Die approbierte Originalversion dieser Dissertation ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Collaboration-assisted computation

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Vitaliy Liptchinsky
Registration Number 1027328

to the Faculty of Informatics
at the TU Wien

Advisor: Univ.-Prof. Schahram Dustdar

The dissertation has been reviewed by:

Javid Taheri

Massimo Villari

Vienna, 8th October, 2016

Vitaliy Liptchinsky

Erklärung zur Verfassung der Arbeit

Vitaliy Liptchinsky
15379 NE 99th Way, Redmond, WA, USA

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.

Wien, 8. Oktober 2016

Vitaliy Liptchinsky

Acknowledgements

First of all, I am endlessly grateful to Prof. Schahram Dustdar for accepting me as a Ph.D. student, his guidance in choosing a research topic, and supporting me throughout all the phases of my research. Without his mentoring, this work would have never seen the world. Also, I would like to express my gratitude to Assoc. Prof. Hong-Linh Truong for his support at the early stages of my research. Additionally, I am very grateful to Dr. Trishul Chilimbi, my supervisor at Microsoft Research, for supporting my Ph.D. research at its final stages.

Furthermore, I would like to thank all my colleagues from the Distributed Systems Group for inspiration and interesting discussions: Florian Skopik, Lukasz Juszczyk, Daniel Schall, Christian Inzinger, Martin Treiber, Fei Li, Ognjen Scekic, Mirela Riveni, and Shaghayegh Sharif Nabavi. But especially, I am infinitely grateful to my "partners in crime" Roman Khazankin, Benjamin Satzger, Assoc. Prof. Stefan Schulte, and Rostyslav Zabolotnyi. I can't imagine how any of this thesis' research contributions could have been carried out without their innovative ideas, constructive criticism, and enormous help.

Most importantly, I am very grateful to my family. I would like to thank my precious wife Nataliya, and my beloved children Denys and Daniel for their patience, understanding, and love. I would like to thank my dear parents Lidiya and Valentyn for motivating me to enroll to Ph.D. studies and supporting me through them.

Abstract

A whole is greater than the sum of its parts. A collaborating team is greater than a group of contributors working in isolation. In this thesis we introduce a novel technique called *collaboration-assisted computation* that evolves *human-assisted computation* in line with these postulates. As human computation focuses on integrating human input at various phases of machine computation, so collaboration-assisted computation aims at integrating machine computation with input from collaborating teams. However, collaboration-assisted computation is something more than a simple replacement of the term *human input* with the term *team input* in the pipeline of machine computation. What is collaboration without social interaction? How effective can collaboration be without convenient software tools? While the answers to these questions lie outside of the scope of this thesis, we argue that a truly efficient collaboration orbits around social context and collaborative software. Therefore, the center of gravity for collaboration-assisted computation lies at the intersection of human computation, social computing and collaborative software. Moreover, collaboration-assisted computing relies on crowdsourcing to execute collaboration at massive scale.

Hence, this thesis presents a holistic framework for modeling and programming collaboration-assisted computation. First, we present a query language capable to express intuitively complex social traits of collaborating groups. Second, we show how to model social collaboration processes. Third, the thesis introduces a programming language to coordinate collaborative teams and a framework for integration of social and collaborative software. Fourth, we show how crowdsourcing models can be extended to scale collaboration processes. The proposed modeling and programming languages were evaluated with extensive use cases, showing intuitiveness and expressiveness of each of the approaches.

Contents

List of Figures

List of Tables

List of Listings

Earlier Publications

This thesis is based on the previous work published in scientific conferences, and journals. For reasons of brevity, these core papers, which build the foundation of this thesis, are listed here once, and will generally not be explicitly referenced again. Parts of these papers are contained in verbatim.

- Vitaliy Liptchinsky, Benjamin Satzger, Rostyslav Zabolotnyi, and Schahram Dustdar "Expressive languages for selecting groups from graph-structured data.", *22nd international conference on World Wide Web (WWW)*, pp. 761-770, ACM, 2013.
- Vitaliy Liptchinsky, Roman Khazankin, Hong-Linh Truong, and Schahram Dustdar "A novel approach to modeling context-aware and social collaboration processes.", *International Conference on Advanced Information Systems Engineering (CAiSE)*, pp. 565-580, 2012. **Best paper award.**
- Vitaliy Liptchinsky, Roman Khazankin, Stefan Schulte, Benjamin Satzger, Hong-Linh Truong, and Schahram Dustdar "On modeling context-aware social collaboration processes", *Information Systems*, pp. 66–82, vol. 43, 2014.
- Vitaliy Liptchinsky, Roman Khazankin, Hong-Linh Truong, and Schahram Dustdar "Statelets: coordination of social collaboration processes.", *International Conference on Coordination Languages and Models (COORDINATION)*, pp. 1-16, 2012.
- Vitaliy Liptchinsky, Benjamin Satzger, Stefan Schulte, and Schahram Dustdar "Crowdstore: A Crowdsourcing Graph Database.", *International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pp. 72-81, 2015.

CHAPTER 1

Introduction

Many problems that appear easy to humans cannot be solved by computers, such as natural language understanding, decision-making, inference and so on. Therefore, incorporation of human elements in machine computation has been extensively considered since more than half a century. Starting from the fitness function in genetic algorithms [Kos01], adoption of human-assisted (human-based) computation gradually evolved from mere language and framework extensions [KKL+05, AAD+07, STD11], Turing Machine extensions [SA07] to fully human-centered programming models, such as crowdsourcing-based programming models. To date, human computation has been considered aligned to machine computation, i.e. human components were connected exclusively by dataflow bindings. In other words, "human" components were integrated as isolated components with predefined inputs and outputs. While such representation perfectly reflects communication between libraries, threads, processes and remote services in classic computer software design, it ignores social connections between human workers and influence of social formations on responsiveness and quality of the work performed. For example, a socially coherent team is more likely to produce a better result than a team of strangers.

While human-assisted computation promotes a computer-centered view, i.e. humans assist computer programs, social computing promotes quite the opposite view of computer software facilitating social interaction [Sch94] and social network analysis [WCZM07]. Social network analysis operates with plethora of patterns and characteristics that define social coherence of a team or connectedness between teams, such as geodesics, n-clique, n-clan, n-club, k-core, k-plex, closeness centrality, betweenness centrality and so on [Sco00].

While social computing focuses on facilitating social interaction, collaborative software (groupware) focuses on improving collaboration. Collaboration capabilities of groupware systems range from mere communication tools, real-time screen or file sharing, conferencing systems, to complex coordination platforms, such as project management and workflow systems.

Crowdsourcing allows outsourcing and distributing simple tasks to a large group (crowd) of anonymous human workers via specialized software platforms. The terms *crowdsourcing* and *human computation* are often intermixed and used synonymously in the literature. Crowdsourcing work, however, can produce results that cannot be understood and comprehended by computers, such as images, or text passages, contradicting thus the main focus of human computation.

Even though there is a significant overlap between human computation, social computing, collaborative software and crowdsourcing, the centers of their gravity are different. This thesis introduces the technique called *collaboration-assisted computation*, which

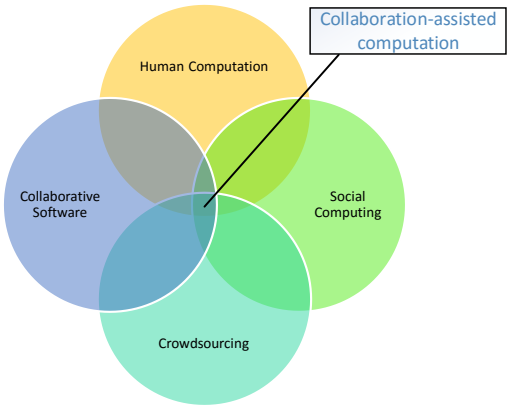


Figure 1.1: Human-centered computing paradigms

evolves human-based computation by outsourcing computation to complex social formations (groups) of human workers, rather than isolated workers, with the focus on collaboration as the driving force to produce the desired outputs for different phases of machine computation. Interest in enhancing human computation with social traits is gaining momentum in academia [DB11], emphasizing thus the necessity to bring these social patterns and characteristics into the domain of human-based computation, and further bridging the gap between human-assisted computation and social computing. Venn digram in Figure 1.1 depicts the focus of collaboration-assisted computation as intersection of the areas of interest of crowdsourcing, human computaton, social computing and collaborative software. Collaboration-assisted computation extends human computation by considering social relations as the driving factor for successful computation results. In the meantime, collaboration is hard to carry out without collaborative software and scale out without the methods of crowdsourcing. Similarly to the case of human-based computation, the terms collaboration-assisted computation and collaboration-based computation can be used interchangeably.

1.1 Problem Statement

Enhancing human-assisted computation with social collaboration requires: (i) a formalism capable of expressing patterns and characteristics widely used in social network analysis; (ii) a tool set of programming models, languages, and frameworks to support collaboration-based computation throughout all the phases of software design and engineering.

Social formations can be intuitively represented as graphs, and graph formalisms and graph query languages are widely adopted for querying social networks and capturing various graph patterns. This makes graph query languages [AG08, Woo12a] a natural choice for expressing groups of people. However, expressing complex social formations, such as variable-sized cliques and k-plexes, hubs, structural equivalence [Sco00] etc., is extremely difficult and verbose in the existing graph query languages. The problem here is how to extend the existing graph query languages in order to express intuitively groups of people, and consequently, advanced patterns in social networks.

A typical life cycle of software design and engineering undergoes through three main phases: modeling, implementation, and scaling. The first phase of software engineering is business process modeling (BPM), during which not only software engineers are involved, but also process analysts and stakeholders. Business process modeling allows companies to describe and document their enterprise processes. Although a variety of techniques and tools have been introduced in this area, modeling of highly dynamic non-routine processes such as human collaboration is still a subject for discussion in research [Nur08]. Chaotic nature of creative human collaboration makes it difficult to apply imperative modeling approaches, predominantly used in business process modeling, as it is hard to predefine exact steps to follow [Nur08].

The second phase of software engineering is implementation, which can be argued to be the key phase. Implementation consists of a data model the program operates on, and coordination model between program parts. The data model in our case is defined by social relations. Utilization of social relations information can only be promoted by integration of various social networking platforms, as those possess comprehensive user data sets and the amount of such data will grow with social computing pervading the enterprise IT[1]. In addition, the coordination of people groups requires integration of groupware platforms and products serving as convenient tools for human collaboration. The problem here is how to seamlessly integrate social and groupware platforms and enable coordination of human collaboration in a unified way. Such integration brings in multiple challenges such as how to manage authentication and authorization, and map entities from different platforms.

The third phase of software engineering is scaling the implementation. Programming for scale often means redesigning the implementation towards cloud computing platforms. Being able to provide dozens of thousands of computing resources, cloud computing can be considered as the endgame for scalability projects. With respect to human computation crowdsourcing platforms provide resources to execute at a large scale comparable to cloud computing. Crowdsourcing today is focused on simple unskilled crowd work, represented

[1] http://www.gartner.com/it/page.jsp?id=1470115

as Human Intelligence Tasks (HITs)[2], which are posted online and are expected to be picked up by workers. It is not possible to express group work with such "pull" model. Enabling programming of group work along with social aspects can extend focus of crowdsourcing from simple and easy tasks to complex collaborative work. The problem here is how to adjust crowdsourcing programming models to express collaborative group work.

