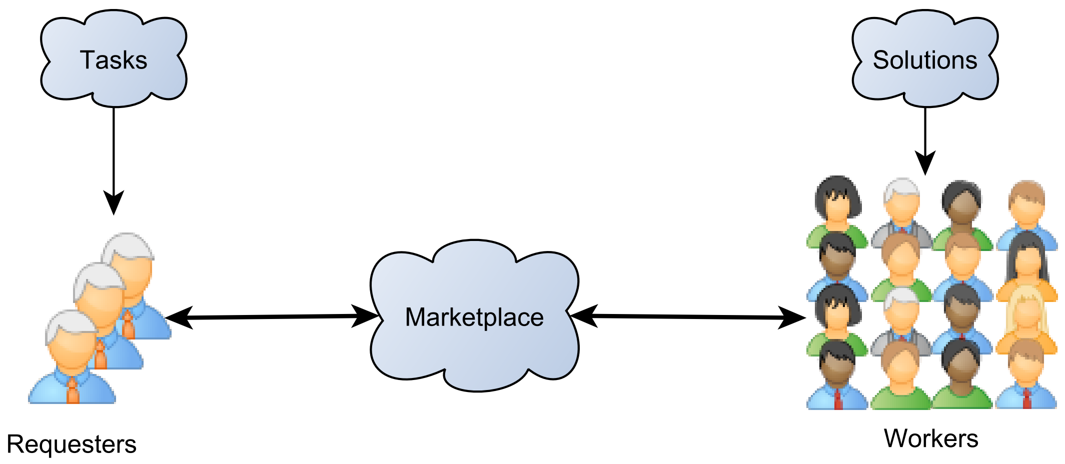

Figure 3.1: The Task-based Crowdsourcing Model

subsection we describe the marketplace. Then we introduce the notion of a task. After explaining the marketplace and the task, the focus is on requesters and workers.

3.1.1 Marketplace

The marketplace acts as the link between requesters and workers. Before any interaction between workers and requesters can take place, both group of actors have to register themselves at the marketplace. The marketplace holds a list of all tasks, created by requesters, which can be solved by an individual worker. The marketplace can be configured in a way to show a particular worker only a specific selection of all available tasks. This selection of tasks depends on the worker's skill set and rating. In TC systems a task listed in the marketplace can only be selected by a single worker.
A requester creates and submits tasks to the marketplace. Upon completion of a task, the requester transfers the offered reward to a worker. Besides that, the requester rates the worker according to the solution, which was submitted by the worker.
Upon registration a worker is able to solve tasks. If the worker submits a valid solution for the selected task within the deadline, specified by the requester, the worker will receive the offered award and a rating based on the worker's solution. The ratings are used to maintain a certain quality level.

3.1.2 Tasks

The TC model is based on the notion of tasks. A task represents an arbitrary set of problems, for which a requester requires a solution. However a task is assigned only to one worker, that means that the task has to be solvable by an individual. Therefore tasks are usually some sort of simple work (e.g. Categorize a certain amount of pictures with predefined categories). Tasks are created by requesters, tasks are distributed to a worker via the marketplace, and the workers solve tasks. A task has certain properties, which have to be specified, according to the TC model we introduce. The properties include the following:

- **a description**
- **the offered salary**
- **a deadline**
- **the set of required skills**

The description is essential in solving a task. It has to give access to all information, which are necessary to start solving the tasks. Furthermore, it has to describe as exactly as possible what challenge the task represents and how a solution for the challenge has to be provided. The salary defines the maximum price a requester is willing to pay for a valid solution. The deadline marks the maximum timespan for a worker to solve the task and to submit a solution. In addition, a requester has to specify all mandatory skills needed for a worker to solve the task. By selecting a certain skill, a requester has also to specify the quality level and the weight of the particular skill. The weight shows how important a skill is to solve a task. The sum of the weights for all specified skills is determined by a weighting function. It is defined over all required skills s and denotes the fact, that the sum of all specified skill weights w cannot exceed 1.0.

$$\sum_{i=0}^{s} w_i = 1.0$$

The skill quality is defined by a number within the interval of 0.0 and 1.0, where a value close to 1.0 means that the quality has to be very high.

3.1.3 Requesters

A requester is a person who has a task to solve. The task includes one or more problems. The requester wants to use the crowd of workers, which the marketplace offers, to find a solution for the problems. Each requester has to register at the marketplace to be able to participate in a TC system. Requesters create tasks. Doing that, requesters have to specify what work needs to be done and until when the task has to be finished. Besides that requesters have to specify a reward for solving the task. Hence in TC systems requesters specify the price they are willing to pay if a worker submits a solution within time. Moreover as the TC system supports workers with multiple skills, requesters have to define which skills are needed to solve a task and how high the skill-based quality level has to be for a worker to successfully perform the task. Then the skills have to be weighted. If all the necessary information has been specified, a requester can send the task to the crowdsourcing platform. A suitable worker can select the task. After receiving a solution provided by a worker, the requester has the possibility to rate the worker's solution. This rating, based on the workers skills, is necessary due to the fact that the quality of the solutions submitted by workers is never constant.

3.1.4 Workers

A worker is a person, who wants to spend time to solve tasks to gain rewards by doing so. The tasks are accessible through the marketplace. Like requesters, workers have to register themselves at the marketplace. Upon successful registration, each worker has to specify the skills he has. An initial rating of skills can be performed with training tasks as described in [34] or with an optional test, as used by AMT.
The workers aim is to solve as many tasks as possible, while delivering high-quality solutions so that the skill-based rating increases. After the qualification of a worker has been defined, a worker can select a task from the list of available tasks, which he

wants to solve. If the worker manages to provide a solution to the problem stated in the task within the deadline the requester has specified, he submits the solution via the marketplace to the requester. In exchange he receives the salary offered by the requester. Besides that the worker will be rated by the requester according to the quality of the worker's solution.

The generic model for task-based crowdsourcing, which we propose, works well for simple work. However task-based crowdsourcing platforms, like AMT, are getting more and more popular. As stated in [41] the rising demands of enterprises around the world to use crowdsourcing show the limitations of the generic task-based crowdsourcing model.

3.1.5 Limitations

The model presented above has a limitation, regarding the complexity of work which is supported. Right now TC systems only support simple work, which is solved by an individual. To overcome this limitation a model for task-based crowdsourcing has to be defined which introduces complex work. So far extensions to micro-task aware crowdsourcing, which are presented in chapter 2, support only a limited range of complex work. Both approaches, described in [27] and [25], focus on work which is partly iterative and which can be modelled as a simple workflow. The workflow uses a divide-and-conquer approach, that means that the complex work is split into sub-tasks. The results of the sub-tasks are then merged together. Splitting and merging can both be performed several times.
However in general not every kind of complex work can be split into parts, which are solved by individuals. That means that a form of collaboration is required between the workers to solve that kind of complex work. If we look at the model for task-based crowdsourcing, which we have introduced before, then we can see that a form of collaboration is not supported. The workers cannot communicate with each other directly, furthermore there is no way for workers to work together within the context of a task-based crowdsourcing system.
Therefore, to overcome the lack of techniques for collaboration, we propose in the next

section extensions to the model for task-based crowdsourcing to support collaboration. As a foundation, to make collaboration possible, we introduce a collaboration-based social network. Furthermore we define a model for complex work, the composite task. Then two approaches to support collaboration within task-based crowdsourcing are presented.

3.2 Extending the Task-based Crowdsourcing Model

This section explains in detail our approach to introduce worker collaboration within the context of a task-based crowdsourcing system. The demand for doing more complex work with the help of crowdsourcing systems is rising, currently state-of-the-art crowdsourcing systems do not offer support for processing complex work. Our approach towards overcoming this limitation is the introduction of worker collaboration. Currently the worker crowd is unstructured, that means that in TC systems workers have no possibility to contact each other directly and there is no support for workers to collaborate. Therefore, in the next subsection we introduce a model for defining the crowd explicitly as a social collaboration network. Besides that we define how complex work is represented in our extended model for task-based crowdsourcing.

3.2.1 The Social Collaboration Network Model

We want to introduce collaboration to task-based crowdsourcing systems, therefore the purpose of this section is to introduce a structure, which makes collaboration between workers quantifiable. The foundation on which our model is based is the crowd of workers and the ability of two workers to work with each other. That means that workers span a social structure. Furthermore to explicitly model the collaboration between workers, we have to introduce a relationship between a pair of workers in the social structure. This leads us directly to the well known model of social networks.
Social networks are a social structure spanned by individuals, acting as nodes, and the relationships between each of these individuals. As can easily be seen, social networks fit all the requirements to model the collaboration between workers.

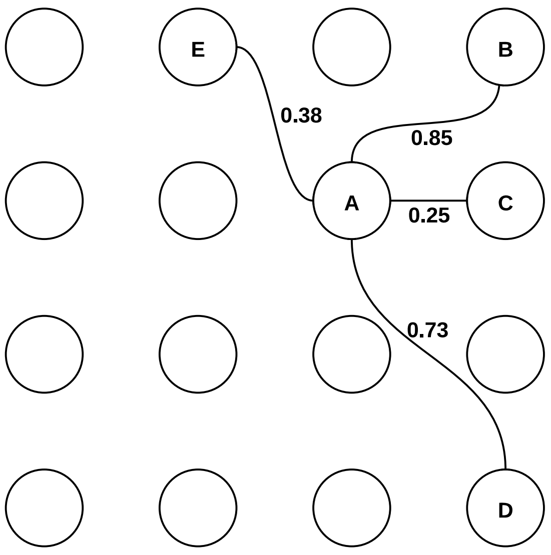

Figure 3.2: The Social Collaboration Network Model

In computer science a network is modelled as a graph. A graph consists of a set of nodes, and a set of edges between the nodes. For our model of a social collaboration network we use an undirected labelled graph. Every node in the graph represents a worker, and each edge between two workers A and B denotes the fact that the workers can collaborate with each other. Figure 3.2 shows a graphical representation of the social collaboration network. We see a number of nodes, representing workers. Besides that we see connections between the nodes, these connections represent the ability of a pair of workers to work with each other. Furthermore we add a label l to each edge to quantify the collaboration between workers. l, the *collaboration factor*, is a numeric value between 0.0 and 1.0, which defines how good worker A and B can work together. As we see later, the collaboration factor is essential for solving a composite task, as it describes how good people work together. Therefore a bad collaboration factor close to 0.0, will have indirectly a negative effect on solving a composite task. The explanation for this is simple. As the work is shared between a group of people, the ability of the people to collaborate with each other becomes important. Collaboration requires a lot of communication, scheduling, adjusting to each other, hence a problem in doing so will have an impact on the result of the work performed by the group of people, and reflects in a low collaboration factor. This also means, that the collaboration factor between a pair of workers can change. If workers work together more often, then their collaboration factor will increase. We argument here that collaboration is based on training and therefore increased training will sustain in higher ability to work together with each other.

3.2.2 Composite Tasks

We have so far introduced the model used for representing our collaboration network. In the next section we introduce techniques of collaboration based on our network model, therefore we have to rethink the structure and the variety of tasks supported by crowdsourcing systems. So far TC systems only offer tasks, which cannot be split and which are solved by a single worker. Supporting only this form of tasks ignores the high demand of enterprises to solve complex work within task-based crowdsourcing systems. Hence we introduce a new form of task, called the composite task. A composite task has a finite set of sub-tasks. Each sub-task can either be a composite task itself or a task. We introduce dependencies between sub-tasks, this is due to the fact that generally complex work cannot be divided into independent tasks. If a dependency between a pair of sub-tasks exists, that implies that the workers assigned to these sub-tasks have to work together to solve both tasks.
Like the social collaboration network, we model the set of sub-tasks and the dependencies between the sub-tasks as undirected labelled graph. Each node of the graph represents a sub-task. As mentioned before sub-tasks can be composite or normal tasks, that implies that one could theoretically make a complex task with an infinite depth. An edge between a pair of sub-tasks shows that these two subtasks depend on each other. The edge is labelled with the *cooperation factor*, a value between 0.0 and 1.0. A cooperation factor between sub-tasks A and B close to the maximum 1.0 denotes the fact that A and B have a high dependability on each other, which implies that a high amount of cooperation, denoted by the collaboration factor between a pair of workers, is required to solve both tasks.

Figure 3.3 shows an example of a composite task. The composite task has 6 sub-tasks A, B, C, D, E and F. As we can see sub-tasks A, B and C depend on each other.

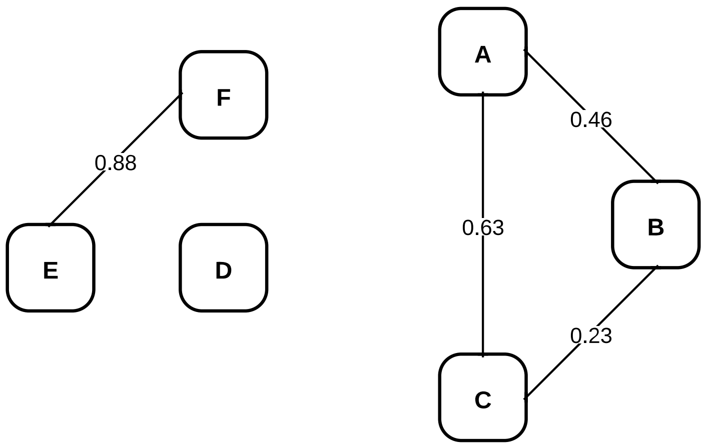

Figure 3.3: An Example of a Composite Task

In detail B and C have a low cooperation factor of 0.23. The cooperation factor between A and C is 0.63, that is moderately higher then the factor between A and B, which is 0.43. Another group of dependent sub-tasks is formed by E and F. E and F have a very high dependency of 0.88 between each other, that means that workers assigned to those two tasks will have to work together very closely to manage to find a solution for both sub-tasks with a high quality. In addition to the two groups of sub-tasks the composite task has also one sub-task D which has no dependency to another sub-task.

Figure 3.4 gives an overview of the extended task-based crowdsourcing model we propose. In this section we have introduced a social collaboration network, which explicitly shows the collaboration between workers. Furthermore we have defined a generic structure to model complex work, the composite task. Therefore in our extended model for task-based crowdsourcing, the requesters, the marketplace and the crowd of workers have to support composite tasks. Besides that, to make the introduction of the social collaboration network meaningful, the next section defines the techniques of collaboration, which are used to take advantage of the social collaboration network.