1.2 Research Questions

We postulate that in order to enable collaboration-assisted computation and address the aforementioned problems it is necessary to answer the following research questions:

Question 1: *How to express complex social formations with relaxed structure in a formal and structured way?*

A formalism for collaboration-based computation needs to operate with groups of people (teams) and individual contributors (army of one) interchangeably, which calls for a notion of group as a first-class construct of the formalism. This would enable expression of advanced composite structures, such as a group comprising of multiple groups. Structure of a group can be represented both by social formations and dataflow dependencies. Graph query languages have come a long way with respect to succinct expression of complex graph patterns, from basic regular path expressions representing repetition of edges to graph motifs [HS08] representing repetition of cycles and trees. Yet the concept of groups has not been employed as a first-class citizen, limiting thus expressivity of complex social formations.

With respect to relaxed graph structures, research in graph query languages focused on two opposite approaches: "flexible" topologies [FLM+11] and approximate graph matching [KS01, GT06, HPW09]. While the former approach relaxes constraints on topology of returned graphs, the latter one allows result approximations with edit distances. The question here is how to design a language that would allow for flexibility in graph structure while avoiding producing completely irrelevant results.

Question 2: *How to model collaboration-assisted computation?*

This question can be in-depth re-phrased as "how to model chaotic social collaboration while preserving its freedom, yet capturing key aspects of collaboration processes?". A modeling notation for social collaboration has to address three main requirements. First, it has to rely on declarative programming paradigm, describing a process in terms of "what" needs to be done, and not "how" to do it. Such formulation is inherent to human collaboration, leaving the necessary freedom of execution to the people involved. Second, the modeling notation has to find a proper balance between intuitiveness and expressivity when it comes to expressing complex social formations with relaxed structure, such as k-plex, liaison, independent teams, etc. Third, the modeling notation has to intuitively blend representation of groups of people and social formations into workflow sequences.

[2] https://www.mturk.com/mturk/welcome

The modeling notation also has to provide a visual formalism that adheres to the discussed requirements. The visual formalism has to focus on extending an existing visual language, rather than introducing a new one, in order to foster its adoption.

Question 3: *How to simplify implementation of collaboration-assisted computation?*

Implementation of collaboration-assisted computation requires an extensive programming framework that simplifies integration of plethora of existing social networking platforms and groupware services. Social networking sites, such as Facebook[3], LinkedIn[4], etc., provide information on social and semantic relations, while groupware services, such as Assembla[5], VersionOne[6], etc., provide an ability to assign work to people and react to work progress. Seamless integration of data and groupware services poses many different challenges, as social computing software exposes heterogeneous application programming interfaces (APIs), which are often asymmetric, have no clear indicators of idempotence or presence of side effects. However, the main challenge comes from coordination of process flow throughout many distinct social computing and groupware platforms, as it often requires complex synchronization patterns. The architecture of an integration and coordination framework should enable entity mapping, seamless transient authentication and authorization, and plug-ability of different APIs into a single programming language.

Question 4: *How to scale collaboration-assisted computation?*

Scaling out collaboration-assisted computation means enabling collaborative work involving hundreds or even thousands of people groups. Crowdsourcing platforms, such as Amazon Mechanical Turk[7], enable massive scalability of human computation. These platforms, however, ignore altogether collaboration aspects of human computation that encompass social relations and intelligent worker (group) discovery. To support collaboration-assisted computation crowdsourcing frameworks and languages need to conform to the "push" model which assigns workers to tasks rather than widely employed "pull" model that allows workers to pick up tasks. Moreover, crowdsourcing languages and programming models should not only support social selection of groups of people, but also allow crowd workers to recruit group members in order to foster collaboration. Scaling computation often goes hand in hand with performance tuning and optimization of a program to minimize response times. In case of collaboration-assisted computation real-time collaborative work can be achieved via real-time crowdsourcing [BBMK11, BKMB12] with a premise to produce results within seconds after initial request.

1.3 Scientific Contributions

In an attempt to solve the research questions formulated in Section 1.2, this thesis focuses on building a novel and holistic framework aiming to enable collaboration-based

[3]http://www.facebook.com
[4]http://www.linkedin.com
[5]https://www.assembla.com/
[6]https://www.versionone.com/
[7]https://www.mturk.com/mturk/welcome

computation. The framework spans the key phases of the software engineering life cycle: modeling, implementation, and scaling. Figure 1.2 depicts the core components of the framework denoted with the contributions they correspond to.

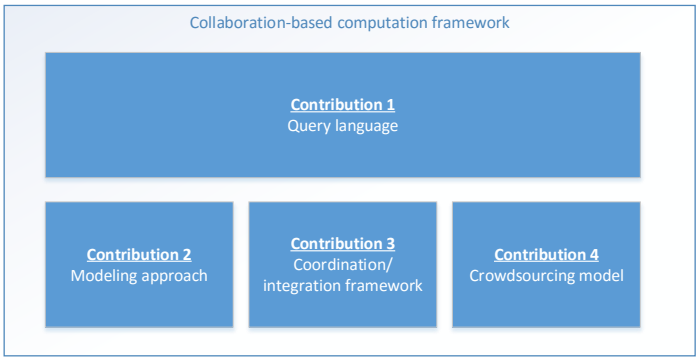


Figure 1.2: Collaboration-based computation framework

Below we briefly overview the major contributions of the thesis along with relations between the contributions. The contributions aim to push forward the state of the art in programming models, languages, and frameworks to support, promote and simplify programming of collaboration-based computation. The contributions aim at extending existing programming or modeling languages rather than inventing new languages from scratch. Apart of the benefit of relying on well-established foundations, such approach also exercises the potential benefit of increased adoption in the research community and industry due to familiarity of programmers with the existing languages.

- **Contribution 1: A query language for expressing complex social formations.**

  We define Conjunctive Set Regular Path Queries (CSRPQs), which allow for intuitive and succinct expression of complex social formations. CSRPQs extend Conjunctive Regular Path Queries (CRPQs) which form the basis for many existing graph query languages, allowing thus easy integration of CSRPQs. CSRPQs introduce a notion of group, allowing thus expression of a variety of patterns in social network analysis and selection of subgraphs with flexible structure. In addition, we define efficient CSRPQ query evaluation techniques for selecting groups from graph structured data, and show that CSRPQ-based query evaluation outperforms CRPQ-based baseline.

  Details on CSRPQs along with examples demonstrating their expressivity are presented in Chapter 3. Contribution 1 has originally been presented in [LSZD13].

- **Contribution 2: A modeling approach and a visual notation to model social collaboration processes.**

We introduce a modeling approach that allows to capture chaotic social collaboration processes by expressing collaboration processes in a bottom-up fashion similar to Cellular Automata [Neu66]. Such representation allows to capture key stages of collaboration processes while preserving freedom of collaboration. The accompanying modeling formalism integrates CSRPQs, which allows to express collaboration within and between social groups. In addition, we define visual notation that allows for graphical representation of modeled processes. The visual notation is based on Statecharts, smoothing thus the learning curve for people familiar with UML[8].

Details on the modeling approach are discussed in Chapter 4. Contribution 2 has originally been presented in [LKTD12a] and further extended in [LKS+14].

- **Contribution 3: A programming language and a framework to coordinate social collaboration.**

  We define Statelets, a coordination language accompanied by an integration framework. Statelets simplifies programming of complex synchronization patterns inherent to collaborative processes. The Statelets design is heavily based on our modeling approach, employing the same principles inspired by Cellular Automata. In addition, Statelets allows for seamless integration of heterogenous social neworking and groupware APIs. Statelets is the only contribution of this thesis that defines a programming language from scratch rather than extending one of the existing approaches.

  Details on Statelets are discussed in Chapter 5. Contribution 3 has originally been presented in [LKTD12b].

- **Contribution 4: A crowdsourcing graph database to augment crowdsourcing processes with social and collaborative aspects.**

  In this contribution we introduce Crowdstore, a crowdsourcing graph database. Crowdstore aims at extending the popular crowd-as-a-database model to incorporate intelligent worker discovery, real-time crowdsourcing, and social formations in order to enable programming of low-latency collaboration-enabled computation at scale. Crowdstore includes yet another CRPQ extension, CRowdPQ, which can be combined with CSRPQ to further enhance expressivity of social formations in crowdsourcing queries. In addition, Crowdstore introduces efficient query evaluation techniques and database components to speed up CRowdPQ evaluation. Similarly to CSRPQ, CRowdPQ relies on extending existing formalism (CRPQ), making it easier to adopt in many existing graph query languages.

  Details on Crowdstore are discussed in Chapter 6. Contribution 4 has originally been presented in [LSSD15].

[8] http://www.uml.org

1.4 Structure of the Thesis

The remainder of this thesis is structured as follows. Chapter 2 provides background information about technologies and concepts used throughout the rest of the thesis. The four main contributions outlined in Section 1.3 are presented in the four core chapters:

- Chapter 3 introduces a query language for expressing complex social formations and efficient query evaluation techniques.
- Chapter 4 presents modeling approach that allows to intuitively capture social collaboration processes.
- Chapter 5 details an integration and coordination framework to simplify implementation of collaboration-assisted computation.
- Chapter 6 introduces a crowdsourcing graph database to enable collaborative aspects in crowdsourcing.

Each of the four core chapters approximately adheres to the same structure:

1. Start with Introduction and Motivation sections, describing use cases of social collaboration.
2. Review related work specific to the chapter in order to show shortcomings of the existing approaches.
3. Present main ideas behind the respective contribution, providing formal definition of query/programming languages or models employed. In addition, Chapters 3 and 6 also discuss query evaluation techniques, while chapter Chapter 5 presents framework architecture design.
4. Thorough evaluation of the proposed approach with respect to the use cases presented in Motivation.
5. Conclude summarizing respective contributions.

Chapter 7 concludes the thesis and outlines the future research directions.

CHAPTER 2

Background

This chapter provides background information about state-of-the-art concepts, approaches, modeling and programming paradigms that form the basis of this thesis.

2.1 Human-related Computing Paradigms

In this section we discuss as to what constitutes human computation, social computing, crowdsourcing and collaborative software and the key differences between them.

Human Computation

The term *human computation* was used in many disjoint research areas, from philosophy to computer science. Interestingly, most modern papers use this term without explicit definition [QB11]. The recent taxonomy by Quinn and Bederson on human computation [QB11] suggests that the modern usage of this term was largely influenced by von Ahn's 2005 dissertation [VA09], which defines *human computation* as *"...a paradigm for utilizing human processing power to solve problems that computers cannot yet solve."*. The taxonomy provides its own consensus as to what constitutes human computation:

- The problems fit the general paradigm of computation, and as such might someday be solvable by computers.
- The human participation is directed by the computational system or process.

An example of human computation is Human-Provided Services (HPS) [STD11], which focus on interaction of humans and services in service-oriented systems.

Social Computing

Contrarily to human computation, social computing focuses on facilitating social interaction between human beings. The definitions of the term *social computing* vary from *"...any*

type of computing that serves as an intermediary or a focus for social relation." [Sch94] to *"Computational facilitation of social studies and human social dynamics as well as the design and use of information and communication technologies that consider social context."* [WCZM07]. While social relation lies at the core of social computing, it is only but an element of complex social formations, such as cliques, k-plexes, n-clubs, etc. For brevity, we delay detailed definitions and analysis of social formations until the next chapter. Examples of social computing technologies are blogs, social networking web sites, online communities, etc. With the advent of social networking platforms, such as Facebook[1], Twitter[2], LinkedIn[3] and many others, social computing is gaining momentum in the research community and even reviving interest in related areas, such as graph query languages.

Collaborative Software

Collaborative software (groupware) is a type of application software designed to help people collaborate on a common task. One of the earliest definitions of collaborative software was *"...computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment."* [EGR91]. Groupware is a rather broad concept encompassing vast variety of applications. One spectrum of groupware classification ranges from real-time software (i.e., people collaborate at the same time, as in Google Docs[4], HackerRank[5], etc.) to asynchronous applications (e.g., e-mail, wikis). Another spectrum goes from distant collaboration to same-place collaboration (i.e., an interactive presentation panel).

Crowdsourcing

The term crowdsourcing was coined by Jeff Howe and defined in his book as *"...an act of taking a task traditionally performed by a designated agent (such as an employee or a contractor) and outsourcing it by making an open call to an undefined but large group of people."* [How08]. A typical crowdsourcing platform allows for publishing tasks that crowd users can contribute to with their time and skills, or even money. Rewards for successful task completion can be monetary, or non-material, such as recognition or intellectual satisfaction. Exposure of programming interfaces by crowdsourcing platforms allows for programmable task publishing and evaluation of results, enabling thus integration of crowdsourcing and human computation concepts. Crowdsourcing finds applications in many different areas, such as rescue missions (find a missing boat in thousands of satellite images [Ols08]), mass artistic works [6], funding (crowdfunding), content creation, problem solving (crowd wisdom) and many others [DRH11, Bra08]. The most widely referred in

[1] http://www.facebook.com
[2] http://www.twitter.com
[3] http://www.linkedin.com
[4] https://www.google.com/docs/about/
[5] https://www.hackerrank.com/
[6] http://www.thesheepmarket.com

literature example of crowdsourcing platform is Amazon Mechanical Turk[7], yet other notable crowdsourcing platforms are Wikipedia[8], Yahoo! Answers[9], and Kickstarter[10].

2.2 Programming Models and Modeling Approaches

This section provides a brief taxonomy of programming models and modeling approaches most relevant for collaboration-based computation.

2.2.1 Graph Query Languages

Graphs are the natural data structure to represent social networks. Graph query languages have been researched starting some 30 years ago [Woo12b]. The ever increasing attention to social networks in research community has brought resurgence of interest in graph query languages, as a natural tool for social network analysis and querying. In its simplest form a graph G is a pair (V, E), where V and E are the finite sets of nodes and edges connecting nodes respectively. A typical query on graph G might ask to find persons who are friends with Adam and Eve - a simple conjunctive query (CQ) returning a set of nodes as an answer. Such query can be expressed using the following syntax: $x\ friendof\ Adam \wedge x\ friendof\ Eve$, which is similar to Datalog. It is common in querying graphs that users might want to find graph nodes connected by a path satisfying a regular expression rather than a path with the exact number of edges. Conjunctive Regular Path Queries (CRPQs) extend simple CQs with the ability to specify a link between two nodes as a regular expression. CRPQs have been thoroughly studied and form the basis for many query languages [CMW87, CM89, AQM+97, FFLS00, BFS00, KSI+08, Tea12] including SPARQL 1.1 [spa12].

Evaluation of graph queries can be classified into three main categories:

- Exact graph matching. This is the most common approach in literature, where queries define precise graph structure and (sub)graphs returned by the query should exactly match the structure of the query graph.

- Approximate graph matching. In this approach queries define precise graph structure, and returned results may not match but rather be "close" to the query graph by means of edit operations (insert/delete/substitute) [KS01, GT06, HPW09].

- Flexible topologies. In this approach queries define "flexible" topologies. GraphQL [HS08] with repetition of graph motifs allows to define not only paths, but also, for example, cycles and trees being thus more expressive in this aspect than regular path expressions. The approach studied in [FLM+11] relaxes constraints on topology of returned graphs in favor of performance. This approach is the most extreme with

[7] https://www.mturk.com/mturk/welcome
[8] http://www.wikipedia.com
[9] https://answers.yahoo.com/
[10] http://www.kickstarter.com

respect to allowed relaxations on topologies, and can lead to completely irrelevant results. It is worth noting though, that even canonical CRPQs allow number of nodes to vary in paths.

2.2.2 Business Process Modeling

Business Process Modeling (BPM) focuses on describing and documenting enterprise business processes in order to improve and analyze them. Techniques to model business processes date back to as early as 1899, when Gantt charts were introduced. Typically, modeling approaches define a formal (visual) notation, that can be easily understood by people not affiliated with IT. Formal definitions allow for automated analysis and verification to discover errors and optimizations, and even generation of source code in various programming or execution languages. Existing modeling approaches can be classified into two main categories:

- Activity-oriented approaches describe processes as sequences of tasks, connected by control- or data-flow links. The most prominent example of activity-oriented modeling approaches is Business Process Modeling Notation (BPMN[11]), which allows for graphical representation of business processes similar to activity diagrams in UML[12]. BPMN models dependencies between processes via messages or events.

- Information-centric approaches describe processes as evolution of business entities, such as documents. Examples of information-centric modeling approaches are Case Handling [vdAWG05] and Artifact-centric workflows [BHS09].

Benefits of BPM are twofold: formal and intuitive documentation of business processes, and reduced maintenance and implementation costs due to code generation rather than reimplementation.

2.2.3 Coordination Languages

Collaboration-assisted computation requires coordination of collaborative actors via groupware platforms. Coordination languages focus on coordination of black-box components with unknown behavior. It is necessary to consider coordination languages via projection of integration of groupware software and social network platforms. Coordination languages can be categorized as follows:

- Control-driven languages [AAB+05, GP07, KCM06] can be considered as incarnation of activity-oriented modeling approaches as they coordinate processes similarly, via messages or events. Activity-oriented modeling approaches often even provide direct mappings to control-driven coordination languages, as in case of the BPMN specification, which provides a mapping between its graphical notation constructs and Business Process Execution Language (BPEL). BPEL allows to define business

[11] http://www.bpmn.org
[12] http://www.uml.org

processes based on Web Services and is specifically designed for integration of various APIs.

- Linda-based languages [BFLM01, BZ05] express coordination as dependencies between removal/reading and insertion of atoms from or into a shared space. Linda-based coordination languages provide the most flexibility as the coordinated actors are unaware of each other.

- Complex Event Processing (CEP) languages [NN08, PE10, SC05] express correlations of events originating from different data streams. Events can be mapped (joined) via their properties/attributes and co-occurrence in time.

Unlike programming languages, coordination languages are typically higher-level, and describe inter-component(service) communication protocols, behaviors and communication topology.

2.2.4 Crowdsourcing Models

We introduced the concept of crowdsourcing in the previous section, in this section we focus on programming models employed in this domain. Existing hybrid human-machine approaches to crowdsourcing-based computation can be classified as:

- Classic. Crowdsourcing platforms expose programming APIs, that can be used by virtually any mainstream programming language. This approach is primitive and involves superfluous efforts on the programmer's side.

- Crowd-as-a-database. This approach extends database models by introducing operators that can be executed by crowd workers, and enabling tuple (columns, tables) generation by the crowd[FKK+11]. The database models employed in this approach are not limited only to relational models, but span over Datalog-like queries [PP11], and graph databases [ASFN12].

- Crowd-MapReduce. A number of studies exploit MapReduce programming model for managing human computation [ABMK11, KSKK11, KCH12]. In the MapReduce programming model a program is described as a dataflow between *Map* and *Reduce* tasks. Map tasks are responsible for data sorting, filtering, and partitioning, while Reduce tasks are responsible for accumulation and aggregation. MapReduce framework manages automatic distribution of defined Map and Reduce tasks between a fixed pool of machines. Crowd-MapReduce approaches allow to replace compuation nodes with crowd workers, integrating thus crowd work into the pipeline. MapReduce allows to restrict machines used for certain tasks by their CPU or memory size, and similarly Crowd-MapReduce approaches allow restricting worker assignment based on worker attributes [ABMK11].

Crowdsourcing programming models play an important role in simplifying integration of human and machine computation.

CHAPTER 3

Expressing Complex Social Formations

Many query languages for graph-structured data are based on regular path expressions, which describe relations among pairs of nodes. We propose an extension that allows to retrieve groups of nodes based on group structural characteristics and relations to other nodes or groups. It allows to express group selection queries in a concise and natural style, and can be integrated into any query language based on regular path queries. We present an efficient algorithm for evaluating group queries in polynomial time from an input data graph. Evaluations using real-world social networks demonstrate the practical feasibility of our approach.

3.1 Introduction

The World Wide Web has rapidly evolved from a network providing access to static webpages to a highly interactive experience. Web-based social networking services connect millions of people and have deeply changed the way we communicate with each other. Graphs are the natural data structure to represent social networks. The swift growth of social networking services fueled with further developments in the Web (e.g., Semantic Web) has led to a rising interest in graph databases and related query languages.

Today, SPARQL [PAG06], the W3C[1] query language for RDF [KC04], is probably the most widely known query language for graph-structured data. SPARQL and many other graph query languages allow users to retrieve nodes based on conjunctive queries (CQs). A simple CQ for finding persons who are friends with Adam and Eve could be expressed as $x\ friendof\ Adam \wedge x\ friendof\ Eve$. An extension called regular path queries (CRPQs) allows querying for nodes that are connected by a path satisfying a regular expression rather than relying solely on static paths. CRPQs have been thoroughly studied and form

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

the basis for many query languages [CMW87, CM89, AQM+97, FFLS00, BFS00, KSI+08, Tea12] including SPARQL 1.1 [spa12]. With CRPQs, however, it is not possible to query sets of nodes. For instance, Adam and Eve might not just want to find their common friends, but a group of common friends that are connected between them. Finding sets of nodes that satisfy certain characteristics is of general interest for graph-structured data, and particularly important for social networks. Let us consider a graph representing the skills and relationships of developers of open source projects. With CRPQ-based query languages it is, for instance, not possible to find socially coherent teams capable of conducting a certain project. While groups are a fundamental concept in social networks they are not well supported by current graph query languages.

In this chapter we propose an elegant and simple, but at the same time expressive, extension - conjunctive set regular path queries (CSRPQs), which extends CRPQs with the notion of sets of nodes. Using the example of SPARQL, we show how CSRPQs can be integrated into existing graph query languages. We show that data complexity of CSRPQ queries is in *PTIME*, and propose a novel algorithm for efficient query evaluation, which leverages structure of the db-graph and the query itself for search space minimization. Experiments show that query evaluation is feasible even for real-world social networks of large scale.

In the remainder of this chapter, we first give a motivating scenario in Section 3.2 and discuss related work in Section 3.3. After recapturing CRPQs in Section 3.4, we present CSRPQs in Section 3.5, and demonstrate their expressivity in Section 3.6. Section 3.7 presents our algorithms for CSRPQ evaluation and reasons about its complexity. An evaluation based on real-world datasets is presented in Section 3.8. Section 3.9 concludes the chapter.

3.2 Motivation

The importance of social ties in software engineering is often emphasized in literature [LKTD12a, DB11]. Conway's law suggests that "organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations" [Con68]. We can rephrase it as: organization's social network structure influences its systems architecture. Also, Peopleware [Oye74] claims the main reason for project failures to be the human factor, i.e., lack of communication or bad social environment, and not the technology factor. Taking into account these two postulates, a proper process of project team selection can be defined as a selection of a subgraph, which exhibits certain structural characteristics, from social and organizational overlay networks. For example, when assembling a project team it is preferable to maximize social coherence within the team as well as to maintain good social connections with other project teams. In social network analysis a number of characteristics and patterns have been defined to characterize social coherence of a group, such as geodesics, n-clique, n-clan, n-club, k-core, k-plex, and so on [Sco00]. Also, efficient coordination and integration of project teams requires selection of special actors with certain structural properties, such as closeness, degree, or betweenness centralities.