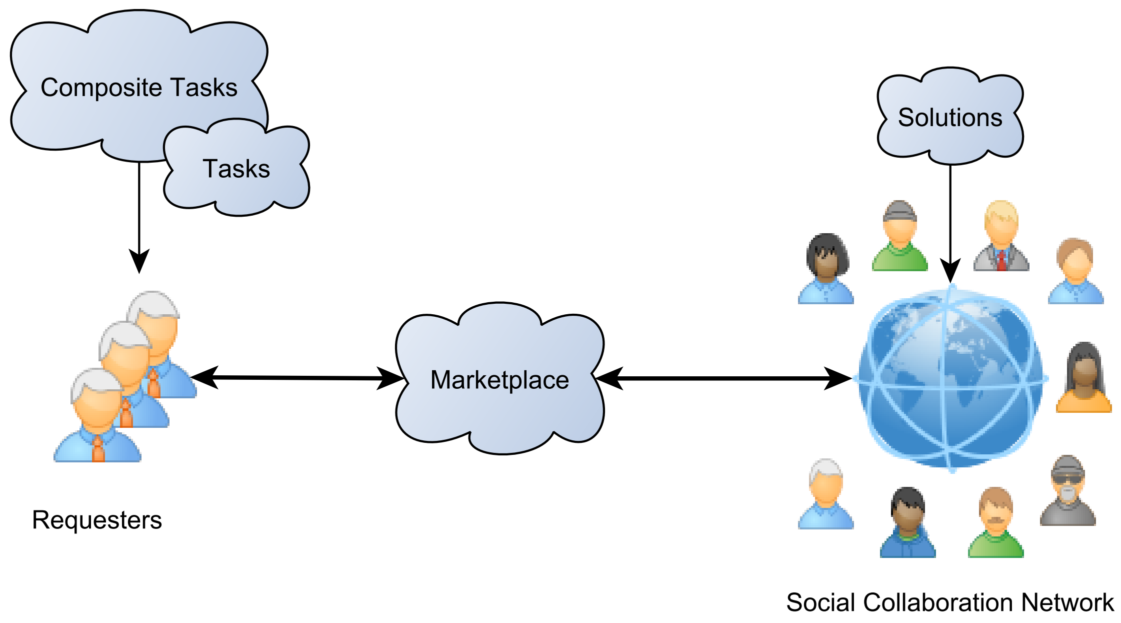

Figure 3.4: The Extended Task-based Crowdsourcing Model

3.3 Introducing Techniques for Collaboration to Task-based Crowdsourcing

Built upon the extended model of task-based crowdsourcing, which is shown in Figure 3.4 we introduce techniques for collaboration. These techniques use the social collaboration network, which the workers span. As we have argued before, the introduction of composite tasks is necessary for workers to collaborate with each other. Therefore we introduce two team structures for workers. Both team structures are based on the social collaboration network, which has been introduced in the section before. Once again this means, that we have to extend our crowdsourcing model in a way that teams can be supported. This means that the marketplace, the central component of a crowdsourcing system has to be extended, to support the registration of and the interaction with teams. For a requester there are no changes necessary, this is due to the fact, that a team just represents a special form of worker.

3.3.1 Static Team

A *static team* (ST) describes our first approach towards introducing collaborative work into crowdsourcing systems. STs are based on the structure of the underlying social collaboration network. Workers who know each other and workers, which work well together, can form a ST. By doing so the workers have to register themselves at the marketplace as a static team. It is implied that all members of the team are themselves registered at the marketplace as well. The ST is represented as a virtual worker and becomes part of the crowd. That means upon successful registration of the ST, members will have a dual role within the task-based crowdsourcing system, they will be normal workers as before and they will also be part of a team, which is also represented as a worker. Therefore they can solve tasks within the bounds of their ST or as a single worker alone. The fact that the team has to be operated by at least one team member to successfully interact with the marketplace is not of our concern.

Static teams offer the members some advantages.

- Skills are aggregated by the virtual worker representing the static team, this means that all different skills of all members are thrown together. This increases the possibility to conduct a wider range of tasks.
- Our model of static teams follows an optimistic approach, where the rating and level for each skill of the virtual team is determined by the maximum of the team member's individual ratings and levels.
- With the approach chosen before to use the maximum rating, workers within a team have a higher possibility to get access to tasks for which they would have been rejected as an individual worker due to their skill level and rating.
- The dual role of being a normal worker and being member of a ST increases the possibility to receive more work over time.

Figure 3.5 gives and example for the teams skill-aggregation. The team consists of four workers. All four workers have together four different skills $S1, S2, S3$ and $S4$. The number close to the skill determines the worker's skill level. A higher value means that a worker is more skilled than a colleague with a lower value. Two workers share the skill $S1$; one worker has a level of 0.8, the other worker has a level of 0.3. Therefore, by using our optimistic approach, this will result in the team having skill $S1$ with a level of 0.8. The same approach is used for all other skills which the workers share.

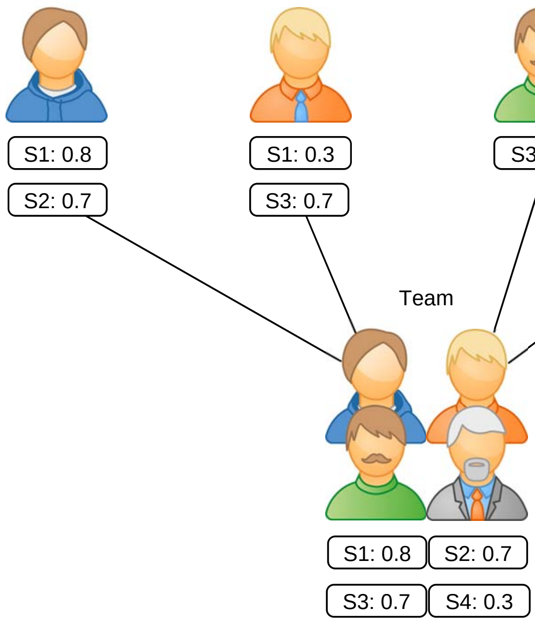

Figure 3.5: Skill-aggregation

The purpose of static teams is to exist for a long time, therefore the average collaboration-factor within static teams is very high, because all the team members are supposed to know each other and because they are used to work with each other frequently.

3.3.2 Dynamic Team

Our second approach introducing team-structures to task-based crowdsourcing is the *dynamic team* (DT). A DT is represented the same way as a static team. Upon forming a dynamic team it is represented by a virtual worker. However forming the dynamic team and also the aim of the dynamic team differ from a ST. A DT is created by one individual worker. The worker tries to create a DT, if he wants to solve a composite task.

However as the task requires more than one worker to solve, he asks his neighbours in the underlying collaboration network if they can help him solving the task. If enough neighbours agree to participate in finding a solution for the task, the single worker will register himself and the neighbours participating as a DT at the marketplace. The purpose of a dynamic team is to solve one specific task, upon completion of this task the DT is disbanded. This concept might also lead to teams, where the average collaboration factor is lower in comparison to ST teams, this is due to the fact, that for each new task a new DT is registered.

3.3.3 Static Versus Dynamic Teams

As the aim of this thesis is to find out which advantages and limitations arise by introducing static and dynamic teams to a task-based crowdsourcing system, we want to compare both approaches presented by us. Table 3.1 summarizes the differences between both team structures.

	Static team	**Dynamic team**
Purpose	solving multiple tasks	solving one task
Life span	undefined	for one task
Team size	static	tailored for solving one task
Collaboration	high	medium

Table 3.1: Static Versus Dynamic Teams

In this chapter we introduced a model for TC. We extended this generic model step by step to overcome limitations, which are present in TC systems. We defined an extended model for task-based crowdsourcing, which supports the collaboration between workers. Furthermore our model supports complex work, represented by a composite task. As a next step we defined two collaboration models, namely static and dynamic teams, and explained their role within the context of a TC system.

As a logical consequence we want to evaluate our extended model for task-based crowdsourcing. Therefore we need to implement the concepts presented in this chapter into a task-based crowdsourcing system. The next chapter demonstrates the implementation of all the concepts presented in this chapter within an existing crowdsourcing simulation framework. The focus will be on explaining how we have overcome the challenges of integrating teams and composite tasks. This includes presenting and algorithm for splitting and assigning a composite task to individual team members.

CHAPTER 4

Implementation of the Simulation Environment

The aim of this thesis is to extend the generic task-based crowdsourcing model, presented in chapter 3, with techniques for collaboration and to evaluate the advantages and limitations of the concepts presented in chapter 3. In chapter 3 we propose a generic model for task-based crowdsourcing. We extend the model by introducing a social collaboration network. Besides that the extended model supports complex work. This is accomplished by extending the simple task model state-of-the-art task-based crowdsourcing systems use. As explained in detail in chapter 3, we use a composite task for that purpose. Built on top of the integration of a social collaboration network and the support of complex work, we introduce two team structures, namely static and dynamic teams.

After introducing these concepts the next logical step is to evaluate the advantages and limitations of both approaches towards introducing teams, based on a social collaboration network, to task-based crowdsourcing. To perform an evaluation we need an environment to do so. The remainder of this chapter deals with the details of the environment used for evaluation, the extensions made to the chosen environment to support the extended model for task-based crowdsourcing. Besides that we describe in detail how an evaluation run is performed and what data will be evaluated.

4.1 A Simulation Framework for Crowdsourcing

In chapter 3 we introduce an extended model for task-based crowdsourcing. We want to evaluate the proposed extensions. Therefore we have to find a solution to perform evaluation. Three possible ways exist to accomplish this:

- Integration into an existing task-based crowdsourcing system, e.g. Amazon's Mechanical Turk
- Design and development of a new task-based crowdsourcing system.
- Usage of an existing simulation framework for task-based crowdsourcing.

The first option seems to be a theoretical one. The integration of a novel model for task-based crowdsourcing within an commercial crowdsourcing system, without having gathered data about the limitations and advantages of the proposed concepts, seems to be very unlikely. Option number two is out of scope of this thesis. There are existing frameworks to simulate crowdsourcing, e.g. work presented in [34] and [16]. Therefore, we choose to extend an existing task-based crowdsourcing framework. We want to use a framework which provides the following capabilities and features:

- **Task-based crowdsourcing**: The chosen system shall support the generic model for task-based crowdsourcing as described in chapter 3.
- **Extensibility**: Implementing new features like our approaches towards collaboration and the integration of the collaboration network should require as little effort as possible.
- **Configuration**: The framework should be highly configurable, so that simulation runs can be easily performed with different settings and scenarios.
- **Reporting capabilities**: Evaluation is a very important part of this thesis, therefore the framework has to have support for reporting data. The reported data is the foundation for performing an accurate evaluation later on.

We choose to implement the extensions to task-based crowdsourcing into a simulation framework for crowdsourcing (SFC) presented in [16] because it fulfils the requirements stated above.

SFC is a round-based simulation framework, which means that a simulation is performed within a certain amount of rounds. The marketplace is the component of SFC, which simulates a task-based crowdsourcing system. Workers and requesters, modelled as agents, communicate bidirectionally with each other and the market with the help of events. SFC uses a more specialized model of task-based crowdsourcing. Instead of allowing workers to pick tasks in a 'first-come-first-serve' matter auctions are used to assign tasks to workers. We do not see the usage of auctions as a disadvantage or a violation of the generic model for task-based crowdsourcing. Actually, the usage of auctions is a common approach used by other task-based crowdsourcing systems as described in [6], [2] and [34].

SFC provides an easy way to configure simulation with the help of Spring [36]. SFC also supports the reporting of data, acquired by a simulation. For a more detailed insight to SFC the interested reader may read [16].
The next subsection gives a more detailed overview of the interactions between requesters, workers and the marketplace and tries to give a better understanding on how SFC works.

4.1.1 Crowdsourcing With SFC

When a simulation with SFC is started the framework initializes itself. Next the workers and requesters register themselves at the framework. Then the normal crowdsourcing workflow is started. Figure 4.1 is an UML [40] sequence diagram, showing the basic interactions between requesters, workers and the marketplace. As shown in Figure 4.1 in each simulation round requesters create and submit tasks to the market. The market spawns an auction for every task, selects workers who are allowed to bid in the auctioning-process and informs those workers about the auctioned task. The informed workers have then the possibility to submit a bid for the task. They can submit only

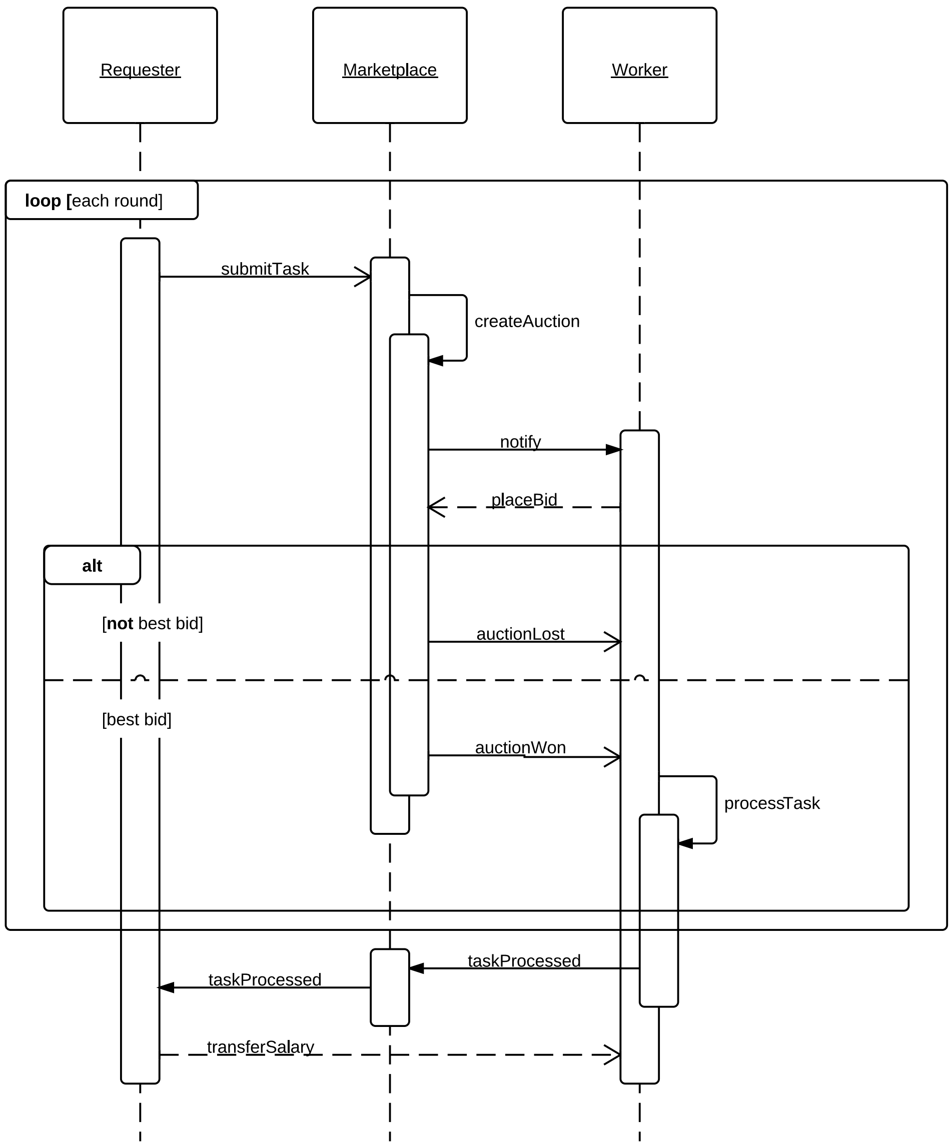

Figure 4.1: SFC Requester - Market - Worker Interactions

one bid per auction and round. The bidding strategies of workers and requesters can be configured; multiple strategies are supported. The market determines at the end of the auction if there is a winner for the auction. If yes, the task is assigned to the winning worker. The worker, who wins the auction, has to solve the assigned task. If the worker manages to process the task within the specified deadline, set by the requester, he will notify the market. Then the promised salary is transferred from the requester to the worker's account and the requester rates the worker according to the result produced by the worker when processing the task.

The next subsection demonstrates the architecture of SFC and defines the components, which need to be adapted and extended, to integrate complex work based on a collaboration network into SFC.

4.1.2 SFC's Architecture

We have selected SFC as the platform on which we want to implement our concepts of a collaboration-based social network and static and dynamic teams. Therefore we have to determine the areas, where we need to apply changes and where we need to add our extensions to SFC. Figure 4.2 gives an overview of the architecture of SFC and shows the areas, which need to be modified.

As shown in Figure 4.2 SFC's *SimulationController* is the core of the framework, it reads a *Configuration File* and configures the framework according to the configuration specified in the file. Configuration includes setting up reports for the *Reporting* component, defining with which settings the simulation is performed and which population is set up for a simulation. Besides that, the SimulationController is the component controlling the simulation itself. SFC is a round-based framework, therefore the SimulationController informs all other components about start and end of a simulation and about the start and end of a single round. The SimulationController uses events to accomplish this. The *MarketSimulation* is the core component for simulating the marketplace, like the SimulationController it sends events to all other components to interact with them, e.g. an event is that the auction has started. The *Marketplace* is closely related to the MarketSimulation, it includes the *Market*, where auctions are preformed. It reacts to the events triggered by the MarketSimulation. The population, which includes requesters and workers, interacts not directly with the market, it uses events to interact with the *MarketFacade*, which is an intermediary between the population and the market. Requesters send tasks to the marketplace to be auctioned, workers interact with the marketplace to bid for an auctioned task and to process the tasks acquired through

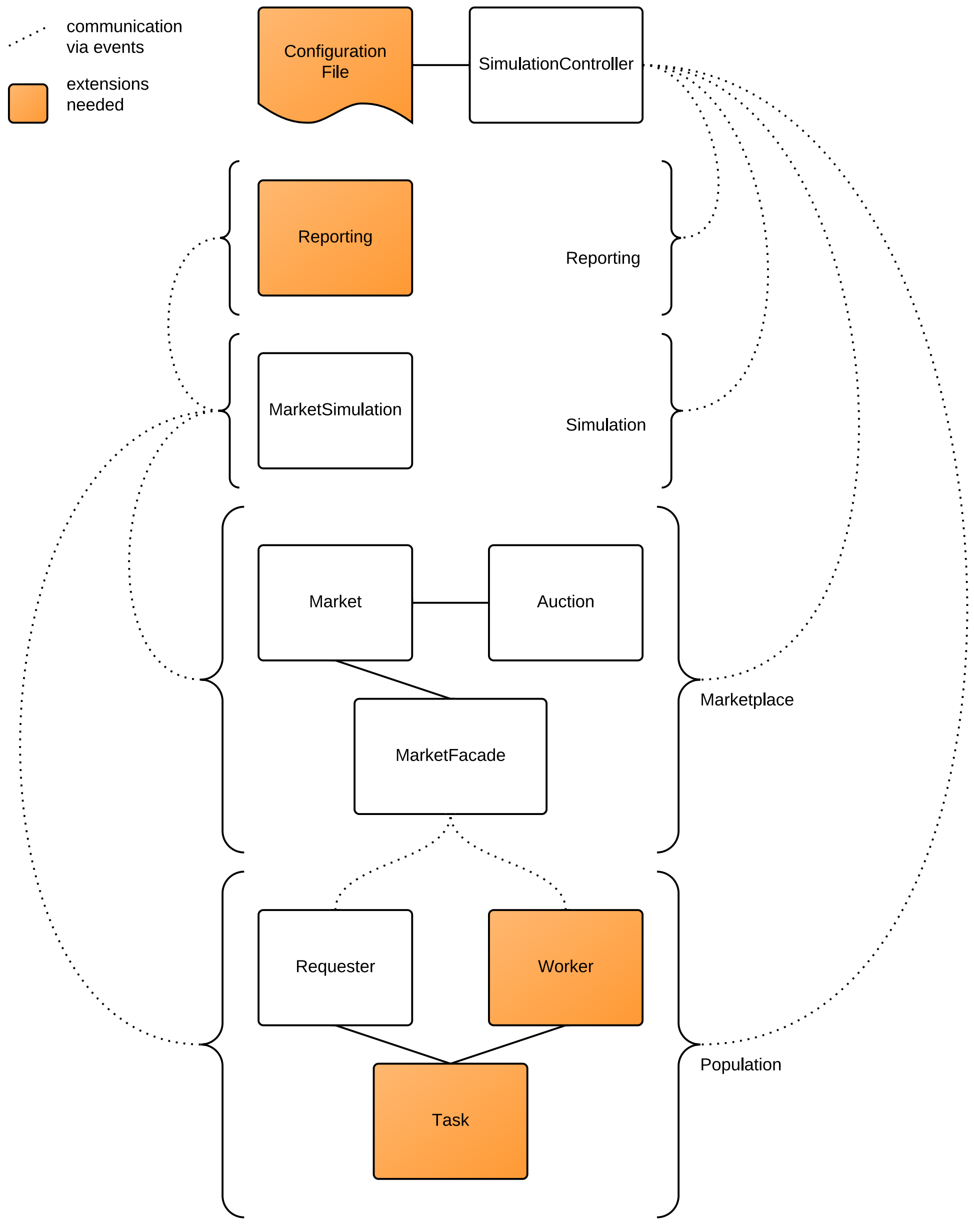

Figure 4.2: Extensions to the SFC Framework

auctions.

The structure of SFC makes it possible for us to implement our proposed extensions in an easy way. The marked components of Figure 4.2 show the areas which need to be adapted or extended. Besides changing the configuration to our needs, this includes

defining reports to collect the data used for evaluation as described in chapter 5. Furthermore to integrate complex work, we have to extend SFC's task model. Next, we have to extend SFC's worker model to introduce the collaboration network, on which we base our teams.
The next section will explain step-by-step how SFC is extended to implement the concepts, which have been introduced by us in chapter 3.

4.2 Extending SFC

So far we have introduced SFC, a simulation framework for task-based crowdsourcing. We have described, which parts of SFC need to be adapted, to include the concepts presented in chapter 3. To start with, we extend SFC to support a social collaboration network. Next we extend SFC's task model to introduce composite tasks. Based on these extensions, we explain all changes necessary to introduce static and dynamic teams.

4.2.1 Implementing the Social Network

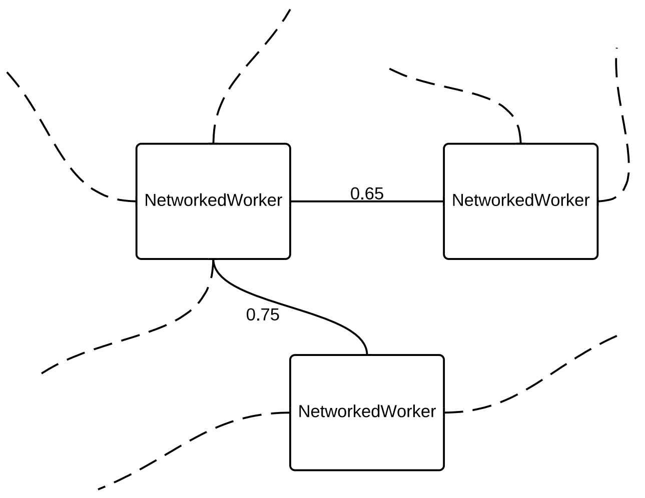

Figure 4.3: Implementation of the Collaboration Network

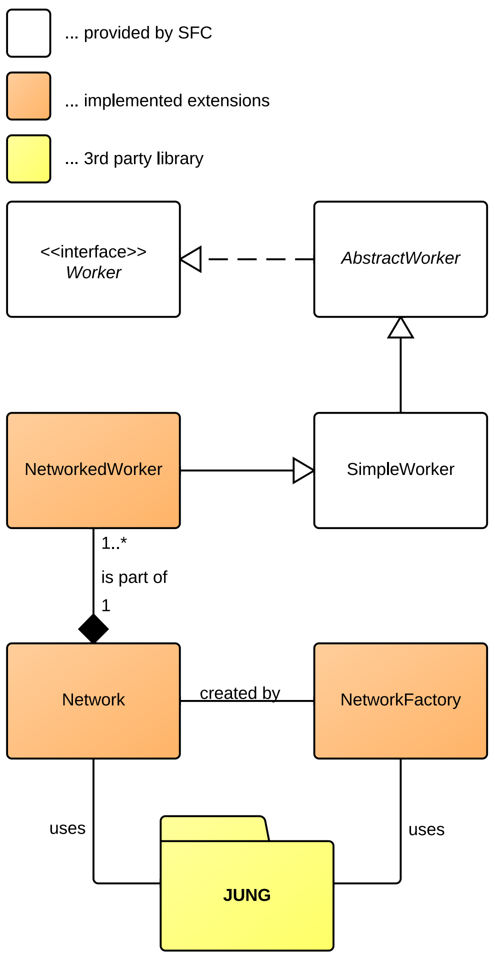

Figure 4.4: Extending SFC's worker model

As we explain in chapter 3, our social collaboration network is represented by an undirected labelled graph. We used the Java Universal Network/Graph Framework (JUNG) [13] to represent the graph. Figure 4.3 shows an abstract model of our network implementation. *NetworkedWorkers* are the nodes of the graph. The edges between the workers show the collaboration factor and represent the fact how well two workers can work together. Like in other task-based crowdsourcing systems, workers implemented in SFC cannot contact each other. Therefore each worker, being part of the social network, needs access to the network, due to that we extend SFC's woker model as shown in Figure 4.4. The NetworkedWorker, which we implement, is part of and has access to the underlying social network, which supports the following methods:

- Retrieve all workers
- Retrieve all neighbours of one worker
- Retrieve the collaboration factor
- Update the collaboration factor

The network implementation supports retrieving all neighbours of a specific worker within the network. This method is used in the context of creating dynamic teams, which we explain in detail in the next section. Retrieving and updating the collaboration factor are both necessary when a task is processed. The collaboration factor plays an important role in determining a tasks result and is updated when two workers work together. This will be explained in detail with the implementation of our teams.

As shown in 4.4 a *NetworkFactory* is introduced. Besides offering generic classes to model graph structures, JUNG provides us with a network generator for generating a random graph structures.

4.2.1.1 Kleinberg small-world network model

Our approach towards generating a random social network is based on a very simple network model introduced in [26] which follows the well known mathematical model for small-world networks defined in [43]. The 'Kleinberg small-world network' model uses the small-world property to generate a random graph structure. The small-world property is based on the corresponding small-world problem, which was introduced in [31]. The small-world property can be described as follows. Each individual in the social network has only a small amount of people it is connected to, however each individual in the whole social network is connected through a very small number of intermediaries to each other. Therefore by using the 'Kleinberg small-word network' model, we create a network which is split into cliques with a few remote connections to other cliques. A clique represents a group of workers which collaborate together. Hence, the team structures, which are introduced by us, heavily rely on these cliques.

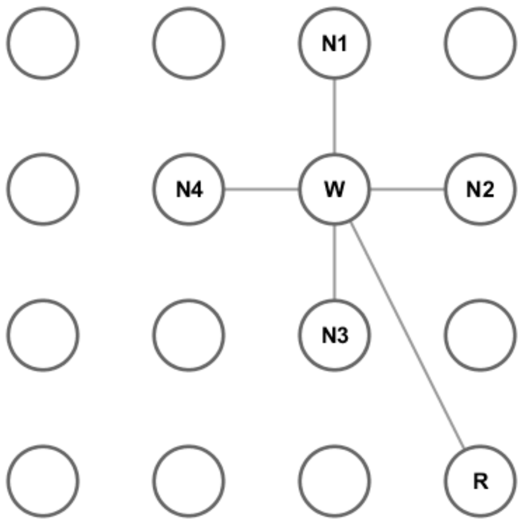

Figure 4.5: The Kleinberg small-world network model

Figure 4.5 shows the *'Kleinberg small-world network model'* used to generate our social network. The network is represented by mathematical undirected labelled graph, with and underlying lattice of size $n \times n$. Node W has 4 connections to his close neighbours $N1$, $N2$, $N3$ and $N4$, besides that W has one remote connection to a node R. This randomly chosen connection is selected with the probability defined by $r^{-\alpha}$, where r is the lattice distance, also known as M̈anhattend̈istance, of node W to node R and $\alpha \geq 0$ is the constant clustering exponent defined at creation time. We use the network generator to generate a social network, this includes generating workers and the dependencies between the workers. The collaboration factor, represented by the edges between the workers, is randomly uniformly distributed within the range of 0.0 and 1.0. SFC is configured to use the factory to initialize the collaboration network and the worker population at simulation start.

After introducing a new worker used by our collaboration network, the next subsection focuses on the extensions to SFC to support complex work.

4.2.2 Implementing Composite Tasks

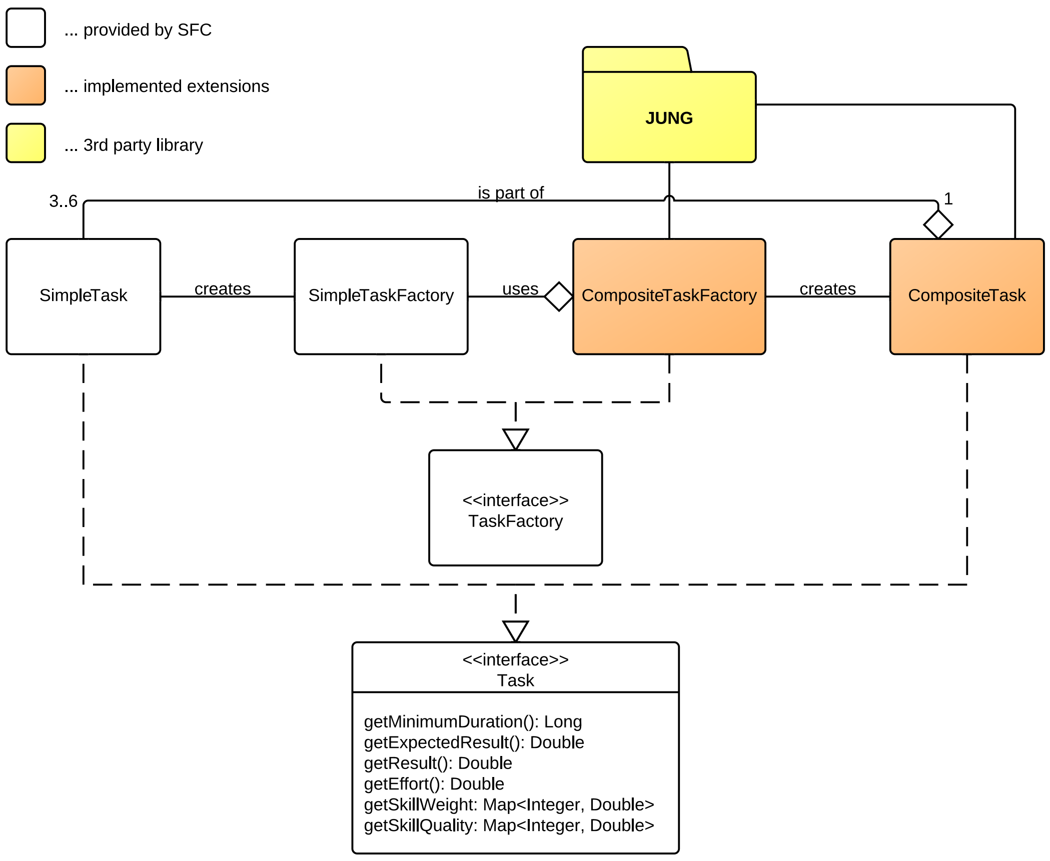

Figure 4.6: Extending SFC's task model

Figure 4.6 is an UML class diagram, showing the extensions made to SFC's task model. JUNG is used for the implementation of composite tasks. Again an undirected graph is used as the data structure to represent composite tasks. Each node in the graph represents a sub-task. An edge between two sub-tasks represents the fact that both tasks depend on each other. The value, the edge is labelled with, is the cooperation factor. Besides representing sub-tasks with the help of an undirected graph, composite tasks share the same interface introduced by SFC for tasks. Therefore, according to the contract defined by SFC's task interface, the following properties have to be implemented for composite tasks:

- **minimum duration**: As mentioned before SFC is round-based. Therefore a duration is defined in rounds. The maximum duration a simple task can have is defined as 24 rounds. The minimum duration defines the time a perfect worker needs to process a task. We defined the minimum duration of a composite task as the sum of all minimum durations of the sub-tasks. The right definition of the minimum duration is important as the framework uses the minimum duration for calculating the deadline of a task.