Let us consider now a motivating scenario with a few examples. An organization wants to start a new open source software (OSS) project, consisting of a number of sub projects. For this purpose it uses a social network comprised of independent software engineering experts as well as its own employees. In the social network two software engineering experts are considered to be connected if they either worked on the same software engineering project in the past or they are connected in a social networking site. Now, let us consider few examples of queries the organization may want to issue in order to select project teams:

1. Closeness centrality and connectedness. In case of a core component, it may be needed to select a project team that has good social connections to all the other teams, i.e. located in an intersection of their neighborhoods. Moreover, the project team can be either implicitly or explicitly connected. A team is implicitly connected if it forms a connected graph. Otherwise, if team members are connected via external nodes, the team is considered to be implicitly connected. Closeness to other teams improves inter-team communication, while explicit connectedness improves communication within the team itself.

2. Cluster. To select a project team for implementation of a monolithic component (i.e., a component with a lot of interdependencies between its parts) one might want to select a clique (complete graph). This is necessary to maximize, again, communication within the team and ensure that everyone is aware of all the changes introduced by other team members. However, in practice cliques of required size and comprised of necessary experts can rarely be found. Therefore, one may want to relax the requirement by selecting a k-plex, where k-plex is defined as a graph G where every vertex has a degree $|G| - k$.

3. Independent teams. In case of highly reliable software systems a valid practice to minimize probability of failures is to execute several mutually replaceable components in parallel. In order to mitigate failures introduced by a human factor (software defects), it is important that such mutually replaceable components are developed by teams isolated from each other. This is necessary to avoid propagation of erroneous approaches.

4. Liaison. When integrating two different components being developed by project teams that have no direct social relations it may be helpful to search for a liaison/broker between two teams. Such a liaison would facilitate communication and increase chances for successful integration.

5. Structural equivalence. In order to replace an expert, that recently left a project, it is necessary to find a structurally equivalent expert, i.e. an expert that has almost the same social neighborhood as the former expert with respect to dependent project teams. This would decrease on-boarding time and restore structural characteristics of the team. Also, if it is not possible to find such a replacement, we may want to find a group of people that is structurally equivalent.

Figure 3.1 shows an exemplary snippet from a graph representing relations among developers of OSS. Developers are represented as nodes, and two of them are connected by an edge if they are socially connected. One possible solution to each of the five aforementioned queries is highlighted in the figure.

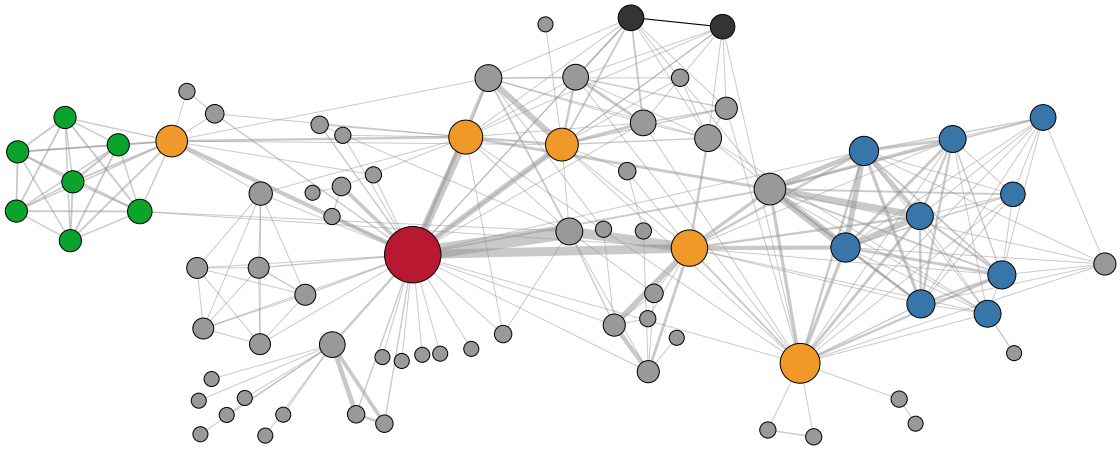


Figure 3.1: Simplified graph representing social relationship among open source software developers. Possible answers for OSS example queries are highlighted: 1) Closeness Centrality: yellow nodes, 2) Cluster: green nodes (and blue nodes in case of 1-plex), 3) Independent teams: green nodes for first team and blue nodes for second team, 4) Liaison: red node, 5) Structural equivalence: the two black nodes connected with bold edge can replace each other

In the next section we discuss existing graph query languages and show their shortcomings with respect to the query examples enlisted above.

3.3 Related Work

In this section we discuss existing graph query languages and consider their capabilities of expressing the queries outlined in the previous section. A comprehensive overview of existing graph query languages can be found in [Woo12a] and [AG08]. Functionality of canonical *Conjunctive Regular Path Queries* (CRPQs) has been employed in many graph query languages, such as **G** [CMW87], GraphLog [CM89], Lorel [AQM+97], StruQL [FFLS00], UnQL [BFS00], NAGA [KSI+08], Cypher [Tea12], and SPARQL 1.1 [spa12].

Query 1 in the motivating scenario defined in CRPQs would search for a disconnected set of nodes satisfying the neighborhood condition, thus being incapable of finding a tightly connected subgraph in a neighborhood. Queries expressed in CRPQ-based (and not only) languages typically return node tuples, and are not able to return groups of varying size. One exception here is NAGA, whose queries return graphs. Extended Conjunctive Regular Path Queries [BHLW10] can return paths along with nodes, which is somewhat similar to returning groups. Another interesting exception is SQL-based PQL [Les] language, which can return nodes with their neighborhoods. *Query 2* is flexible in the sense that it does specify neither exact number of nodes in a group to be found, nor its exact topology, but rather general characteristics. In literature there

were a number of approaches employed for definition of "flexible" topologies in query languages. Even canonical CRPQs allow number of nodes to vary in paths. GraphQL [HS08] with repetition of graph motifs allows to define not only paths, but also, for example, cycles and trees being thus more expressive in this matter than regular path expressions. Approach studied in [FLM+11] relaxes constraints on topology of returned graphs in favor of performance. This approach is the most extreme with respect to allowed relaxations on topologies, and can lead to completely irrelevant results. Queries specifying "flexible" topologies are quite the opposite approach to approximate graph matching [KS01, GT06, HPW09], where queries define precise graph structure, and returned results may not match but rather be "close" to the query graph by means of edit operations (insert/delete/substitute).

In the domain of social networks there are several dedicated graph query languages, such as SoQL [RS09], BiQL [DNDR09], SocialScope [AYLY09], SNQL [MGW11], QGraph [BIJ02], as well as extensions of SPARQL for social network analysis [EBGC09]. SoSQL is the only one of them that can specify flexible selection of groups of nodes with SELECT FROM GROUP queries. Its expressivity, however, is limited with respect to graph patterns used in social network analysis, as it is not capable, for example, to select groups exhibiting characteristics of k-plex, n-clique, n-club, etc. Also, it does not have groups as first-class citizens, i.e., it is not possible to specify paths or connections between two or more groups. For instance, in *Query 3*: select two groups that are mutually isolated. Selection of actors, as in *Query 4* and *Query 5*, is also not supported. Finally, SoSQL provides no algorithm for query evaluation, which seems to be a complex computational problem.

BiQL [DNDR09] suggests integration of external tools for finding graphs with certain characteristics, e.g., quasi-cliques or clusters. Its distinguishing feature is unification of nodes and edges, which makes it possible, though with integration of external tools and algorithms, to find clusters of edges and not only clusters of nodes. Integration of external tools for querying certain graph patterns, however, reduces flexibility of a query language with respect to definition of possible graph patterns. SocialScope [AYLY09] defines an algebraic language with node and link selection operators, union, disjunction, subtraction and composition of graphs, as well as set and numerical aggregations. Node and link selection can utilize an optional scoring function. SNQL [MGW11] is a query language of similar functionality, as it has been claimed to cover SocialScope [RGW11]. SNQL is intended for data management in social networks. It extends GraphLog with Skolem functions to create new nodes as part of the output. Both languages do not allow flexible selection of groups and actors exemplified in the motivating scenario. Extension of SPARQL [EBGC09] for social network analysis can examine global metrics of a graph, such as density and diameter. However, it is not capable to search for groups exhibiting specific metrics. QGraph [BIJ02] is a visual query language employed in a tool called Proximity [JN03], which is used for data mining in social networks. QGraph queries graph patterns can have numeric annotations, e.g., 'Find all directors that had at least 2 movies each of them winning at least 3 awards'. Such annotations resemble basic quantitative conditions needed for k-plex, e.g., each node has at least N neighbors in the

group. However, QGraph does not have notion of groups to succeed in such selection.

Many efficient algorithms (e.g., [BE93, CN85]) were proposed for selection of social formations exhibiting certain structural characteristics (e.g., regular equivalence, n-clique, k-plex, n-club), and implemented in such popular social network analysis tools as Pajek [BM02] and Ucinet [BEF02]. Being capable of handling some examples in the motivating scenario, these algorithms, however, are limited to specific problems they address and are not as flexible and expressive as a query language might be. While offering far greater expressivity, a query language requires a complete and generalized query interpretation algorithm capable of solving mixed and combined problems.

The overview of related work shows that no existing language is able to fully cope with the queries enumerated in Section 3.2. In the next section we provide definition of CRPQs, and then, in Section 3.5, we present our extension.

3.4 Preliminaries

A database is defined as a directed graph $K = (V, E)$ labeled over the finite alphabet Σ. If there is a path between node a and node b labeled with $p_1, p_2, ..., p_n$ we write $a \xrightarrow{p_1 p_2 ... p_n} b$. In the remainder of this section we give definitions of (conjunctive) regular path queries, similar to other works, like [CGLV00a].

Definition 1 (Regular Path Queries). A regular path query (RPQ) $Q^R \leftarrow R$ is defined by a regular expression R over Σ. The answer $ans(Q^R, K)$ is the set connected by a path that conforms to the regular language $L(R)$ defined by R:

$$ans(Q^R, K) = \{(a, b) \in V \times V \mid a \xrightarrow{p} b \text{ for } p \in L(R)\}.$$

Conjunctive regular path queries allow to create queries consisting of a conjunction of RPQs, augmented with variables.

Definition 2 (Conjunctive Regular Path Queries). A conjunctive regular path query (CRPQ) has the form

$$Q^C(x_1, ..., x_n) \leftarrow y_1 R_1 y_2 \wedge ... \wedge y_{2m-1} R_m y_{2m},$$

where $x_1, \ldots, x_n, y_1, \ldots, y_m$ are node variables. The variables x_i are a subset of y_i (i.e., $\{x_1, \ldots, x_n\} \subseteq \{y_1, \ldots, y_m\}$), and they are called distinguished variables. The answer $ans(Q^C, K)$ for a CRPQ is the set of tuples $(v_1, ..., v_n)$ of nodes in K such that there is a total mapping σ to nodes, with $\sigma(x_i) = v_i$ for every distinguished variable, and $(\sigma(y_i), \sigma(y_{i+1})) \in ans(Q^R, K)$ for every RPQ Q^R defined by the term $y_i R_i y_{i+1}$.

3.5 Conjunctive set regular path queries

In this section we describe how CRPQs can be extended to overcome their shortcoming for finding sets of nodes. Our extensions allow to express all queries presented in the motivating scenario. The proposed extensions can be used to augment any graph query language that employs CRPQs, like SPARQL.