- **expected result**: The expected result is a randomly generated result between 0.0 and 1.0, it is used to determine later how well a worker performed and is randomly generated, when the task is created.

- **result**: The result is the outcome of the worker processing a task, nothing had to be changed here.

- **effort**: The effort for a simple task is defined by dividing the minimum duration by the maximum duration, which is as mentioned before 24 rounds. As the effort can be used by workers to determine how high a bid has to be, we use the average minimum duration of all-subtasks and divide it by the maximum duration. We chose this approach, because of the fact that the sub-tasks of a composite task are usually assigned to different persons.

- **skill quality and weight**: SFC uses the same approach as described by the generic model for task-based crowdsourcing, skills have two values, a quality determining how good a worker should be to perform the task, and a weight to show how important the particular skill is. The average quality and weight, calculated over all sub-tasks, is used as quality and weight for a composite task.

A factory has been implemented to automatically generate composite tasks. The factory is used during simulation by requesters to generate new composite tasks. For simplicity and performance reasons the factory implementation is limited to generate composite tasks with a number of three to six sub-tasks. We only support the creation of simple task as sub tasks, furthermore we generate the dependencies between the sub tasks randomly and assign to each dependency, represented as an edge, a random uniformly distributed cooperation factor as described in chapter 3.

So far the implementation of the collaboration network and the implementation of a composite task has been explained. Next, we explain how we implement our team structures on top of SFC.

4.2.3 Introducing Teams to Task-based Crowdsourcing

This section focuses on the implementation of both team structures. As the biggest difference between static and dynamic teams is the life cycle, we choose to base both approaches on the implementation of a common team worker. The team worker implements SFC's worker interface. To explain our implementation in detail we first have to elaborate on the details of workers in SFC. As in our generic model for task-based crowdsourcing each worker has multiple skills. Currently SFC supports a maximum of 5 skills. Each skill is represented by an integer, which is the unique id of a skill. For each skill the following three values are specified:

- **real performance**: The real performance determines the performance of a worker. The value is between 0.0 and 1.0 and is generated with a normal distribution. Mean and standard deviation have to be specified in the configuration file to run a simulation with SFC.
- **confidence**: The confidence in the worker's performance, a number in the range of 0.0 and 1.0, is set at creation time of a worker. The confidence is generated with a normal distribution. The mean and standard deviation for the confidence have to be specified in the Spring configuration file to run SFC.
- **observed performance**: The observed performance is the performance value observed by requesters. It will increase or decrease according to the workers rating. Initially the observed performance rating is determined with a normal distribution. The real performance is used as mean and $1.0 - confidence$ is used as the standard deviation. Skills are randomly initialized when a worker is created by the SFC framework.

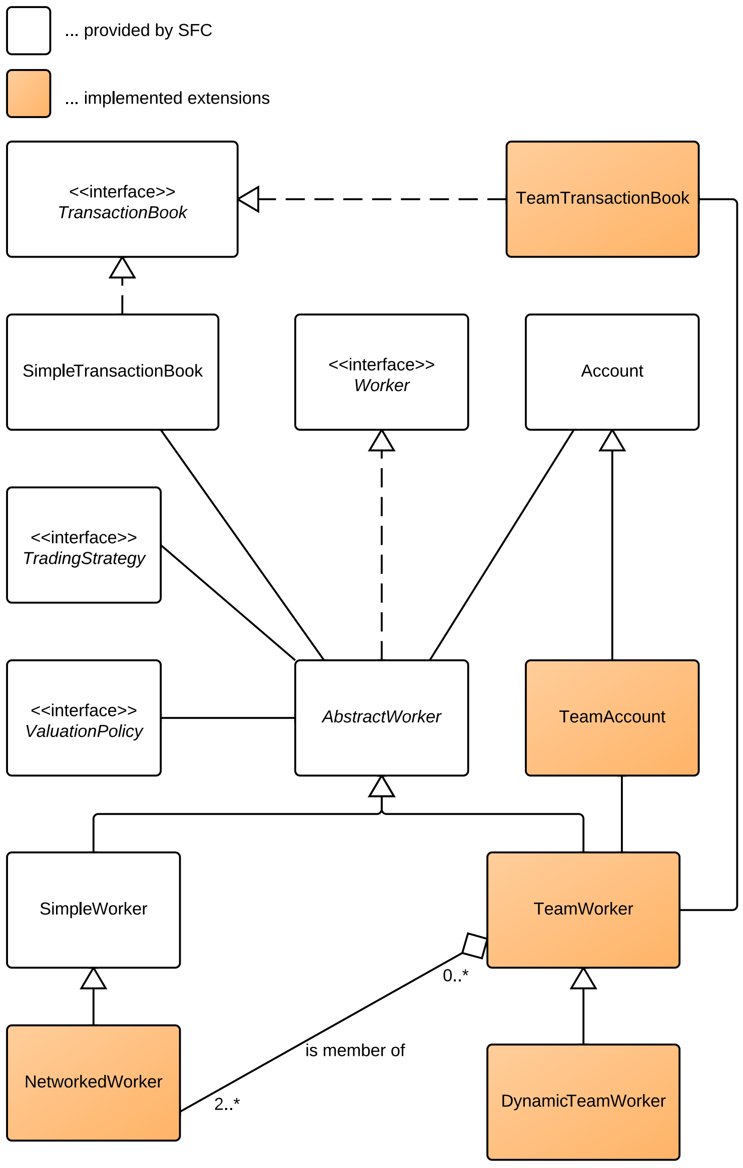

Figure 4.7: Implementing Teams

The next step to introduce teams to SFC is to extend SFC's existing model of workers with all the components needed to use teams later. Figure 4.7, an UML class diagram, shows the extension, which we introduce to SFC, to support teams. SFC's worker model is based on a *Worker* interface and an *AbstractWorker* implementation of the interface. The AbstractWorker is the class which needs to be extended to implement a

TeamWorker. Therefore we have to extend or use the four main components, which an AbstractWorker uses to interact with SFC. This four components are:

- **valuation policy**: The valuation policy determines the price a worker is willing to bid for an auctioned task. SFC supports different valuation policies. Chapter 5 explains in detail which valuation policy is used for simulation.
- **trading strategy**: SFC supports different types of trading strategies. A trading strategy is used to change the price, determined by the valuation policy before, according to a set of rules and facts determined by the strategy. E.g. a rule could be that a worker decreases or increases the price depending on the current workload.
- **transaction book**: The market assigns a transaction for each won auction to the worker. A transaction includes a reference to the requester, the winning worker and the task. It specifies the requester price, the worker price and the price for which the auction was cleared. Besides that it also specifies the deadline, until the worker has to finish the auctioned task. All the transactions assigned to a worker are processed with the help of the transaction book. The transaction book is the place where the time, a worker needs to process a task, is determined and the result of processing the task is calculated.
- **account**: All funds earned by a worker by processing an auction successfully are stored in the account. The price to be paid is determined by the result of the auction, and is as described before stored in a transaction.

For a detailed explanation of all components described above the interested reader is referred to the work presented in [16].

As Figure 4.7 shows, we introduce a *TeamTransactionBook* and a *TeamAccount* to implement teams properly. This is necessary, because athe *SimpleTransacationBook* can only handle normal SFC tasks. We also had to adapt the account, as it was implemented to be used by a single worker and not by a team of workers. Therefore as the transaction book is the core component for processing and handling tasks, we start by implementing a transaction book which allows us to work with the composite tasks, which we have introduced before. Our transaction book for teams has to do the following tasks:

- **Handle composite tasks**: Handling of composite tasks includes splitting up complex tasks into parts, which can be solved by team members. It also means that the transaction book has to make sure that the parts can later be put together again.
- **Process composite tasks**: Processing of composite tasks means, that all sub-tasks assigned to members have to be put together when they are finished. A result for the composite task has to be determined. To determine a result accordingly, it is necessary to include the cooperation factor between sub-tasks and the collaboration factor between the team members.

The next three subsections will focus on both items mentioned above. First we describe in detail how composite tasks are split up. Next we will explain how a group of workers is selected for processing a sub-task and then we show how the result of a composite task is calculated. After that we explain how we implemented the *TeamAccount* shown in Figure 4.7 and introduce static and dynamic teams.

4.2.3.1 Handling Composite Tasks

To handle and process composite tasks we had to implement a transaction book which supports composite tasks. The implementation we have chosen, splits a composite task into sub-tasks and assigns each sub-task to one or more workers.

Algorithm 4.1 describes how we split a task. The algorithm itself is split into two parts. The first part assigns a composite task. With the help of the function *map* we introduce a bidirectional mapping between the parent task and the composite task we want to assign. We need to establish this mapping for the following reasons:

- Tasks are processed by the team members. Upon completion of a task, a team member informs the worker, that the task is completed. The team worker checks the parent of the task to determine if the parent can be processed and finished.
- A parent task is processed, when all its sub-tasks have been processed. Therefore a parent task has a list of all the sub-tasks which are not processed.

The communication between the team worker and its team members is done via the help of team transactions. As described before, the market assigns a transaction to the winner of an auction, to be able to use the same interface to assign team tasks we derived our team transaction from the normal transaction. The only difference to a normal transaction is that a team transaction stores a reference to the team which created the transaction. This reference is used by a team member to notify the team when the member finishes a task, which was assigned by the team. After mapping the task and its parent, we invoke *assign* recursively for each sub-task st of the task to be assigned.

The second part of the algorithm assigns simple tasks to networked workers. In most cases a simple task is assigned to one simple worker, however as we have an environment with multiple skills, it can be the case that simple tasks require more skills than each single team member has. Therefore, more than one worker can be assigned. Then for each worker a bidirectional mapping between parent and task is made. This ensures that a parent task is only completed when all its sub-tasked are processed, even when one sub-task is distributed to more than one worker to process.

Algorithm 4.1: assign(Task task, CompositeTask parent)

```
input: task: the task to be assigned, parent: the parent of the task to be assigned
if task is a CompositeTask then
    map(task, parent);
    createTransaction(task, parent);
    foreach sub-task st of task do assign(st, task);
end
else
    team ← selectTeam(task);
    transaction ← createTransaction(task, parent);
    foreach worker w of team do
        assignTransaction(w, transaction);
        map(task, parent);
    end
end
```

4.2.3.2 The Team Selection

This section focuses on the design of the *select* function, which was introduced in Algorithm 4.1.. The *select* function determines, to how many team members a task is distributed for processing. A team worker only participates in an auction if the team has all skills required by the auctioned task. To accomplish this, we merge all skills of the team members together. Merging is performed by selecting from each available skill the best confidence and performance value the group of members offers. Therefore we reduce the problem of selecting a suitable team to the following.

We have a task which requires n skills to be processed, and we have x workers, which each have a subset k of the required n skills. Therefore it can easily be seen, that the problem is an instance of the NP-hard set covering problem (SCP) [4]. We use a greedy algorithm to overcome the SCP problem. The input to our algorithm is a set of workers and all required skills of the task for which we want to find a suitable team. Our algorithm uses the following steps:

1. Remove all workers who have no required skills.
2. Sort the workers according to the number of required skills they have.
3. Select the worker with the most required skills. If more than one worker has the same amount of required skills, choose the worker with best average real performance.
4. Remove all required skills, which the selected worker has, from our set of required skills.
5. Stop if no skills are required any more or no workers can be chosen, else start with step 1 again.

To achieve better results, we perform this algorithm several times. As a start the algorithm searches for a group of workers including all team members. If a group of workers can be found, then we continue with a sub set all team members. If a team consists of x team members, then we try this approach x times. Each time we take all team members and remove one them. The reduced group of people is then used as the input to the algorithm stated above. This strategy of team selection leads to the possibility that we have a set of teams, which are suitable to perform the specified task. Therefore, we choose the team, which has the highest *suitability factor*. The suitability factor is the average real performance of all team members multiplied by the result of 1.0 minus the team's workload. We use this approach to distribute the workload within a team as best as possible, while keeping a maximum level of performance.

4.2.3.3 Processing Composite Tasks

So far we have implemented a team transaction book, which supports splitting up complex tasks. Besides that the team transaction book uses an algorithm to select members, to which tasks get assigned. Therefore as a next time, we have to implement an algorithm to determine a composite task's result. We have multiple sub-tasks, and it is possible that each sub-task has more than one result, therefore determining the result of a composite task becomes more difficult. Listing 4.1 describes how a result is calculated. Besides having multiple results for one task, we have to in-cooperate the collaboration factor, which shows how well workers work together, and the cooperation factor, which shows how much cooperation is needed to solve the sub-tasks of a composite task successfully.

```java
double maxCooperation = 0d, maxCollaboration = 0d;
double avgDifference = 0d;

for (int i = 0; i < subtasks.size(); i++) {
  Task task1 = subtasks.get(i);
  avgDifference +=
      Math.abs(task1.getResult() - task1.getExpectedResult());

  for (int j = i + 1; j < subtasks.size(); j++) {
    Task task2 = subtasks.get(j);

    double localCooperation = composite.getCooperation(task1, task2);
    maxCooperation += localCooperation;

    List<NetworkedWorker> workers1 =
        transactions.get(task1).getWorkers();
    List<NetworkedWorker> workers2 =
        transactions.get(task2).getWorkers();

    double avgCollab = 0d;

    for (NetworkedWorker w1 : workers1) {
      for (NetworkedWorker w2 : workers2) {
        double collab = worker.getCollaboration(w1, w2);
        updateTeamwork(collab, localCooperation, w1, w2);
        avgCollab += collab;
      }
    }
    avgCollab /= (double) workers1.size() * workers2.size();
    maxCollaboration += avgCollab * localCooperation;
  }
}

avgDifference /= (double) subtasks.size();
double ratio = maxCollaboration / maxCooperation;

if (ratio > 1) {
  ratio = 1.0;
}

double result = random.nextGaussian(
  expectedResult, avgDifference * (1.0 + (1.0 - ratio)));
```

Listing 4.1: Determining the result of a composite task

The algorithm shown in Listing 4.1, is a simplified version of the algorithm implemented in our extension to SFC. It calculates three basic values and determines a result based on these three values. The maximum cooperation represents the sum of every single cooperation factor between two sub-tasks of the composite task. The maximum collaboration represents the sum off all average cooperation between the workers of one subtask to another. The third value necessary to calculate a result for our composite task, is the average difference between the result and the expected result of each sub-task. We determine the three values between line 4 and line 39 of Listing 4.1. Next we determine the ratio between maximum collaboration and maximum cooperation. This ratio shows us if our team is working good enough together to accomplish the complex task with a good result. On line 41 the result is randomly generated by a normal distribution. The mean of the normal distribution is the complex task's expected result. The standard deviation σ is determined by the following formula:

$$\sigma = avgDifference * (1.0 + (1.0 - ratio))$$

The calculation of the result of a simple task is performed with the same algorithm, the only difference in that case is that the cooperation factor will be 1.0 as distributing one task over several workers requires as much cooperation as possible to solve this simple task.