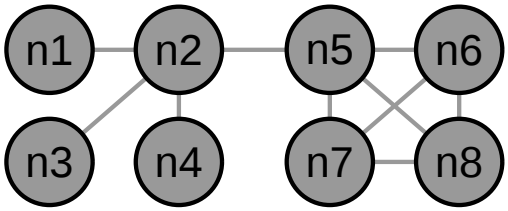


Figure 3.2: Exemplary graph K, serving as knowledge base for some simple queries. Nodes represent persons and edges represent the friendship relation. Since we assume that this is a symmetric relation, arrows have been omitted.

Before we introduce set regular path queries (SRPQs), which extend RPQs, we introduce a set of generalized quantifiers. SRPQs allow to make statements about which fraction of a set is affected by a path query. For giving SRPQs the expressiveness necessary to handle sets we allow quantifiers beyond the standard quantifiers $\forall$ and $\exists$, similar as proposed in [Mos57]. All extended quantifiers refer to a certain set. In the following we define the quantifiers we use by showing the mapping they signify with relation to some arbitrary set M.

- Universal quantification $\forall_M = \{M\}$
- Existential quantification $\exists_M = \{A \subseteq M : A \neq \varnothing\}$
- Counting quantification: $\exists_M(\odot n) = \{A \subseteq M : |A| \odot n\}$, where $\odot \in \{>, \geqslant, =, \leqslant, <\}$ and $n \in \mathbb{N}$
- Fractional quantification: $\exists_M(\odot p) = \{A \subseteq M : |A| \odot p|M|\}$, where $\odot \in \{>, <\}$ and $p \in [0, 1]$

We use capital Greek letters Ξ and Ψ as placeholders for one of the above defined quantifiers. SRPQs are similar to RPQs, but extend them with the notion of sets. They come in different flavors since we distinguish between paths from a single node to a set of nodes, paths from a set to a single node, and paths from a set of node to another set of nodes. Like RPQs, all SRPQs are defined by a regular expression R over Σ.

Definition 3 (SRPQ: Node – Set). A set regular path query $Q^{\bullet\Xi} \leftarrow R$ describes a relation between a single node and a set, based on a regular expression R together with a quantifier Ξ. The quantifier defines to how many nodes from the set the single node must be connected by a path conforming to the regular language $L(R)$. The respective answer set $ans(Q^{\bullet\Xi}, K)$ is defined as

$$\{(a, B) \in V \times 2^V : \ a \xrightarrow{p} b \text{ for } \ b \in \Xi_B \ , \ p \ \in L(R)\}.$$

The following example queries refer to graph K, as defined in Figure 3.2. SRPQ $Q^{\bullet\forall} \leftarrow friendof$ requires that a node is connected to all nodes within a set. Thus,

a partial answer set is $\{(n1, \{n2\}), (n2, \{n1, n3, n4, n5\})\} \subset ans(Q^{\bullet\forall}, K)$. The query $Q^{\bullet\exists<3} \leftarrow friendoffriendof?$ requires that the node is connected to less than three nodes within the set. Nodes are connected when there is a "friendof" path of length one or two. A partial answer set of this query is $\{(n1, \{n2, n5, n6, n8\}), (n2, \{n1, n8\})\} \subset ans(Q^{\bullet\exists<3}, K)$.

Definition 4 (SRPQ: Set – Node). A set regular path query $Q^{\Xi\bullet} \leftarrow R$ describes a relation between a set and a single node, based on a regular expression R together with an quantifier Ξ. The quantifier defines how many nodes within the set must be connected to the single node by a path conforming to the regular language $L(R)$. The respective answer set $ans(Q^{\Xi\bullet}, K)$ is defined as

$$\{(A, b) \in 2^V \times V : \ a \xrightarrow{p} b \text{ for } a \in \Xi_A, \ p \ \in L(R)\}.$$

The following example queries refer to graph K, as defined in Figure 3.2. Query $Q^{\exists\bullet} \leftarrow friendoffriendof$ defines that there must be at least one node in the set that is connected to the single node by a "friendof" path of length two. A partial answer set is $\{(\{n1, n2\}, n5), (\{n4\}, n1)\} \subset ans(Q^{\exists\bullet}, K)$. The second example query defined as $Q^{\exists>50\%\bullet} \leftarrow friendof$ requires that more than half of the nodes within the set are directly connected to the single node. A partial answer would be the set $\{(\{n1, n4, n7\}, n2), (\{n1\}, n2)\} \subset ans(Q^{\exists>50\%\bullet}, K)$.

Definition 5 (SRPQ: Set – Set). A set regular path query $Q^{\Xi\Psi} \leftarrow R$ describes a relation between two sets, based on a regular expression R together with two quantifiers Ξ and Ψ. The quantifiers define how many nodes from within the "left" set must be connected to how many nodes from the "right" set by a path conforming to the regular language $L(R)$. The respective answer set $ans(Q^{\Xi\Psi}, K)$ is defined as

$$\{(A, B) \in 2^V \times 2^V : \ a \xrightarrow{p} b, \ a \in \Xi_A, \ b \in \Psi_B, \ p \in L(R)\}$$

As before, the following example queries refer to graph K, as defined in Figure 3.2. Query $Q^{\forall\forall} \leftarrow friendof$ requires that all nodes within the first set are friends with all nodes in the second set, for which a partial query answer is $\{(\{n5, n6\}, \{n7, n8\}), (\{n1\}, \{n2\})\} \subset ans(Q^{\forall\forall}, K)$. Query $Q^{\exists,\exists>2} \leftarrow friendoffriendof?$ determines that there must be at least one element in the first set that is connected to more than two elements in the second set by a path of length one or two. Thus, a partial query answer would be the set $\{(\{n1, n5\}, \{n6, n7, n8\})\} \subset ans(Q^{\exists,\exists>2}, K)$.

Through introduction of set variables SRPQs extend RPQs in a similar way as *Monadic Second-Order Logic (MSOL)* extends *First-Order Logic (FO)*. Along with set variables *MSOL* introduces an atomic formula $t \in S$, where t is a first-order term and S is a set variable. Next, we show how this atomic formula can be expressed in SRPQs. Empty string $\perp$ is a valid regular expression. However, it is never used in RPQs, as $\perp$ path does not convey any functional load. In SRPQs it does: the query $a \xrightarrow{\perp} B$ means that node a is an element of B, and $A \xrightarrow{\perp} B$ defines the subset relation.

In RPQs paths between nodes are specified over the input graph G. Specification of more advanced structural properties, like explicitly connected groups, requires greater flexibility on this matter. Therefore, we define the following query type.

Definition 6 (SRPQ Closure). For the above defined query flavors $Q^{\Xi\Psi}$ and $Q^{\Xi\bullet}$ the closures $\bar{Q}^{\Xi\Psi}$ and $\bar{Q}^{\Xi\bullet}$ further restrict the answer set by requiring that paths connecting A and B/b stay within A, i.e.,

$$ans(\bar{Q}^{\Xi\bullet}, K) = \{(A, b) \in 2^V \times V : a \xrightarrow{p} b \text{ for } a \in \Xi_A,\ p \in L(R) \text{ and}$$

$$\forall i \in \{1, \ldots, n-1\}\ a \xrightarrow{p_1 \ldots p_i} c \implies c \in A,\ \text{for } p = p_1 \ldots p_n\}$$

and

$$ans(\bar{Q}^{\Xi\Psi}, K) = \{(A, B) \in 2^V \times 2^V : a \xrightarrow{p} b \text{ for } a \in \Xi_A,\ b \in \Psi_B,\ p \in L(R) \text{ and}$$

$$\forall i \in \{1, \ldots, n-1\}\ a \xrightarrow{p_1 \ldots p_i} c \implies c \in A,\ \text{for } p = p_1 \ldots p_n\}.$$

The following example queries again refer to graph K, as defined in Figure 3.2. Query $\bar{Q}^{\exists\bullet} \leftarrow friendoffriendof$ defines that there must be at least one node in the set that is connected to the single node by a "friendof" path of length two. A partial answer set is $\{(\{n1, n2\}, n5)\} \subset ans(\bar{Q}^{\exists\bullet}, K)$, but $(\{n1\}, n5) \notin ans(\bar{Q}^{\exists\bullet}, K)$.

Definition 7 (Set Size Query). A set size query $Q^{|\cdot|} \leftarrow (from, to)$ describes an unary relation. The variables $from$, $to \in \mathbb{N}$, with $from \leqslant to$, define minimum and maximum allowed set sizes. The respective answer set is defined as the sets of subsets of sizes from $from$ to to.

$$ans(Q^{|\cdot|}, K) = \{A \in 2^V :\ |A| \in \{from, \ldots, to\}\}$$

The following example queries refer to graph K, as defined in Figure 3.2. Query $Q^{|\cdot|} \leftarrow \{2, 3\}$ requires that all sets have size 2 or 3. A partial query answer is $\{\{n1, n2\}, \{n1, n3\}, \{n1, n2, n3\}\} \subset ans(Q^{|\cdot|}, K)$.

Definition 8 (Conjunctive Regular Set Path Query). A conjunctive regular set path query (CSRPQ) has the form

$$Q^S(x_1, ..., x_n) \leftarrow \tilde{y}_1[(R_1)^{\Psi_1^1 \Psi_2^1}]y_2 \wedge ... \wedge \tilde{y}_{2m-1}[(R_m)^{\Psi_1^m \Psi_2^m}]y_{2m} \wedge\ \ Z_1[f_1, t_1] \wedge ... \wedge Z_l[f_l, t_l]$$

where $x_1, \ldots, x_n, y_1, \ldots, y_m$ are either node or set variables. $Z = \{Z_1, ..., Z_l\}$ represents all set variables among y_i, i.e., there is no set variable y_k such that $y_k \notin Z_i$. The variables x_i are among y_i. The $\sim$ symbol may be either empty or $-$; the latter case is only possible if y_i is a set and defines a SRPQ closure. Each of the Ψ is either a quantifier, or $\bullet$. Each R_i is a regular expression. The answer set $ans(Q^S, K)$ for a CSRPQ is the set of tuples $(v_1, ..., v_n)$ of nodes and sets of nodes in K such that there is a total mapping σ to nodes and sets of nodes with $\sigma(x_i) = d_i$ for every distinguished variable, and $(\sigma(y_i), \sigma(y_{i+1})$ in the set of the answer set of the respective query type; i.e., if y_{2i-1} and y_{2i} represent both nodes, then $y_{2i-1} R_i y_{2i}$ represents Q^R. If y_{2i-1} is node and y_{2i} is set, then $Q^{\bullet\Xi}$. If y_{2i-1} is set and y_{2i} is node, then $Q^{\Xi\bullet}$. If y_{2i-1} is set and y_{2i} is set, then $Q^{\Xi\Psi}$. And finally, $Z_i[f_i, t_i]$ denotes set size queries.

3.6 CSRPQ Expressiveness

In this section we demonstrate the expressiveness of CSRPQs via formal specification of the use cases from the motivating scenario. For this purpose we use the formal notation defined in the previous section. Also, we exemplify ease of CSRPQs integration into CRPQ-based languages by showing how the same queries could be implemented in CSRPQ-enhanced version of SPARQL 1.1.

3.6.1 Closeness Centrality and Connectedness

Goal. Given the three predefined organizational groups `Team1`, `Team2`, and `Team3`, we need to assemble an explicitly connected team of three to five members with a maximum diameter of two. The team should have a connection to every group.