We also want to point out that on line 26 the teamwork between two workers is updated. We update the teamwork according to the following formula:

$$teamwork = teamwork + cooperation * (1.0 - teamwork)$$

This formula guarantees us that the teamwork cannot exceed a value of 1.0, moreover the teamwork increases as the cooperation, which is required to solve the task, enhances. We take the assumption here that a training effect applies to teamwork, that means the more you work together, the better the teamwork gets. Therefore, if a higher amount of cooperation is required the workers benefit from it.

4.2.3.4 Team Account

Besides changing the transaction book, we had also to change another component, namely the *Account*. Our implementation of an account distributes the team income to all team members in equal parts. We have chosen this simple implementation because we imply that the workload in a team is distributed evenly over all team members. Hence, there was no need to implement a more complex team account.

So far we have implemented a new transaction book and a new account. Two core components, namely trading strategy and valuation policy are still missing. However it was not necessary to change this part of the code, as the representation of our complex task was implemented according to the contract SFC specified. Therefore we can use the standard implementations of trading strategies and valuation policies.
As shown in Figure 4.7 this means that we have defined all the core components of our team worker. Therefore we can introduce a *TeamWorker* and can describe in detail how static teams and dynamic teams are created and how the life cycle of both team structures looks.

4.2.3.5 Static Teams

As shown in Figure 4.7 we implement a *TeamWorker*, which represents a static team. Static teams are created at simulation start, directly after the creation of the network. We randomly generate teams with the help of a normal distribution $\mathcal{N}(5.0, 2.0)$. We have chosen this size due to the fact that our composite tasks have a maximum of 5 subtasks. We generate static teams in a way that each team member knows all other team members. We also increase the collaboration factor between static teams slightly. We do this due to the assumption, that people willing to create a static team, have a better understanding of each other. Hence those people can work together better. As defined in chapter 3 static teams exist from the start to the end of the simulation, we do not change the structure of teams, nor are teams dissolved in the mid of the simulation.

4.2.3.6 Dynamic Teams

In contrary to static teams, dynamic teams are not created at simulation start. Therefore, we had to implement a specific implementation of our team worker for dynamic teams. The only change we implement for dynamic teams is that dynamic teams automatically dissolve themselves after finishing their task, besides that dynamic teams do not participate in any auction. We also have to extend our implementation of a networked worker to implement dynamic teams in the following way:

- If a networked worker receives the notification from the market that an auction for a composite task starts, the networked worker asks his neighbours via the social network if they want to help him in solving the task.
- Each neighbour is a networked worker himself. Currently a neighbour accepts a request to solve a composite task with another neighbour if the worker's own workload is smaller than 80
- If a networked worker receives enough positive answers from his neighbours, he uses the team selection strategy described before to form a team to solve the task.
- If a suitable team for solving the composite task is found, the networked worker who initiated the team finding process, spawns a dynamic worker and assigns the auction to the dynamic worker.
- The worker spawning the dynamic worker can be seen as the team leader. The trading strategy and the valuation policy of this worker are used by the dynamic worker to make a bid for the auction.

These are the only changes necessary based on our implementation of a team worker.

4.3 A Simulation Run With SFC

We have introduced all the extensions to SFC, which are necessary to perform an evaluation. To perform an evaluation and to retrieve data, which can be used to show the advantages and limitations of dynamic and static teams we need to perform a simulation run with SFC. We first have to explain, what SFC does on a simulation start.

4.3.1 Simulation Start

On simulation start, SFC's *Simulation Controller* initializes the framework according to the configuration. This includes the following steps:

- The *Reporting* component gets set up according to the configuration specified. The reports are used to retrieve data, which is generated during the simulation.
- The *MarketSimulation* is configured. SFC's MarketSimulation component is responsible for initializing the *Market*, besides that it handles the interaction between Requesters, Market and Workers.
- After setting up the MarketSimulation, SFC initialzes the population. The population includes requesters and workers.
  - Requesters are set up according to the configuration. This includes configuring a valuation policy and a trading strategy. Besides that requesters get a task factory assigned. This task factory is used to generate tasks later. We use the task factory to generate composite tasks as described in section 4.2.2.
  - Besides the requesters, workers have to be set up. We have extended SFC to use a social collaboration network. This network has to be generated at simulation start. It includes all workers. We use the factory, described in section 4.2.1.1, for this purpose.
  - After creating the social collaboration network, static teams can be generated. Once again we use a factory to accomplish this. The details on how static teams are created are described in section 4.2.3.5.
- All requesters and workers are registered at the *Market*.

After all the initialization tasks have been performed, the actual round-based simulation is started. The *SimulationController* notifies all components of the framework about this.

4.3.2 Simulation Round

The two groups of actors, namely requesters and workers, interact each round with the *Market* component. We try to give a chronological overview, on the actions which are performed:

- When a new round is started, all requesters are informed. A Requester has then the choice to create composite tasks. The task is sent to the Market for auctioning.
- The market receives the task and spawns a new auction. Then the market informs all workers (including static teams) about the new notification.
- The workers are then allowed to bid for the task. We have to distinguish here between two cases:
  - Static teams are participating, they just send a bid for the task to the market.
  - Normal workers, implemented as *NetworkedWorkers* by us, will try to spawn a dynamic team. If this is successful, then the dynamic team will send a bid in for the task, else no bid will be send in.
- At round end the Market checks all started auctions. Each auction is cleared. We have to distinguish again between two cases:
  - No valid bid was submitted. The auction gets closed and the task is not assigned.
  - At least one valid bid is submitted. According to the auction, the winner is determined and requester and worker are notified. Besides that, the task is assigned to the worker.
- If a task is assigned to a worker, the worker (either a dynamic or a static team) has to distributed the task to its members. This is accomplished as described in section 4.2.3.1. The team use the algorithm described in 4.2.3.2 to form a team.
- When a round is being finished, the market informs as well the workers. The workers have to check then if a task, which has been assigned before, has already

been processed. As described in section 4.2.3.3 processing a task is a non trivial task. We use an algorithm which determines the result of a composite tasks by calculating a factor describing the proportion of required cooperation to the offered teamwork of the team. If a task is processed, the worker sends the task to the market.

- When the market receives a finished task, the market informs the requester. Next the worker is rated according to the quality of his result. As the last action the salary is transferred from the requester's to the worker's account.

To be able to perform an evaluation, we use a report, which records every transaction happening between requester and worker. A transaction is created by the market upon receiving a new task from the requester. The transaction shows the result of the auction, besides that it also shows if a task was processed, when a task was processed and how well a task was processed. Besides that every round each worker registered at the market is recorded as well. The records include information about skills, if the worker represents a team, additional information is recorded. The information includes the average teamwork within the team and and the team's size.

4.3.3 Simulation End

At simulation end, the SimulationController stops the MarketSimulation. A notification to the workers is send that the simulation has stopped. All collected data is written to the report files and the framework shuts itself down.

In this chapter we explained the implementation of the extensions to SFC, a task-based crowdsourcing simulation framework, which we proposed in chapter 3. First we explained how the different components of SFC interact with each other, then we introduced the implementations of composite tasks and the collaboration network. Based on these implementations we introduced step-by-step all extensions and adaptations to SFC to support static and dynamic teams. The next chapter focuses on the evaluation of both team-based approaches and shows advantages and limitations of both approaches.

CHAPTER 5

Evaluation

The previous chapter explains the implementation of the concepts and data structures introduced in chapter 3. This chapter's purpose is to give an overview of the evaluation we have performed, to show advantages and limitations of the extensions, which we have proposed and implemented as described before. The chapter is split into three sections. The first section explains in detail the configuration which is used to perform the evaluation. The second section explains the scenarios, which we use for our evaluation. The third section discusses the results of the evaluation and summarizes the advantages and limitations of static and dynamic teams operating within a task-based crowdsourcing system.

5.1 Configuring SFC

We perform our evaluation with the help of SFC, which is a task-based crowdsourcing simulation framework. Therefore we have to configure SFC accordingly to perform our evaluation. SFC is configured with the help of a Spring configuration file. We give an overview of our configuration in this section. For a detailed insight into the configuration, the interested reader is referenced to the example configuration for a simulation for static workers, which can be found in Appendix A.

Simulation	
# of simulations	1
# of rounds	500
Auction	
Type	second price sealed-bid auction
Duration	1 round
Requester	
Valuation policy	quality and effort aware
Trading strategy	truth telling
# of tasks per round	1
Networked workers	
Valuation policy	skill and effort aware
Trading strategy	truth telling
Social (collaboration) network	
# of workers	2500 (graph lattice size 50 x 50)
clustering exponent	0.55

Table 5.1: Basic framework configuration

Table 5.1 gives an overview of the basic configuration of the SFC framework. Each scenario, which we introduce in the next section, will be performed as one simulation run with 500 rounds. The first property we have to configure is the auction type we want to use. SFC offers different kind of auctions that can be used to perform a simulation. All auctions are reversed auctions. The following auction types are supported:

- Dutch auction
- English auction
- Second price sealed-bid auction
- Continuous double auction

We use the second price sealed-bid auction for our evaluation. Sealed-bid auction means that all bids are secret, the workers and requesters do not know what other workers are bidding. Second price means that the auction winner, which has made the bid with the lowest price, has to pay the second lowest price. We have chosen this auction type, because as described in [16] the second price sealed-bid auction leads to slightly better and fairer results than the other auction types. The duration of each auction, started by SFC, is configured to be a single round, that means the workers have to send in their bids within one round. The winner of the auction is determined at the end of the round.

As a next step we have to configure the requesters, which participate in our simulation. Requesters use a quality and effort aware valuation policy, that means that the price p is calculated depending on the weighted quality q and the effort e. The weighted quality q is determined as the average of quality times the weight for each skill.

$$\mu = max(e, q)$$

$$\sigma = min(e, q)$$

$$p = \mathcal{N}(\mu, \sigma)$$

The price p is defined according to the normal distribution stated above. As a trading strategy we use a truth telling strategy. This means that the price, defined by the valuation policy, is not changed and is used as the maximum price the requester is willing to pay for a task. Each requester creates one task each round. The task gets then submitted to the market to be auctioned.

Next we have to define the worker population. The networked workers, that we introduced in chapter 4, are configured with a skill and effort aware valuation policy. The price is determined by two factors. The first factor, incorporating the quality and effort, is calculated exactly the same way as the price for a requester. The second factor is calculated as the average proportion of the task's required quality to the worker's real performance. We also use a truth telling strategy as trading strategy, hence the price calculated by the valuation policy is not changed by the trading strategy.

As explained in chapter 3 and 4, the underlying collaboration network is generated randomly by a factory. The Kleinberg small-world generator provided by JUNG, is used by the factory. We configure the generator with a $n * n$ lattice size of 50, resulting in a worker population of 2500 workers. As there was no information available for selecting a clustering exponent for a collaboration network, we choose a factor of 0.55.

A summary of the most important configuration settings is shown in Table 5.1, the next section will introduce the scenarios we use to perform the evaluation.

5.2 Evaluation Scenarios

So far this chapter explained the basic configuration of the SFC framework, this section focuses on explaining the scenarios, that we use for evaluation. We define six scenarios. Each scenario is performed by static teams and dynamic teams separately resulting in a total amount of 12 simulation runs. The aim of the scenarios is to simulate a system with low, medium and high workload for the workers. To compare all 12 simulation runs, each run uses the same collaboration network, representing 2500 workers. Therefore the only possibility to increase or decrease the workload of the workers is to vary the number of requesters. Hence 20, 40 and 60 requesters are used to simulate a task-based crowdsourcing marketplace with low, medium and high work load. Besides that, the focus is also on the workers themselves. As described in chapter 4 the workers' performance and confidence can be configured via SFC's configuration files. We choose, for each of the three scenarios described before, the following two worker configurations:

```
<bean id="highPerformanceConfig"
        class="at.ac.tuwien.dsg.crowdsim.agent.worker.SkillConfig">
    <property name="meanPerformance" value="0.7"/>
    <property name="deviancePerformance" value="0.25"/>
    <property name="meanConfidence" value="0.8"/>
    <property name="devianceConfidence" value="0.25"/>
</bean>
```

Listing 5.1: High performance worker configuration

```
<bean id="lowPerformanceConfig"
        class="at.ac.tuwien.dsg.crowdsim.agent.worker.SkillConfig">
    <property name="meanPerformance" value="0.3"/>
    <property name="deviancePerformance" value="0.25"/>
    <property name="meanConfidence" value="0.8"/>
    <property name="devianceConfidence" value="0.25"/>
</bean>
```

Listing 5.2: Low performance worker configuration

Listing 5.1 shows a worker configuration with a high performance value of 0.7, Listing 5.2 shows a configuration with significant lower performance of 0.3.

The three scenarios concerning the system's workload, combined with the two different worker configurations, each performed with static and dynamic teams separately lead to 12 simulation runs, as shown by Table 5.2.

worker performance / **# requesters**	**0.3**	**0.7**
20	D1 S1	D2 S2
40	D3 S3	D4 S4
60	D5 S5	D6 S6

Table 5.2: Evaluation scenarios

The simulation runs, shown in Table 5.2 are labelled with SX and DX. A label starting with S represents a static team run and a label starting with D represents a dynamic team run.

Static teams are generated at start, the team size is normal distributed with a distribution of $\mathcal{N}(5, 2)$, workers are randomly added to a team. After adding workers to a team, it is assured that all team members know each other, that means that we add edges to our collaboration network if necessary. We also boost the team's collaboration factor slightly. We argument this by the fact that a group of people most likely only registers itself as a static team if the team members know each other and if the team members are sure that they can work together well.

To compare the results of dynamic and static teams better, we implemented a worker selection policy for the dynamic team runs. That means that SFC allows 500 dynamic teams to participate within a single auction. This is reasonable because at the start of a static simulation run 502 static teams are generated. The number of static teams is determined, by the settings chosen for social collaboration network and the normal distribution used to generate static teams. Therefore the number of static teams is for each simulation run constant.

5.3 Evaluation Results

To begin with the results of our evaluation, we first give an overview of the worker population. We show how the skills are distributed and calculate the average worker performance. Based on that we compare the simulation runs of static and dynamic teams with the same scenario. Therefore we will compare simulation run S1 with D1, S2 with D2 and so on. The focus of the evaluation is on the following properties:

- **Team size**: We use histograms to present the team size of all static and dynamic teams, that have performed a task successfully.
- **# of tasks submitted/ finished**: We compare the number of tasks assigned and finished in each simulation. This should give insight on how well static and dynamic teams are reacting to higher and lower load rates.
- **Quality**: Besides checking the quantity of finished tasks, we check also if there are significant differences in the quality of tasks solved by dynamic and static teams.

Both quantity and quality are important properties to evaluate. The quantity is a measurement of how high the throughput of the crowdsourcing platform is. As crowdsourcing is gaining more and more interest, throughput is becoming more and more important, as more tasks get created and processed on crowdsourcing platforms. Therefore we want to check, which of our collaboration based approaches has a better throughput rate. Besides that quality is also a very important aspect of crowdsourcing, however a balance between quality and quantity has to be found. This means that a successful collaboration-based approach like dynamic or static teams has to maintain a certain level of quality while processing tasks with an acceptable speed.

We retrieve all the data, used for the evaluation, with the help of SFC's reporting capabilities. We use two reports for collecting all the data necessary for our evaluation. Both reports use a CSV format.

- **Transaction report**: The transaction report collects the data of all transactions performed by the system. The most important information the report includes is, if a task was assigned, successfully finished or if a deadline violation occurred. Besides that, the report shows us the quality of a successfully processed task.
- **Worker report**: The worker report is round-based. It shows us all registered workers at the market each round. It shows the worker performance and gives us information regarding the team's size and the average collaboration factor of a team.