Formal specification. In knowledge graphs organizational groups are usually represented as single nodes, e.g., `Sales Department` or `Human Resources Department`, and affiliation of a person to a group is represented as a connection between the corresponding nodes. We use the same approach and assume there are three predefined nodes: `Team1`, `Team2`, and `Team3`. Affiliation of a person to a team is represented with the *inteam* edge, and social connection is represented with the *knows* edge in the semantic graph. Question mark applied to an atom, e.g., *knows?*, in regular expression specifies that the edge is optional.

$$
\begin{aligned}
Q^S(A) \quad \leftarrow \quad & A\,[(knows \cdot inteam)^{\exists}]\mathbf{team1} \\
\wedge \quad & A\,[(knows \cdot inteam)^{\exists}]\mathbf{team2} \\
\wedge \quad & A\,[(knows \cdot inteam)^{\exists}]\mathbf{team3} \\
\wedge \quad & \bar{A}\,[(knows \cdot knows?)^{\forall\forall}]A \wedge A[3,5]
\end{aligned}
$$

SPARQL. In the formal notation we used capital letters to define variables representing groups. In SPARQL we use two question marks (??) for this purpose in analogy with single question marks (?) that precede node variables. **ALL** and **SOME** keywords are used to denote universal and existential quantification respectively, and **CLOSURE** keyword denotes a query closure. To conform to SPARQL semantics we put group size operator in the **FILTER** clause.

Listing 3.1: Closeness centrality and connectedness

```
SELECT ??A
WHERE {
 SOME ??A knows/inteam 'Team1'.
 SOME ??A knows/inteam 'Team2'.
 SOME ??A knows/inteam 'Team3'.
 ALL CLOSURE(??A) knows{1,2} ALL ??A .
 FILTER ( ??A{3,5} ) }
```

Expressiveness. This example shows how explicitly connected teams can be defined with CSRPQs. Note, that implicitly connected teams can be defined by removing the query closure.

3.6.2 Cluster

Goal. Find a socially coherent group of people of size 10, which exhibits property of 2-plex.

Formal specification. In contrast to the previous example, for the sake of simplicity we omit specification of neighborhoods. According to the definition of 2-plex, we need to find a team, where every team member has at least $8 = 10 - 2$ connections to other team members.

$$Q^S(A) \leftarrow A\,[(friendof)^{\forall,\exists\geqslant 8}]A \wedge A[10,10]$$

SPARQL. In this query parameterized version of **SOME** keyword denotes $FO(COUNT)$ quantification. This goes in line with previous example, where **SOME** keyword without parameters denotes existential quantification.

Listing 3.2: Cluster

```
SELECT ??A
WHERE {
 ALL ??A knows SOME(>=8) ??A.
 FILTER ( ??A{10,10} ) }
```

Expressiveness. This example shows succinctness of CSRPQs with respect to definition of structures with relaxed coherence.

3.6.3 Independent Teams

Goal. Find two independent teams of experts, i.e., no inter-team social connections exist.

Formal specification. Here we specify neither any search space, nor structural constraints for the teams A and B. The interesting peculiarity is that negation expressed in the query goal (i.e., no inter-team social connections exist) can be expressed with countable quantifiers.

$$Q^S(A) \leftarrow A\,[(knows)^{\forall,\exists=0}]B \wedge B\,[(knows)^{\forall,\exists=0}]A \wedge A[3,5] \wedge B[3,5]$$

SPARQL. This query does not introduce any additional extensions to SPARQL and simply reuses keywords already defined in the previous examples.

Listing 3.3: Independent teams

```
SELECT ??A ??B
WHERE {
 ALL ??A knows SOME(=0) ??B
```

```
ALL ??B knows SOME(=0) ??A
FILTER ( ??A{3,5}, ??B{3,5})
}
```

Expressiveness. This example shows expressivity of CSRPQs with respect to definition of multiple groups. Also, it shows how non-existence can be expressed with extended quantifiers.

3.6.4 Liaison

Goal. Find a liaison between `Operational Department` and a team of three external consultants **C** in order to foster adoption of new practices. The liaison should know at least two out of the three consultants, and at least 70% of `Operational Department` members.

Formal specification. In this example, in contrast to the previous ones, it is necessary to find an actor, not a group. The group **T** used in the example should represent the whole department. Therefore, in order to avoid selection of subgroups we fix size of **T** to the known size of `Operational Department`, which is retrieved as an additional query expressed in canonical $CRPQ^{agg}$ (CRPQs extended with aggregation).

$$C := \{\mathbf{jill}, \mathbf{jade}, \mathbf{jack}\}$$
$$Q^{count}(t) \leftarrow t\ [(inteam)]\mathbf{optdept}$$
$$Q^{S}(a) \leftarrow a\ [(knows)^{>0.7}]T \wedge T\ [(inteam)^{\forall}]\mathbf{optdept}$$
$$\wedge a\ [(knows)^{>2}]C \wedge T[Q^{count}, Q^{count}]$$

SPARQL. This example shows that notion of groups also allows specification of predefined constant groups, i.e., group of four consultants. For this purpose we use syntax reminiscent to many mainstream programming languages. In order to retrieve size of `Operational Department` we use a nested SPARQL query.

Listing 3.4: Liaison

```
SELECT ?a
WHERE {
 ?a knows SOME(>70%) ??T.
 ALL ??T inteam 'OptDept'.
 ?a knows SOME(>2)
    {'jill', 'jade', 'jack'}.
 {
  SELECT COUNT(?t) as ?c
  WHERE {?t inteam 'OptDept'}
 }
 FILTER ( ??T{?c,?c} )
}
```

Expressiveness. This example shows how CSRPQs can define selection of single nodes based on relations to groups.

3.6.5 Structural Equivalence

Goal. In order to replace a coordinator *John* we need to find a structurally equivalent person, i.e., a person that has the same social and organizational connections as *John*. Sometimes, however, it may not be possible to find a single person fulfilling this requirement, and a group of persons may be needed.

Formal specification. This example is somewhat similar to the previous one as we want to consider the whole neighborhood of a predefined node, e.g., group (T) in both examples. Therefore, for the sake of clarity, we omitted the aggregation query as it can be, with a small adjustment, reused from the previous example.

$$Q^S(EQ) \leftarrow \textbf{john}\ [(workswith)^{\forall}]T \wedge EQ\ [(workswith)^{\forall}]T \wedge EQ[1,3] \wedge T[Q^{count}, Q^{count}]$$

SPARQL. Again, to keep the definition succinct, in SPARQL implementation we also omitted the nested aggregation query.

Listing 3.5: Structural equivalence

```
SELECT ??EQ
WHERE {
 'john' workswith ALL ??T.
 SOME ??EQ workswith ALL ??T.
 FILTER ( ??EQ{1,3},
        ??T{?count,?count})}
```

Expressiveness. This query exemplifies an opposite approach to the one employed in the previous example. In the previous example we allowed to neglect certain connections in favor of the size of the resulting group (single node), while in this example we relax the constraint on the resulting group size in order to preserve all connections.

3.7 Complexity of Query Evaluation

In this section we discuss complexity of CSPRQs evaluation, optimization techniques enabled by the design of CSRPQs, and propose an algorithm for the purpose of query evaluation. Greater expressiveness always comes at a price, and CSRPQ extension is no exception here. The problem of SRPQ evaluation contains a variant of a subset selection problem, e.g., find a subgraph connected to a node a, incurring thus high complexity. In the next two subsections we discuss upper bounds for the problem of CSRPQ evaluation.

3.7.1 Data Complexity

We fix query Q over a finite alphabet Σ, and K to be the maximum of upper bounds for all the groups specified in the query. Now, to show data complexity of CSRPQs we consider the following decision problem:

PROBLEM:	CSRPQS-EVAL(Q)
INPUT:	A Σ-labeled *db*-graph G of size N, a tuple of nodes $\overline{v}$, and a tuple of groups $\overline{V}$
QUESTION:	Does $(\overline{v}, \overline{V})$ belong to $Q(G)$

Since number of groups is fixed in the query, let us denote it as some constant C. Similarly, let us define c as a number of nodes in the query. In order to answer the problem we need to traverse N^{c+K*C} possible solutions. Since c, C, and K are all fixed, we can see that data complexity of CSRPQ evaluation is in *PTIME*.

3.7.2 Query Complexity

We turn now to query complexity, i.e., where a query Q is an input parameter, while Σ-labeled graph G is fixed.

PROBLEM:	CSRPQS-EVAL(G)
INPUT:	A CSRPQ query Q over Σ, a tuple of nodes $\overline{v}$, and a tuple of groups $\overline{V}$
QUESTION:	Does $(\overline{v}, \overline{V})$ belong to $Q(G)$

Let us define $k_1, ..., k_l$ as upper bounds of groups specified in Q, where l is number of groups. Let us also define c as number of nodes in the query graph. In order to answer the problem we need to traverse $N^{c+k_1+...+k_l}$ possible solutions. Given that N is fixed we can see that query complexity of CSRPQ evaluation is in *EXPTIME*.

3.7.3 The CSRPQ Evaluation Algorithm

The key observation behind CSRPQ evaluation is the mandatory presence of constraints on the group size. A candidate for inclusion in a set should not only preserve structural characteristics of the set, but also enable other potential candidates to be added later in order to satisfy the set size constraints. This approach enables for efficient search space pruning techniques. For example, a node with degree 5 cannot be selected as a candidate for a clique of size 10. Without the loss of generality we can consider CSRPQs with any number of predefined node/set constants and only one set variable specified. Such an CSRPQ technically can be represented as a composite filter $F(G, S, K)$ that selects potential candidates from the search space S for inclusion in assembled so far set G with target set size upper bound K. Given queries $Q^{\Xi\bullet}_{const}$, where the left operand is the single set variable and the right operand is a constant node $\mathbf{c}$, and queries $Q^{\Xi\Psi}_{self}$, where both operands are the single set variable itself, let us consider several examples of filters $F(G, S, K)$:

- $Q^{\exists>5\bullet}_{const} \leftarrow R$: $F_Q(G, S, K) := f(s) \rightarrow |G^*| > 5 \vee K > 5 - |G^*| \vee s \overset{p \in L(R)}{\rightarrow} \mathbf{c}$, where $G_* = \{g^* \in G | g^* \overset{p \in L(R)}{\rightarrow} \mathbf{c}\}$.

- $Q^{\forall\forall}_{self} \leftarrow R.\ F_Q(G,S,K) := f(s) \rightarrow \forall g \in G : g \overset{p\in L(R)}{\rightarrow} s \wedge s \overset{p\in L(R)}{\rightarrow} g \wedge d(s) \geqslant K - |G|$, where $d(s)$ denotes an out degree of a node s.
- $Q^{\forall\exists\leqslant m}_{self} \leftarrow R.\ F_Q(G,S,K) := f(s) \rightarrow \forall g \in G : (d_G(g) \geqslant \min\{m, |G| - m + 1\} \vee g \overset{p\in L(R)}{\rightarrow} s) \wedge d_G(s) \geqslant \min\{m, |G| - m + 1\}$, where $d_G(s)$ denotes an out degree of a node s towards elements of a set G.
- $\bar{Q}^{\forall\forall}_{self} \leftarrow R.\ F_Q(G,S,K) := f(s) \rightarrow \exists p \in L(R), |p| = 1, \forall g \in G, \exists u \in L(R) : pu \in L(R) \wedge s \overset{pu}{\rightarrow} g$, where for each u there exists a physical path corresponding to u that resides within G.