Based on the scenarios described in the last section the evaluation is performed. As a start we focus on the workers and show the skill distribution of the worker population. Next we give an overview of the team size used during the simulations and compare the size of the static teams to the size of the dynamic teams. After doing so, we focus on the task throughput during simulation. We show detailed statistics about the submitted, assigned and successfully finished tasks and compare static and dynamic teams. Finally we change the focus from quantity to quality. We show how the ratings for successfully finished tasks are distributed and compare the average ratings of our static and dynamic approaches.

5.3.1 Worker and Team Population

Figure 5.1 shows the skill distribution of all 2500 workers of the collaboration network. We show distributions for the high and low worker configurations, which were explained

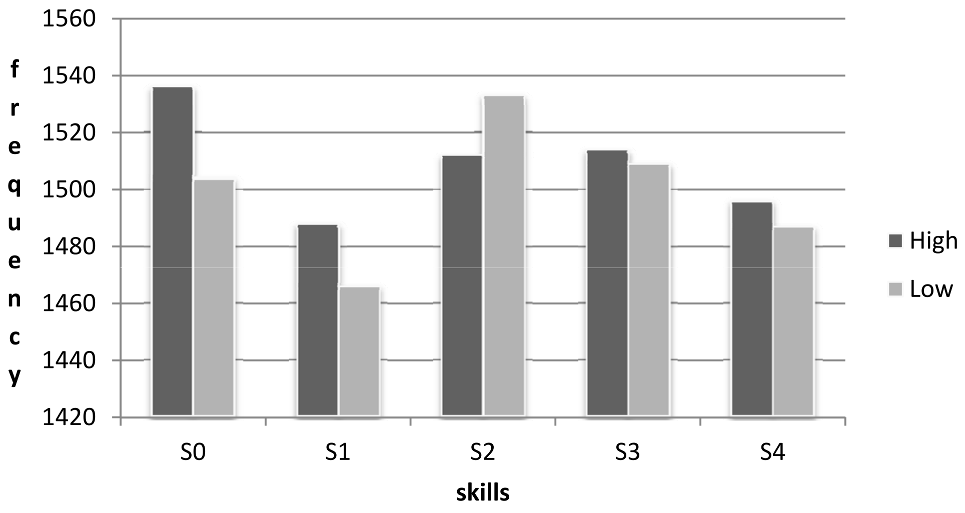

Figure 5.1: Worker skill distribution

in the section before. The X-axis shows the five different skills, which are supported by the worker population, on the Y-axis we see the frequency of the skills occurring. As we can see the skills are not as evenly distributed. The skill S1 is occurring less in the worker population in comparison to the four other skills S0, S2, S3, S4.

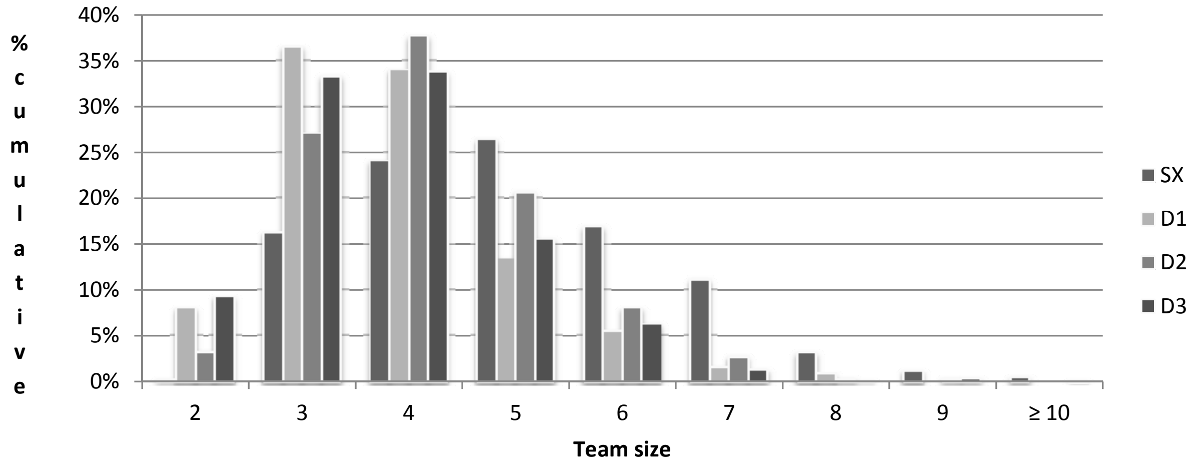

Figure 5.2: Distribution of team size 1

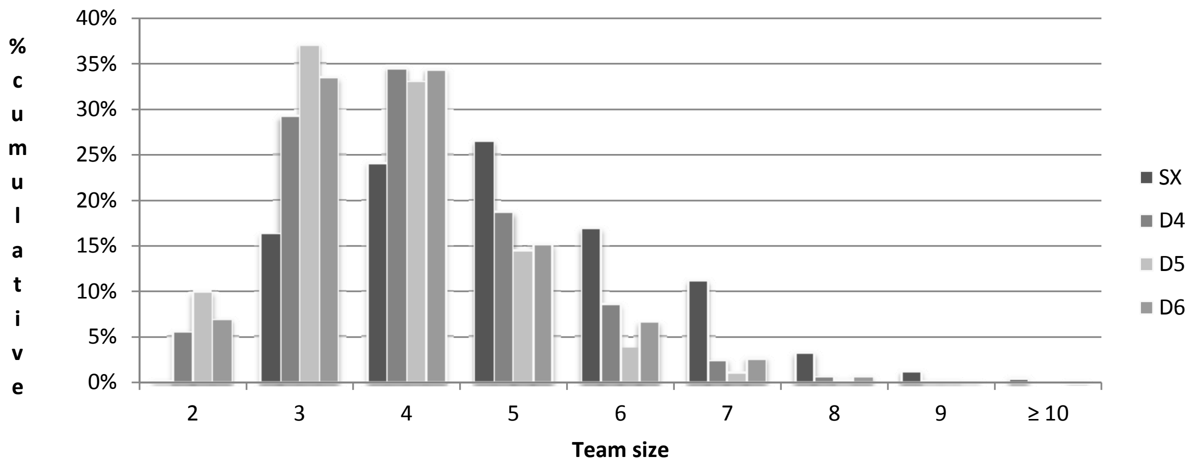

Figure 5.3: Distribution of team size 2

	SX	D1	D2	D3	D4	D5	D6
# teams	502	834	811	1597	1568	2251	2227
ACF	0.75	0.62	0.62	0.62	0.64	0.59	0.60

Table 5.3: Created teams

Figure 5.2 and Figure 5.3 show the distribution of the team-sizes. In the interest of clarity, we have made two diagrams, each diagram compares the static team size distribution, shown as SX, with three dynamic teams. The X-axis corresponds to the team size. The Y-axis shows the series labels, which correspond to the ones shown in Table 5.2. Series SX represents the static teams, as those are always created with the same algorithm the results regarding the team-size are the same for all simulation runs. As expected the size of dynamic teams is varying, the size gets always adapted to the needs of a task. Therefore dynamic teams have always a high team utilization close to 100%, static teams cannot compete with this due to the fact that they are created at simulation start and that their size is not adaptable during simulation. Besides the information shown in Figure 5.2 and Figure 5.3 Table 5.3 gives an overview of the number of teams created. Moreover, the second line of the table shows the average cooperation factor (ACF). As we can see, static teams have in general a higher ACF. The increased ACF of static teams is based on the reasoning that the members of static teams know each other and work better together, because they actively register as a static team. Therefore we imply that there is a higher commitment to the team and the work performed by a static team in comparison to a dynamic team.

5.3.2 Task Throughput

scenario	submitted	assigned	completed	% assigned/ submitted	completed/ assigned
S1	796	429	301	54%	70%
D1	1194	789	731	66%	93%
S2	819	520	315	63%	61%
D2	1172	776	712	66%	92%
S3	1416	699	416	49%	60%
D3	2287	1512	1356	66%	90%
S4	1609	984	552	61%	56%
D4	2255	1522	1355	67%	89%
S5	2034	923	516	45%	56%
D5	3234	2083	1778	64%	85%
S6	2454	1452	786	59%	54%
D6	3208	2122	1784	66%	84%

Table 5.4: Overview of the 12 simulation runs

The data presented in Table 5.4 shows the task throughput of all simulation runs. The table shows the number of submitted, assigned and completed tasks. It can easily be seen that static teams have a lower throughput than dynamic teams. This is due to the reason that each round the same 502 static teams are asked to perform tasks. At some point the constant number of static workers will reach a high workload and therefore the numbers of bids go down and the number of deadline violations increases. Besides that, deadline violations and therefore bad ratings of tasks trigger the requesters to stop creating new tasks, the reasoning for this is described in detail in [16]. In comparison to the static teams, each round a randomly chosen set of 500 workers can perform in auctions and can spawn dynamic teams. As we already have seen the size of dynamic teams is dependent on the task. This means that the workload is better distributed among all workers when using dynamic teams.

Another detail we can see is that the number of completed tasks is going down drastically if we increase the workload by increasing the number of requesters. Scenarios S1 through D6 show this clearly. The static teams percentage of successfully completed tasks drops from 70% in S1 to 54% in S6; for dynamic teams the percentage drops from 93% in D1 to 84% in D6.

5.3.3 Quality - Ratings

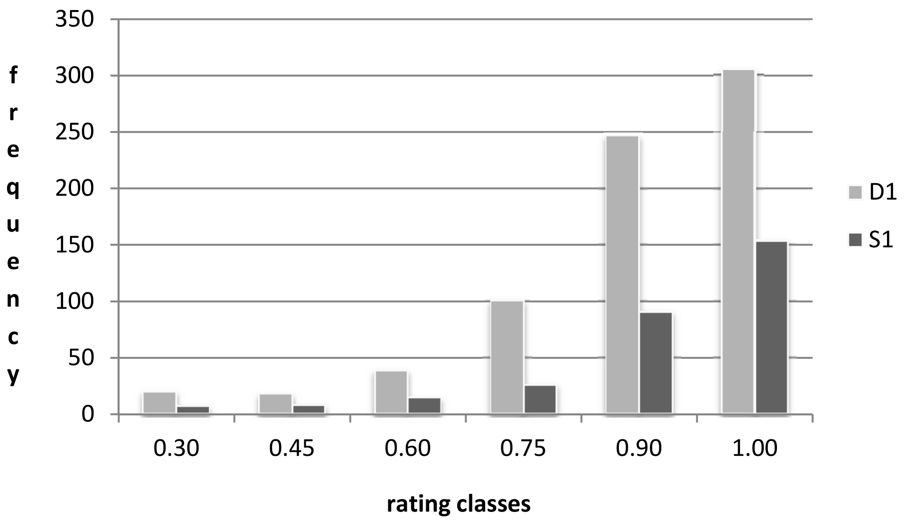

Figure 5.4: Rating histogram - Scenario 1

Teams \ Run	1	2	3	4	5	6
S	0.838	0.871	0.834	0.868	0.830	0.869
D	0.814	0.863	0.830	0.854	0.820	0.856

Table 5.5: Average ratings

In the last subsection, we have focused on throughput, a measurement for the quantity of the solved tasks. This subsection focuses on quality, we can monitor quality with SFC by comparing the task ratings. A high rating close to 1.0 means good quality, a rating below 0.3 stands for very bad quality. As explained before, workers will stop

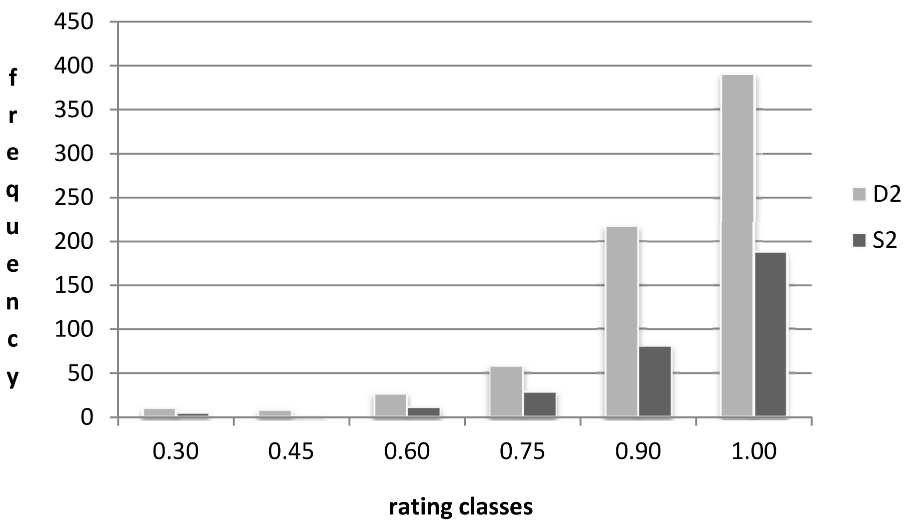

Figure 5.5: Rating histogram - Scenario 2

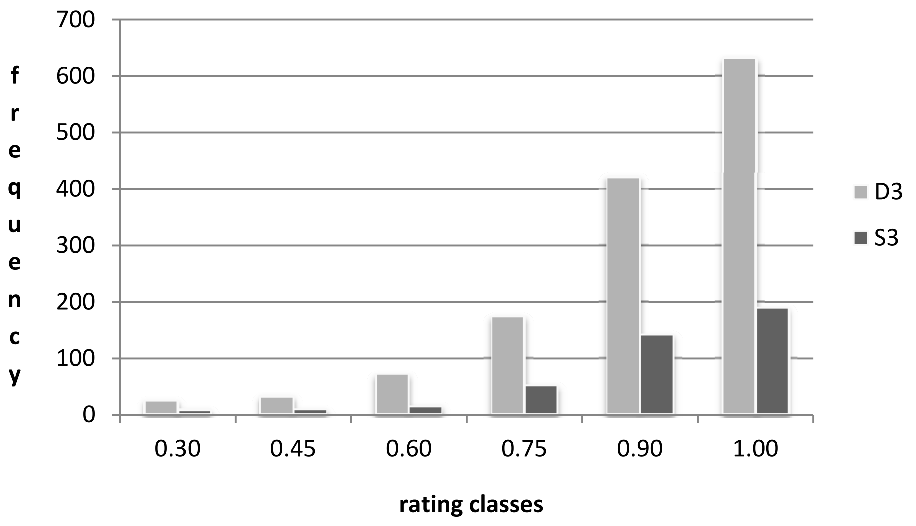

Figure 5.6: Rating histogram - Scenario 3

submitting tasks, if the rating of a finished tasks is under 0.3. Figures 5.4, 5.5, 5.6, 5.7, 5.8 and 5.9 show each a histogram of the ratings of completed tasks. Each diagram compares the ratings for static and dynamic teams for one of the six evaluated scenarios.

Besides that, Table 5.5 shows the average rating for all successfully completed tasks.