Algorithm 3.1 sketches basic inductive routine for CSRPQ evaluation. Given a set G, we say that it does not violate CSRPQ Q, if it is possible to add nodes to G, such that all SRPQs in Q are satisfied including the set size query. The algorithm takes set G of size L, and tries to build a set G^* of size $L+1$, $G \subset G^*$. The core of the algorithm, filter function F_Q represents conditions of CSRPQ Q violation. F_Q returns a set of nodes from the search space, such that the set G^*, induced by adding any of them to G, does not violate Q. Then, the algorithm traverses through each of the nodes returned by F_Q outputting all solutions.

Algorithm 3.1: CSRPQ evaluation $CSRPQ_{EVAL}$

Input: Search space S,group size K;
program stack G, list of already processed nodes P;
CSRPQ Q;
Result: List of graphs

```
1  if K = 0 then
2  |  output G
3  end
   /* initialize the neighborhood function                    */
4  f_N(s) ← F_Q(S, G, K) ;
   /* filter the neighborhood                                 */
5  N ← {s_i ∈ S/P | f_N(s_i)};
6  for neighbor ∈ N do
7  |  G* ← G ∪ {neighbor};
8  |  CSRPQ_EVAL(S, K − 1, G*, P ∪ {neighbor}, Q);
9  |  P ← P ∪ {neighbor};
10 end
```

As a basis for $CSRPQ_{EVAL}$ algorithm we took the algorithm proposed by Chiba and Nishizeki [CN85], which has been designed for search of complete subgraphs (cliques) of predefined size k. They have shown that the algorithm has time complexity $O(ka(G)^{k-2}m)$, where G is an input graph, m is number of edges and $a(G)$ is

arboricity of the graph. The arboricity of a graph is a measure of how dense the graph is: graphs with many edges have high arboricity, and graphs with high arboricity must have a dense subgraph.

As it can be seen, performance of $CSRPQ_{EVAL}$ heavily depends on the efficient reduction of the search space at each step. Therefore, given a query graph Q, graphs containing high number of subgraphs that satisfy Q, or are close to Q within small edit distance, result in longer execution times.

$CSRPQ_{EVAL}$ algorithm can be easily converted to a greedy algorithm. Indeed, instead of traversing in undefined order all nodes in N, it can first choose the fittest one in order to come up with a first solution faster. Randomization approaches are also possible to balance query evaluation time. Greedy and randomization approaches are crucial when it is enough to return only one result. Also, similarly to the subset selection problem, our problem exhibits natural data parallelism, making thus applicable parallel data processing techniques.

So far we discussed only evaluation of queries defining groups with structural constraints, i.e., constraints that specify relations within the group. Absence of such constraints may introduce high space complexity incurred by large result set. For example, a CSRPQ query specifying selection of a group F without any predefined structure from a finite search space S could yield $|P_{\leqslant|F|}(S)|$ possible results, where $|P_{\leqslant|F|}(S)|$ is a power set of S of cardinality $|F|$. The result set, however, can be represented as a single element describing a commutative monoid corresponding to the power set. This approach would discard time required to enumerate possible answers as well as minimize space needed for the output.

3.8 Evaluation

Query evaluation performance is one of the key attributes of database query language success. In Section 3.7 we have shown that data complexity of CSRPQ evaluation is in *PTIME*. Since data complexity of CRPQ evaluation is in *NLogspace* and $NLOGSPACE \subset PTIME$ it is hard to reason about actual query evaluation times. Therefore, we quantify the evaluation time of CSRPQ queries by conducting experiments. In the remainder of this section we compare the time of CSRPQ evaluation based on our approach (Algorithm 1) with a baseline. Also, we investigate key aspects influencing CSRPQ evaluation time.

3.8.1 Evaluation Setup

Our evaluation is based on two real-world data sets from the social networks Slashdot[2] and Friendster[3]. Both data set graphs consist of anonymous users represented as nodes and social relationships represented as edges. Friendster is a social gaming web site and Slashdot is a community-enabled news website. Table 3.8.1 depicts main structural

[2]http://snap.stanford.edu/data/soc-Slashdot0811.html
[3]http://archive.org/details/friendster-dataset-201107

characteristics of the two data set graphs. Due to the large size of the Friendster graph, which exceeded our evaluation computer's physical memory constraints, we extracted a subgraph consisting of around 11 million nodes out of the full graph, which contains almost 120 million nodes.

	Friendster	Slashdot
#Nodes	10,999,986	77,359
#Edges	297,395,506	905,468
Avg Node Degree	27.036	11.705
Node Degree Dev	94.939	36.844
Max Node Degree	4,014	2,508
Avg Clustering	0.169	0.0555
Clustering Dev	0.309	0.382

Table 3.1: Main characteristics

The queries chosen for evaluation try to cover a large spectrum of CSRPQs by varying structural constraints (e.g., how tightly connected the group should be), the search space size (e.g., in what neighborhood the group should be searched for), and the size of the group defined by the query. Query 1 represents a query with strict structural constraints (complete subgraph), while Query 2 looks for less strictly defined groups (k-plex). For each query the group size is provided as input parameter, while the search space size is varied by both input parameter *rand* and its neighborhood depth.

Query 1 selects a complete graph A of size n in the neighborhood of input node **rand**. The neighborhood includes friends of the predefined node, as well as friends of friends.

$$\begin{aligned} Q_1(A) \quad &\leftarrow \quad \mathbf{rand}[(friendOf friendOf?)^{\forall}]A \\ &\wedge \quad A[(friendOf)^{\forall\forall}]A \\ &\wedge \quad \mathbf{rand}[(friendOf)^{\exists}]A \wedge A[\mathbf{n},\mathbf{n}] \end{aligned}$$

Query 2 selects a 2-plex B from the neighborhood of input node **rand**. Here numeral n not only specifies group size, but also its structural characteristics.

$$\begin{aligned} Q_2(B) \quad &\leftarrow \quad \mathbf{rand}[(\bot)^{\exists}]B \\ &\wedge \quad B[(friendOf)^{\forall\exists(\geqslant \mathbf{n}-2)}]B \wedge B[\mathbf{n},\mathbf{n}] \end{aligned}$$

Strictness of structural constraints influences the number of results: Query 1 above describes more rigid structural constraints than Query 2, e.g., there might be no cliques in a graph, but many 2-plexes. For the input parameter n, which defines the group size, we choose values 5, 10, 15, and 20, as they seem to be reasonable for real life group selection tasks. For the input node *rand* we randomly chose evenly distributed non-isolated nodes, 200 nodes for Friendster and 100 nodes for Slashdot, to get informative, averaged results.

All CSRPQ evaluation scenarios are implemented in Java 1.7.0 x64, and time values are measured using the standard API's *System.nanotime()* method. The system configuration

used for the tests is: Intel Core i7 2840QM (2GHz, Sandy Bridge), 8GB DDR3 1333MHz RAM, Windows 7 x64. Even though Algorithm 1 is inherently parallel, we execute all experiment runs in a single thread in order to show the real running time.

The baseline algorithm is based on transforming CSRPQ to SPARQL queries and evaluating them using Apache Jena[4]. Only the pure SPARQL evaluation time is measured, without query transformation overhead. The transformation is necessary because SPARQL does not support queries for group patterns. However, it is possible to rewrite a fixed query expressing strict structural constraints (complete graph) as a conjunction of terms, i.e., as a series of CRPQs. Relaxation of structural constraints (k-plex) can be expressed as a disjunction of CRPQs, where each single term represents a possible group pattern. This can be seen as a demonstration of the problem CSRPQ tries to solve, because with growing complexity and size of the queried groups, length and complexity of the SPARQL query increases exponentially. This is the reason the group size in baseline evaluation was set to 5 only, as bigger group sizes resulted in immersive queries impossible to be handled by the SPARQL evaluation engine.

The code we use for evaluation along with baseline query examples is publicly available[5], and can be compiled and executed to verify the published results.

3.8.2 Comparison to Baseline

Algorithm 1 is compared to the baseline algorithm by executing Query 1 and Query 2 for different input nodes and data sets. Each single dot in Figure 3.3 represents the execution on particular query instance, i.e., with fixed input node. The horizontal axis reflects the size of the search space that is relevant for executing the defined queries, i.e., accumulation of node degrees of input nodes' direct neighbors. Query evaluation times for our approach are shown as red triangles, valid query evaluations for the baseline are depicted as blue diamonds, and green '×' stand for instances, for which the baseline failed to deliver the result within 30 minutes. In such timeout cases we discontinued query evaluation.

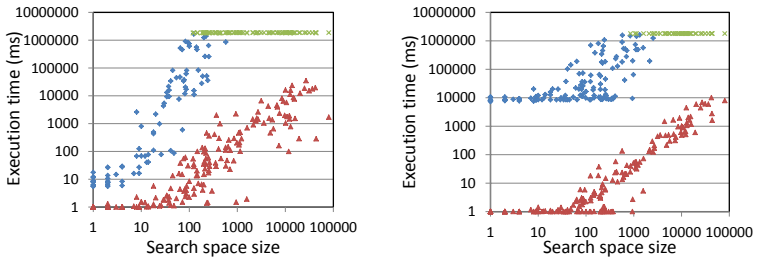
Figure 3.3 demonstrates that our approach outperforms the baseline in orders of magnitude. There are many timeouts for the baseline algorithm; in contrast, our algorithm is always able to return the query result without timeout violations. Thus, CSRPQ extension proves to be not only more expressive, but obviously also enables efficient search space pruning and traversal that marks already visited nodes, as discussed in the previous section.

3.8.3 Influences on Evaluation Time

In this section we illustrate how varying groups sizes influence the evaluation time of Algorithm 1, together with information about the corresponding sizes of result sets. In particular, Figure 3.4 depicts dependencies between the queried group size and resulting

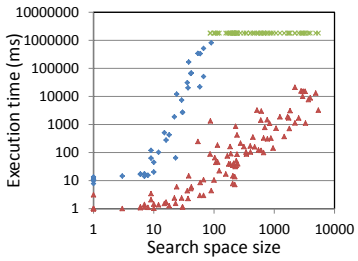
[4]http://jena.apache.org/

[5]http://www.infosys.tuwien.ac.at/prototypes/csrpq

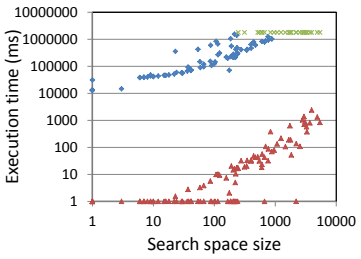


(a) Friendster Query 1

(b) Friendster Query 2



(c) Slashdot Query 1



(d) Slashdot Query 2

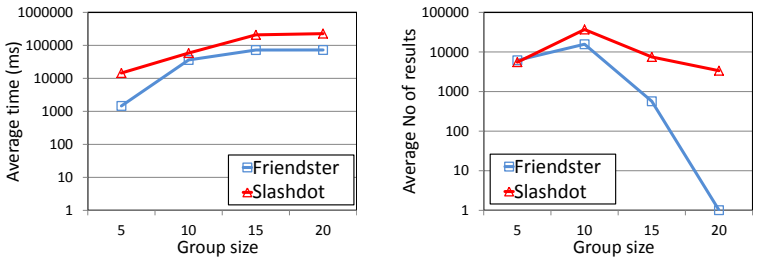
Figure 3.3: Performance of CSRPQ evaluation algorithm versus CRPQ SPARQL-based implementation. Each dot denotes a single query evaluation. ◆ and × symbols stand for regular and timed-out CRPQ baseline evaluations respectively, ▲ symbols denote CSRPQ-based evaluations.

average query evaluation times ((a) for Query 1 and (c) for Query 2), and between the group size and average number of results ((b) for Query 1 and (d) for Query 2).