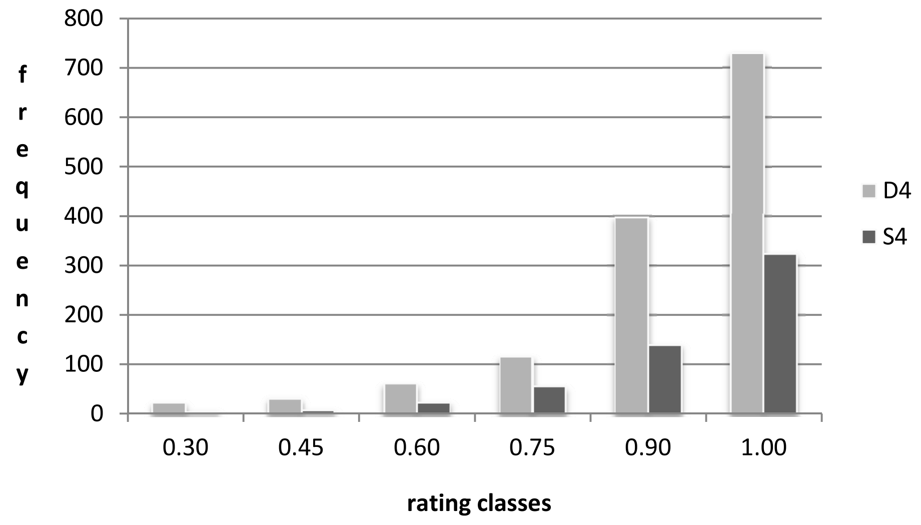

Figure 5.7: Rating histogram - Scenario 4

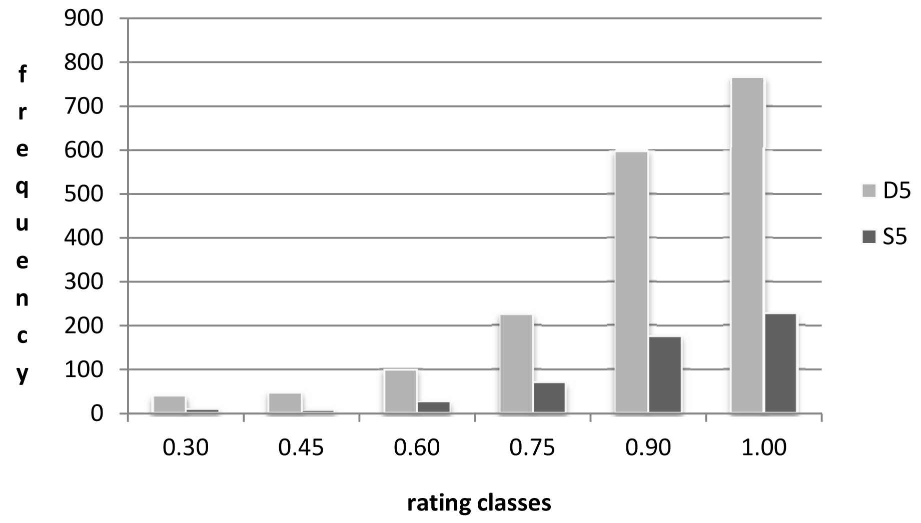

Figure 5.8: Rating histogram - Scenario 5

We can see that although dynamic teams process way more tasks, they still have lower average rating than static teams. That means that static teams process tasks with an average higher quality than dynamic teams. We argue that these results are due to the reason that the same workers in static teams work together all the time. This leads to a higher collaboration factor due to the increased number of attempts to collaborate with

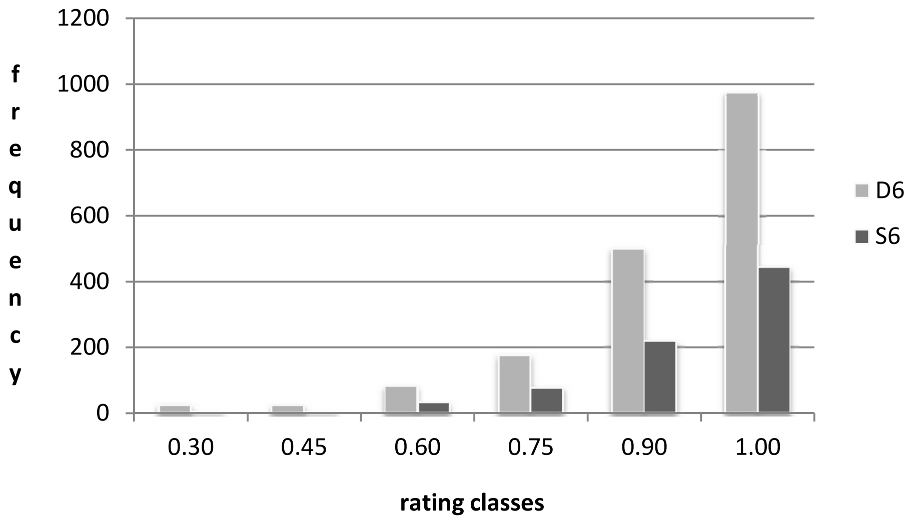

Figure 5.9: Rating histogram - Scenario 6

each other. As explained in chapter 4 the collaboration factor is important in determining the result of a composite task. A higher average collaboration factor within a team leads to a better quality of the processed work and therefore triggers a higher rating of the requester.

5.4 Discussions

In general, performing evaluation with crowdsourcing systems is a challenging task. Many premises have to be made about worker and requester behaviour. The optimal solution to evaluate the work that we present in this thesis, would be access to a real life crowdsourcing systems. However as the approach itself is introducing team structures based on a social collaboration network, this seems very hard to accomplish. Right now no existing crowdsourcing platform, which supports the generic task-based crowdsourcing model presented in chapter 3, is supporting this. The data gathered by our evaluation shows us advantages and limitations of introducing static and dynamic teams to task-based crowdsourcing. The biggest advantage of introducing techniques of collaboration is the possibility to solve complex work. Teams provide the means to solve complex work. Supporting teams seems to be a solution to overcome the problem of the arising demand for the integration of complex work within task-based systems. Companies themselves use teams and divisions to manage complex work, therefore it is a natural solution to bring these techniques of collaboration to task-based crowdsourcing systems. The data gathered by our evaluation shows us the following advantages and limitations of both approaches:

- **Dynamic Teams:** Our implementation of dynamic teams has a higher throughput than static teams. In numbers this means that dynamic teams process minimum 50% more complex tasks than static teams. This is due to the following reasons:
  - Dynamic teams are generated on demand. The team is trimmed to the demands of composite tasks. Each auction 500 randomly chosen workers are asked to place a bid for a composite task. That means that a possible amount of 500 different dynamic teams can be formed. In comparison to static teams this is a huge advantage. All 502 static teams participate in every auction. Therefore the pressure generated by the requesters to solve tasks is higher than for a randomly selected dynamic team. This also means that the workload, generated by the requesters, is better distributed throughout the whole crowd if dynamic teams are used.
  - Workers with a high workload are not participating in a dynamic team. Therefore a dynamic team always consists of workers with a low workload.
  - Both reasons stated before reduce the number of deadline violations, which increases throughput as well.
- **Static Teams:** Static teams have due to the nature of their implementation not as high throughput as dynamic teams. This is based on the fact that static teams cannot change their team size. However static teams process tasks with moderately better results. This is based on the fact, that static teams have a higher average collaboration factor. Besides that integrating static teams in an existing crowdsourcing platform seems to be easier to perform, this is due to the fact that static teams are not dependent on a social network when performing work. Systems,

implementing static teams, have to provide a meaningful way for individuals to register as a team, but there is no need to implement and support a social network. Therefore static teams are an interesting possibility to support complex work, within the boundaries of existing crowdsourcing systems.

CHAPTER 6

Conclusion and Future Work

Modern task-based crowdsourcing systems like Amazon's Mechanical Turk offer a marketplace-based crowdsourcing platforms, where requesters can post tasks which get solved by a crowd of workers. The tasks supported by such platforms are limited in the way, that they only support tasks which can be independently solved by one worker. The arising demand of enterprises around the world to crowdsource more complex work with task-based crowdsourcing platforms, shows the need to extend state-of-the-art crowdsourcing systems in a way, that they can fulfil these demands.

In this thesis we have addressed the problem of integrating complex work into task-based crowdsourcing and have come up with a solution, which is based on the introduction of techniques for collaboration. We introduce two team structures, namely static and dynamic teams, for workers, which make it possible to integrate and process complex work within the context of a crowdsourcing system. Both team approaches are based on a social collaboration network, which shows how well the workers in a the crowd can work together. Static teams exist in contrast to dynamic teams for a long time. The aim of static teams is to solve as many tasks together as possible. In comparison to that, dynamic teams are formed to solve a single task. To integrate complex work to a task-based crowdsourcing, we introduce a composite task. A composite-task has a set of sub-tasks, which can have dependency between each other. The dependencies show how much cooperation is required to solve the sub-tasks. We implement our model of composite tasks and both team structures into an existing simulation framework for crowdsourcing systems. The implementation addresses the challenges of splitting, assigning and processing composite tasks. In the evaluation of our implementation for static and dynamic teams, it became apart that there are differences in the quantity and quality of processed tasks. We show that dynamic teams have a higher performance, but a moderately lower level of quality when solving tasks. Besides that, we argue that the integration of static teams into existing crowdsourcing platforms is easier, as it does not solely rely on a collaboration-based social network, which is mandatory for the implementation of dynamic teams.

6.1 Future work

The current implementation of a model for complex work and techniques for collaboration between workers into a simulation framework can be seen as a first step of introducing complex work to crowdsourcing. Further improvements and changes have to be made to make the integration into an existing crowdsourcing platform, e.g. Amazon's Mechanical Turk, possible.

- Finding and selecting teams is right now based on a greedy algorithm. As explained in chapter 2 different strategies to solve the team formation problem exist. These strategies could be supported to give alternatives to select suitable teams to process a composite task.
- Instead of automatically forming teams, a mechanism could be implemented, to assist workers in creating a dynamic or static team. The suggestions could be based on the team selection strategies discussed in chapter 2. Based on these suggestions a worker could decide then to form a team individually.
- Static and dynamic teams, have been discussed separately. In this thesis they both stand in contest to each other. Future work could include, combining both approaches. This could lead to a generally higher level of quality and throughput for composite tasks.

- Another possibility would be to extend the classical model of crowdsourcing in a way, that workers can also interact with the system as requester. This dual role of workers could make it possible for teams to split composite tasks and to crowdsource parts of a composite task.

- We have evaluated both approaches with a specific simulation framework for task-based crowdsourcing with a randomly generated social collaboration network. A next step could be the evaluation of our approach using a real social network.

APPENDIX A

Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
An example configuration to perform a simulation
with 20 requesters and 2500 workers.

       static teams: enabled
       dynamic teams: disabled

       reports: CSV worker report
 -->

<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
   http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd">

<import resource="reports.xml"/>
<import resource="network.xml"/>

<bean id="marketFacade"
   class="at.ac.tuwien.dsg.crowdsim.market.MarketFacade">
<property name="controller" ref="simulationController"/>
```

```
<property name="market" ref="market"/>
</bean>

<!-- Start class of the framework, simulation and reports are configured here -->
<bean id="simulationController"
  class="net.sourceforge.jabm.SpringSimulationController">
<property name="numSimulations" value="1"/>
<property name="simulationBeanName">
    <idref local="marketSimulation"/>
</property>
<property name="reports">
    <list>
        <ref bean="workerCSVReport"/>
        <ref bean="workerReportVariables"/>
    </list>
</property>
</bean>

<bean id="marketSimulation" scope="prototype"
  class="at.ac.tuwien.dsg.crowdsim.market.MarketSimulation">
<property name="maximumRounds" value="500"/>
<property name="population" ref="population"/>
<property name="agentInitialiser" ref="agentInitialiser"/>
<property name="agentMixer" ref="randomRobinAgentMixer"/>
<property name="simulationController" ref="simulationController"/>
</bean>

<bean id="market"
  class="at.ac.tuwien.dsg.crowdsim.market.Market">
<property name="marketFacade" ref="marketFacade"/>
<property name="auctionFactory" ref="auctionFactoryBean"/>
<property name="skillUpdatePolicy" ref="uniformSkillUpdate"/>
<property name="auctionQualificationPolicy" ref="minPerformanceQualification"/>
</bean>

<!-- auction factory, an auction runs for 1 round -->
<bean id="auctionFactoryBean"
  class="at.ac.tuwien.dsg.crowdsim.market.auction.AuctionFactory">
```

```
<property name="auctionDuration" value="1"/>
<property name="auctionPrototype" ref="sealedBid"/>
<property name="orderBookProtoType" ref="lowestPriceOrderBook"/>
<property name="seed" ref="seed"/>
</bean>

<bean id="lowestPriceOrderBook" scope="prototype"
  class="at.ac.tuwien.dsg.crowdsim
    .market.auction.orderbook.LowestPriceOrderBook"/>

<!-- auction to be used -->
<bean id="sealedBid" scope="prototype"
  class="at.ac.tuwien.dsg.crowdsim
    .market.auction.SealedBidSecondPriceAuction">
<property name="marketFacade" ref="marketFacade"/>
</bean>

<bean id="agentInitialiser"
  class="net.sourceforge.jabm.init.BasicAgentInitialiser">
</bean>

<bean id="randomRobinAgentMixer"
  class="net.sourceforge.jabm.mixing.RandomRobinAgentMixer">
<property name="prng" ref="prng"/>
</bean>

<bean id="population" scope="prototype"
  class="net.sourceforge.jabm.Population">
<property name="agentList">
    <bean class="net.sourceforge.jabm.agent.AgentList">
        <constructor-arg>
            <list>
                <ref bean="requesterAgentList"/>
                <ref bean="workerAgentList"/>
            </list>
        </constructor-arg>
    </bean>
</property>
```

```
<property name="prng" ref="prng"/>
</bean>

<bean id="requesterAgentList"
  class="net.sourceforge.jabm.agent.AgentList">
<constructor-arg ref="requesterList"/>
</bean>

<bean id="requesterList"
  class="org.springframework
    .beans.factory.config.PropertyPathFactoryBean">
<property name="targetObject" ref="requesterListFactory"/>
<property name="propertyPath" value="requesters"/>
</bean>

<!-- Defines the number of requesters -->
<bean id="requesterListFactory"
  class="at.ac.tuwien.dsg.crowdsim.agent.requester.RequesterFactory">
<constructor-arg value="20"/>
<constructor-arg ref="requesterFactory"/>
<constructor-arg ref="compositeTaskFactory"/>
</bean>

<!-- The factory used for manufacturing the patrons -->
<bean id="requesterFactory"
  class="org.springframework
    .beans.factory.config.ObjectFactoryCreatingFactoryBean">
<property name="targetBeanName">
    <idref local="simpleRequester"/>
</property>
</bean>

<!-- The prototype used to manufacture requester agents -->
<bean id="abstractRequester" scope="prototype"
  class="at.ac.tuwien.dsg.crowdsim.agent.requester.SimpleRequester">
<property name="strategy" ref="noStrategy"/>
<property name="marketFacade" ref="marketFacade"/>
<property name="scheduler" ref="simulationController"/>
```

```
<property name="priceModel" ref="qualityEffortVP"/>
<property name="supplyPolicy" ref="simpleTaskSupplyPolicy"/>
<property name="ratingPolicy" ref="simpleRatingPolicy"/>
</bean>

<!-- The prototype used to manufacture requester agents -->
<bean id="simpleRequester"
  scope="prototype" parent="abstractRequester"
  class="at.ac.tuwien.dsg.crowdsim.agent.requester.SimpleRequester">
<property name="tradingStrategy" ref="truthTellingTS"/>
</bean>

<!-- Trading strategy - truth telling -->
<bean id="truthTellingTS" scope="prototype" parent="strategy"
  class="at.ac.tuwien.dsg.crowdsim
    .agent.strategy.trading.requester.TruthTellingRequesterTS">
<property name="randomData" ref="randomData"/>
</bean>

<!-- Valuation policy - quality and effort awar policy -->
<bean id="qualityEffortVP"
  class="at.ac.tuwien.dsg.crowdsim.agent.pricing.PriceQualityEffort">
<property name="randomData" ref="randomData"/>
</bean>

<bean id="simpleTaskSupplyPolicy" scope="prototype"
  class="at.ac.tuwien.dsg.crowdsim.agent.policy.SimpleTaskSupply">
</bean>

<bean id="workerAgentList"
  class="net.sourceforge.jabm.agent.AgentList">
<constructor-arg ref="workerList"/>
</bean>

<bean id="workerList"
  class="org.springframework
    .beans.factory.config.PropertyPathFactoryBean">
<property name="targetObject" ref="networkFactory"/>
```

```
<property name="propertyPath" value="workers"/>
</bean>

<!-- Worker factory -->
<bean id="workerAgentFactory"
  class="org.springframework.beans
       .factory.config.ObjectFactoryCreatingFactoryBean">
<property name="targetBeanName">
   <idref local="workerAgent"/>
</property>
</bean>

<!-- The prototype used to manufacture networked workers -->
<bean id="workerAgent" scope="prototype"
  class="at.ac.tuwien.dsg.crowdsim.agent.worker.NetworkedWorker">
<property name="strategy" ref="noStrategy"/>
<property name="marketFacade" ref="marketFacade"/>
<property name="scheduler" ref="simulationController"/>
<property name="valuationPolicy" ref="greedySkillAwareVP"/>
<property name="randomData" ref="randomData"/>
<property name="tradingStrategy" ref="truthTellingStrategy"/>
<property name="skillConfig" ref="experiencedWorkerConfig"/>
<property name="transactionBook" ref="simpleTransactionBook"/>
</bean>

<bean id="simpleTransactionBook" scope="prototype"
  class="at.ac.tuwien.dsg.crowdsim
    .agent.worker.transaction.SimpleTransactionBook">
<property name="randomData" ref="randomData"/>
</bean>

<bean id="teamWorkerAgentFactory"
  class="org.springframework
    .beans.factory.config.ObjectFactoryCreatingFactoryBean">
<property name="targetBeanName">
   <idref local="teamWorkerAgent"/>
</property>
</bean>
```

```
<!-- The prototype used to manufacture static team workers -->
<bean id="teamWorkerAgent" scope="prototype"
  class="at.ac.tuwien.dsg.crowdsim.agent.worker.TeamWorker">
<constructor-arg ref="randomData"/>
<property name="strategy" ref="noStrategy"/>
<property name="marketFacade" ref="marketFacade"/>
<property name="scheduler" ref="simulationController"/>
<property name="valuationPolicy" ref="greedySkillAwareVP"/>
<property name="tradingStrategy" ref="truthTellingStrategy"/>
</bean>

<!-- Network factory -->
<bean id="networkFactory" scope="singleton"
  class="at.ac.tuwien.dsg.crowdsim
    .agent.worker.network.factory.NetworkFactory">
<constructor-arg ref="networkGenerator"/>
<constructor-arg ref="teamWorkerAgentFactory"/>
<constructor-arg ref="seed"/>
<constructor-arg value="true"/>
<constructor-arg value="false"/>
</bean>

<!-- JUNG random network generator -->
<bean id="networkGenerator" scope="singleton"
  class="edu.uci.ics.jung.algorithms
    .generators.random.KleinbergSmallWorldGenerator">
<constructor-arg ref="graphFactory"/>
<constructor-arg ref="vertexFactory"/>
<constructor-arg ref="edgeFactory"/>
<!-- Worker population n * n -->
<constructor-arg value="50"/>
<constructor-arg value="0.55"/>
<property name="randomSeed" ref="seed"/>
</bean>

<bean id="graphFactory" scope="singleton"
  class="at.ac.tuwien.dsg.crowdsim
```

```
      .agent.worker.network.factory.GraphFactory"/>

<bean id="vertexFactory" scope="singleton"
   class="at.ac.tuwien.dsg.crowdsim
      .agent.worker.network.factory.VertexFactory">
<constructor-arg ref="workerAgentFactory"/>
</bean>

<bean id="edgeFactory" scope="singleton"
   class="at.ac.tuwien.dsg.crowdsim
      .agent.worker.network.factory.EdgeFactory">
<constructor-arg ref="randomData"/>
</bean>

<!-- Worker skill configuration -->
<bean id="experiencedWorkerConfig"
   class="at.ac.tuwien.dsg.crowdsim.agent.worker.SkillConfig">
<property name="meanPerformance" value="0.7"/>
<property name="deviancePerformance" value="0.25"/>
<property name="meanConfidence" value="0.8"/>
<property name="devianceConfidence" value="0.25"/>
</bean>

<!-- Worker trading strategy -->
<bean id="truthTellingStrategy"
   class="at.ac.tuwien.dsg.crowdsim
      .agent.strategy.trading.worker.TruthTellingStrategy"
   parent="strategy">
<property name="nrOfMaxOpenBids" value="100"/>
<property name="deadlineMultiplier" value="1.8"/>
</bean>

<bean id="simpleRatingPolicy"
   class="at.ac.tuwien.dsg.crowdsim
      .agent.policy.TeamRatingPolicy"/>

<bean id="allQualificationPolicy"
   class="at.ac.tuwien.dsg.crowdsim
```

```
        .agent.policy.AllQualifyAuctionQualificationPolicy"/>

<!-- Worker valuation policy -->
<bean id="greedySkillAwareVP"
    class="at.ac.tuwien.dsg.crowdsim
        .agent.policy.SkillEffortAwareValuationPolicy"
    scope="prototype">
<property name="randomData" ref="randomData"/>
</bean>

<bean id="noStrategy" scope="prototype"
    class="at.ac.tuwien.dsg.crowdsim.agent.strategy.NoStrategy">
<property name="scheduler" ref="simulationController"/>
</bean>

<bean id="minPerformanceQualification"
    class="at.ac.tuwien.dsg.crowdsim
        .agent.policy.MinPerformanceQualificationPolicy"/>

<bean id="uniformSkillUpdate"
    class="at.ac.tuwien.dsg.crowdsim
        .agent.policy.UniformSkillUpdatePolicy">
<property name="OBS_PERF_UPDATE_RATE" value="0.2"/>
<property name="CONF_UPDATE_RATE" value="0.1"/>
</bean>

<!-- Factory for composite tasks -->
<bean id="compositeTaskFactory"
    class="at.ac.tuwien.dsg.crowdsim
        .agent.task.factory.CompositeTaskFactory">
<property name="edgeFactory" ref="edgeFactory"/>
<property name="taskFactory" ref="simpleTaskFactory"/>
<property name="randomData" ref="randomData"/>
<property name="maxNumberOfTasks" value="6"/>
<property name="minNumberOfTasks" value="3"/>
</bean>

<bean id="simpleTaskFactory"
```

```
  class="at.ac.tuwien.dsg.crowdsim
     .agent.task.factory.SimpleTaskFactory">
<property name="randomData" ref="randomData"/>
</bean>

<!-- Pseudo random number generators for the simulation-->
<bean id="prng" scope="singleton"
  class="cern.jet.random.engine.MersenneTwister64">
<constructor-arg ref="seed"/>
</bean>

<bean id="randomData" scope="singleton"
  class="org.apache.commons.math.random.RandomDataImpl">
<constructor-arg ref="randomGenerator"/>
</bean>

<bean id="randomGenerator" scope="prototype"
  class="org.apache.commons.math.random.MersenneTwister">
<constructor-arg ref="seed"/>
</bean>

<bean id="seed" scope="singleton" class="java.lang.Long">
<constructor-arg value="123"/>
</bean>

<bean id="strategy" scope="prototype" abstract="true"
  class="net.sourceforge.jabm.strategy.AbstractStrategy">
<property name="scheduler" ref="simulationController"/>
</bean>

<!-- Reporting to collect evaluation data-->
<bean id="workerCSVReport" scope="singleton"
  class="at.ac.tuwien.dsg.crowdsim.report.CSVMasterReport">
<property name="reportVariables">
     <bean class="at.ac.tuwien.dsg.crowdsim
        .report.taskReportVariables.CSVReportVariables">
          <property name="numColumns" value="31"/>
          <property name="fileNamePrefix" value="data/worker"/>
```

```
            <property name="reportVariables" ref="workerReportVariables"/>
        </bean>
</property>
</bean>

<bean id="workerReportVariables"
   class="at.ac.tuwien.dsg.crowdsim
      .report.taskReportVariables.WorkerReportVariable">
<constructor-arg ref="marketFacade"/>
</bean>
</beans>
```

Listing A.1: Example configuration with static teams

Bibliography

[1] Inc. Alexa Internet. http://www.alexa.com/topsites. Accessed: 2012-04-10.

[2] Nikolay Archak. Optimal design of crowdsourcing contests. *ICIS 2009 Proceedings*, 2009.

[3] Daren C. Brabham. Crowdsourcing as a Model for Problem Solving. *Convergence: The International Journal of Research into New Media Technologies*, 14(1):75–90, February 2008.

[4] Alberto Caprara, Paolo Toth, and Matteo Fischetti. Algorithms for the set covering problem. *Annals of Operations Research*, 98:353–371, 2000. 10.1023/A:1019225027893.

[5] W Van der Aalst and Kees Van Hee. Workflow Management: Models, Methods, and Systems. In *Workflow Management: Models, Methods, and Systems (Cooperative Information Systems)*, chapter 5 Function, pages 145 – 210. The MIT Press, paperback edition, 2004.

[6] Dominic DiPalantino and Milan Vojnovic. Crowdsourcing and all-pay auctions. In *Proceedings of the tenth ACM conference on Electronic commerce - EC '09*, page 119, New York, New York, USA, 2009. ACM Press.

[7] Christoph Dorn, Florian Skopik, Daniel Schall, and Schahram Dustdar. Interaction mining and skill-dependent recommendations for multi-objective team composition. *Data & Knowledge Engineering*, 70(10):866–891, October 2011.

[8] Schahram Dustdar and Kamal Bhattacharya. The Social Compute Unit. *IEEE Internet Computing*, 15(3):64–69, May 2011.

[9] Schahram Dustdar and Martin Gaedke. The Social Routing Principle. *IEEE Internet Computing*, 15(4):80–83, July 2011.

[10] Enrique Estelles-Arolas and Fernando Gonzalez-Ladron-deGuevara. Towards an integrated crowdsourcing definition. *Journal of Information Science*, (X):1–14, 2012.

[11] Wikimedia Foundation. http://en.wikipedia.org. Accessed: 2012-04-10.

[12] JABM Framework. http://sourceforge.net/apps/phpwebsite/jabm/index.php. Accessed: 2012-04-25.

[13] Java Universal Network/Graph Framework. http://jung.sourceforge.net/. Accessed: 2012-04-12.

[14] M Girvan and M E J Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[15] M Granovetter. The strength of weak ties: A network theory revisited. *Sociological theory*, 1:201–233, 1983.

[16] Tobias Hammerer. A Simulation Framework for Task-based Crowdsourcing with Auctions. Master's thesis, Vienna University of Technology, 2012.

[17] H He. What is service-oriented architecture. pages 1–5, 2003.

[18] Heather Hobden and Mervyn Hobden. *John Harrison and the Problem of Longitude*. The Cosmic Elk, paperback edition, 1988.

[19] Jeff Howe. The rise of crowdsourcing. *Wired magazine*, (14), 2006.

[20] Amazon Inc. https://www.mturk.com. Accessed: 2012-04-10.

[21] Innocentive Inc. http://www.innocentive.com/. Accessed: 2012-04-28.

[22] Stack Exchange Inc. http://stackoverflow.com/. Accessed: 2012-04-28.

[23] Yahoo Inc. http://answers.yahoo.com/. Accessed: 2012-04-28.

[24] IStockphoto.com. http://www.istockphoto.com. Accessed: 2012-04-10.

[25] Aniket Kittur, Boris Smus, and Robert Kraut. CrowdForge: crowdsourcing complex work. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, CHI EA '11, pages 1801–1806, New York, NY, USA, 2011. ACM.

[26] Jon Kleinberg. Navigation in a small world. *Nature*, 406(6798):845, August 2000.

[27] AP Kulkarni and Matthew Can. Turkomatic: automatic recursive task and workflow design for mechanical turk. *Proceedings of the 2011 annual*, 2011.

[28] Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, page 467, New York, New York, USA, 2009. ACM Press.

[29] Menghui Li, Ying Fan, Jiawei Chen, Liang Gao, Zengru Di, and Jinshan Wu. Weighted networks of scientific communication: the measurement and topological role of weight. *Physica A: Statistical Mechanics and its Applications*, 350(2-4):643–656, May 2005.

[30] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. TurKit: tools for iterative tasks on mechanical Turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation - HCOMP '09*, page 29, New York, New York, USA, 2009. ACM Press.

[31] Stanley Milgram. The Small World Problem. *Psychology Today*, 2:60–67, 1967.

[32] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement - IMC '07*, page 29, New York, New York, USA, 2007. ACM Press.

[33] M E Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences of the United States of America*, 98(2):404–9, January 2001.

[34] Benjamin Satzger, Harald Psaier, Daniel Schall, and S Dustdar. Stimulating Skill Evolution in Market-Based Crowdsourcing. In *9th International Conference on Business Process Management (BPM)*, volume 6896 of *Lecture Notes in Computer Science*, pages 66–82. Springer, 2011.

[35] Guido Sautter and Klemens Böhm. High-throughput crowdsourcing mechanisms for complex tasks. In Anwitaman Datta, Stuart Shulman, Baihua Zheng, Shou-De Lin, Aixin Sun, and Ee-Peng Lim, editors, *Social Informatics*, volume 6984 of *Lecture Notes in Computer Science*, pages 240–254. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-24704-0_27.

[36] Springsource. http://www.springsource.org/. Accessed: 2012-04-10.

[37] Opera House Sydney. http://www.sydneyoperahouse.com/about/house_history_landing.aspx. Accessed: 2012-04-10.

[38] Taskcn. http://www.taskcn.com/. Accessed: 2012-04-28.

[39] Arthur ter Hofstede, Wil van der Aalst, Arthur ter Hofstede, and Mathias Weske. Business process management: A survey. In Mathias Weske, editor, *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 1019–1019. Springer Berlin / Heidelberg, 2003. 10.1007/3-540-44895-0_1.

[40] The Object Management Group UML. http://www.uml.org/. Accessed: 2012-04-13.

[41] M Vukovic. Crowdsourcing for Enterprises. *Services, IEEE Congress on*, 0:686–692, 2009.

[42] Fei-Yue Wang, Kathleen M. Carley, Daniel Zeng, and Wenji Mao. Social Computing: From Social Informatics to Social Intelligence. *IEEE Intelligent Systems*, 22(2):79–83, March 2007.

[43] D J Watts and S H Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[44] Doug Wightman. Crowdsourcing human-based computation. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction Extending Boundaries - NordiCHI '10*, page 551, New York, New York, USA, 2010. ACM, ACM Press.

[45] Erik Wilde and Martin Gaedke. Web Engineering Revisited. In *Visions of Computer Science - BCS Int'l Academic Conf.*, pages 41–49. British Computer Society, 2008.

[46] Simon Winchester. *The Surgeon of Crowthorne: A Tale of Murder, Madness and the Oxford English Dictionary*. Penguin, paperback edition, 1999.

[47] Jiang Yang, Lada A Adamic, and Mark S Ackerman. Crowdsourcing and knowledge sharing. In *Proceedings of the 9th ACM conference on Electronic commerce - EC '08*, page 246, New York, New York, USA, 2008. ACM Press.