The evaluation results show that the group size exponentially influences query evaluation time only up to some limit. We can see that group sizes greater than 10 do not result in substantial increase of evaluation time. As per design of our CSRPQ evaluation algorithm, further increasing of the group size will eventually lead to zero evaluation time as it will not be possible to find a single node with high enough degree, so that query evaluation can be stopped immediately with empty result set.

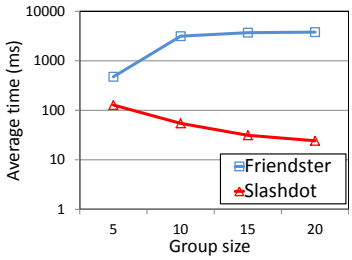
As for the amount of results, we notice that a big number of results guarantees high query evaluation time. But vice versa does not hold, i.e., absence of results does not guarantee fast evaluation. This confirms our observation that the key aspect of query evaluation time is the similarity of the underlying search space graph structure to the query graph structure. A big number of subgraphs structurally similar to the query graph, i.e., with small edit distances, would result in increased evaluation times.

Overall, the evaluation results prove the feasibility of Algorithm 1 for solving CSRPQs, even for large, real world data sets. It is much more efficient compared to solving a corresponding SPARQL query with a state of the art evaluation engine. Evaluation times of our algorithm depends on the query and the dataset, and how effectively pruning can be applied.

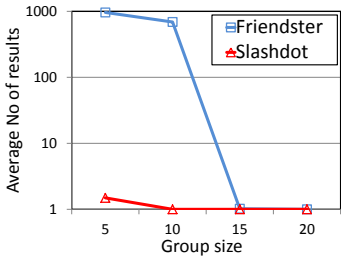


(a) Average runtime for Query 1.

(b) Average number of results for Query 1.



(c) Average runtime for Query 2.



(d) Average number of results for Query 2.

Figure 3.4: Influences of queried group size and size of result set on the evaluation time.

3.9 Conclusions

Being able to query groups based on structural properties and relations to other entities is an important asset for languages working on graph-structured data. However, to the best of our knowledge, there is no query language backed by an evaluation engine able to express such queries. In this chapter we presented CSRPQs, which extend CRPQs with such group selection capabilities. Being more expressive, CSRPQs also enable efficient evaluation techniques, outperforming thus CRPQ-based implementations. Experiments show that our CSRPQ-based implementation is capable of finding groups in orders of magnitude faster than the state-of-the-art CRPQ-based SPARQL query evaluation engine.

As discussed in Section 3.3, there exist many algorithms for selecting social formations exhibiting specific structural characteristics (e.g., clique, k-plex). Being more restrictive, these special-purpose algorithms might be more efficient than the general-purpose query evaluation algorithm at hand. By operating with the notion of a set, however, CSRPQs enable the evaluation algorithm to recognize special cases and fall back to the special-purpose algorithms when needed.

CHAPTER 4

Modeling Collaboration-assisted Computation

Modeling collaboration processes is a challenging task. Existing modeling approaches are not capable of expressing the unpredictable, non-routine nature of human collaboration, which is influenced by the social context of involved collaborators. We propose a modeling approach which considers collaboration processes as the evolution of a network of collaborative documents along with a social network of collaborators. Our modeling approach, accompanied by a graphical notation and formalization, allows to capture the influence of complex social structures formed by collaborators, and therefore facilitates such activities as the discovery of socially coherent teams, social hubs, or unbiased experts. We demonstrate the applicability and expressiveness of our approach and notation, and discuss their strengths and weaknesses.

4.1 Introduction

Business process modeling (BPM) allows companies to describe and document their enterprise processes. If captured accurately, such knowledge allows to analyze, improve, and execute those processes with higher efficiency. Although a variety of techniques and tools have been introduced for BPM, modeling of highly dynamic non-routine processes, such as human collaboration, is still a subject of discussion in research and very few approaches have been proposed so far [Nur08].

While collaboration in general means working together to achieve a goal [DD00, MM06], with the proliferation of collaboration software, such as groupware or wikis, the manner of human collaboration has taken the form of incremental contributions to a network of shared documents, e.g., source code files, wiki pages and so on. Relations between documents, actors, and other artifacts may influence the collaboration process. For example, some tasks should be done by actors chosen based on social relations, actions

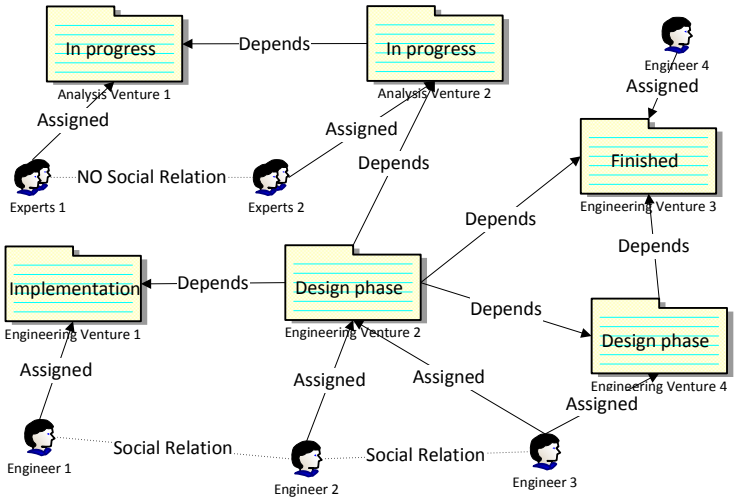


Figure 4.1: Software engineering collaboration process snapshot

on some documents should not be performed before related documents reach certain conditions, or a change in a related document might force to re-do an activity. Moreover, social structures formed by collaborators affect produced network of artifacts. Indeed, Conway's law suggests that "organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations" [Con68]. For example, socially coherent teams tend to produce more seamless solutions. Therefore, a proper modeling of collaboration processes must consider both semantic structures in networks of artifacts, as well as structural formations in social networks formed by collaborators. Although artifact-based process models have already been researched [San11a, vdAWG05, BHS09], existing modeling approaches do not emphasize the relations between artifacts and actors, and are not capable of capturing complex social structures formed by collaborators.

We thus propose a novel modeling approach and a graphical notation for collaboration processes. The key idea is to treat each document's evolution as an individual process that is explicitly influenced by the states of related documents and patterns in the surrounding social network. We propose to formalize the relations in line with the data from collaboration software, e.g., two developers can be considered related if they committed code to the same project folder in a source code repository. The amount of such data will grow with social computing pervading the enterprise IT[1], thus allowing process modelers to create richer models of people-intensive processes that support information-centric, bottom-up and context-aware and social modeling techniques for collaborative tasks.

The main research contributions of this chapter are (i) a novel approach for modeling context-aware social collaboration business processes, (ii) an expressive formalism that

[1] http://www.gartner.com/it/page.jsp?id=1470115

allows to define complex dependencies as network of artifacts and people, and (iii) a visual graphical modeling notation. The visual notation is a result of linking two threads of research in a novel way by combining graph query languages and control flow languages. Moreover, with the introduction of the notion of *groups*, this combination is further extended with fundamental concepts of social network analysis by allowing to express such advanced patterns as clique, k-plex, betweenness centrality, closeness centrality, structural equivalence and so on [Sco00].

The rest of this chapter is organized as follows: Section 4.2 describes the motivation behind the modeling approach and presents a motivating example. In Section 4.3 we show the lack of expressiveness in existing modeling approaches with regard to the scenario at hand. Section 4.4 describes the proposed modeling paradigm and the corresponding graphical notation. Section 4.5 demonstrates the usability of the approach through realistic use cases. Our modeling approach is critically discussed in Section 4.6. The chapter is concluded in Section 4.7.

4.2 Motivation

Collaboration is a recursive process comprised of human interactions towards realization of shared goals [DD00, MM06]. Groupware and social software foster collaboration of individuals who work across time, space, cultural and organizational boundaries, i.e., virtual teams [PPI04]. Using this type of software, people interact through conversations (e.g., e-mails and instant messages) and transactions (e.g., create/ modify/assign/restructure a document) in order to augment a common deliverable, e.g., the documentation of an idea, a technical specification, a source code file, or a wiki page. Typically, such interactions are disorganized, non-routine, and are hard to predict and model. However, as side-effects they produce semantical and social relations between actors and artifacts (e.g., *authorship*, *friendship*). Furthermore, artifacts are usually semantically connected into hierarchical or network structures, e.g., references in wiki pages, or dependencies between software components. Likewise, actors contributing to artifacts form complex social or communication formations, whose structure significantly influences collaboration processes and artifacts themselves. For example, given that a group of collaborators can be represented by a graph with edges denoting regular communication, a group forming a complete graph has more chances to produce a successful artifact(s), than a group forming a sparse graph with many isolates. Patterns of interest differ in artifact and social networks in the sense that structural patterns in artifact networks focus rather on types of relations and artifacts, and their states, while structural patterns in social networks focus on the density of edges by considering single type of relation, e.g., such social formations as clique, k-plex, and notions of structural equivalence, betweenness centrality (broker), and so on [Sco00].

As a motivating example, let us consider in-house software engineering in a dot-com company. Projects, or ventures, in such a company can be classified as engineering ventures (development of new functionality), or analysis ventures (incident investigation, proof-of-concepts). Both types of ventures produce deliverables, such as source code or

technical documentation. Figure 4.1 demonstrates a snapshot of a collaboration process as a directed graph of venture deliverables and collaborating actors.

Edges connecting ventures represent functional dependencies (i.e., a venture depends on either an investigation report or a software component produced by other ventures). Edges connecting actors depict social relations, i.e., there is a regular communication over instant messaging channels between them, or they contribute to the same venture. Contrarily, the edge `NO Social Relation` denotes absence of social ties, e.g. actors never worked on the same venture. Analysis ventures, representing rather creative and non-routine work, can reside only in two possible phases, namely `In Progress` and `Finished`, while engineering ventures, representing more structured and long-running work, can reside in more phases, such as `Design`, `Implementation`, `Testing`, and `Finished`.

Now, let us consider a process modeler that possesses knowledge of the working environment, the culture, and the scale of the company, and aims at modeling the following rules (we refer to them as context dependency rules (CDRs)):

1. *A venture project team should be notified of any changes in the technical documentation of other ventures it depends on. However, if two functionally interdependent ventures share any team members, then enforced communication is not required.* This rule ensures proper knowledge sharing between functionally interdependent ventures while avoiding overcommunication. For example, any new technical reports of `Analysis Venture 2` should be communicated to the project team of `Engineering Venture 2`. However, the same synchronization between `Engineering Venture 2` and `Engineering Venture 4` is not critical, because `Engineer 3` is anyway aware of any such changes.

2. *Venture technical documentation (i.e., design, or a report) should be reviewed by an expert from a functionally dependent venture, socially unrelated to the venture team members. Moreover, it might be necessary to find a group of such experts.* This rule tries to avoid biased reviews by finding socially unrelated experts. For example, it is more preferable to assign `Engineer 4`, than `Engineer 1`, as a reviewer of `Engineering Venture 2`, as `Engineer 4` does not have strong social relations with the `Engineering Venture 2` team.

3. *An engineering venture can be started if at least one venture, it depends on, has passed* `Design phase`. This rule defines a balance between total serialization of dependent ventures `Design phases`, which results in a longer time-to-market, and total parallelization of `Design phases`, which results in more iterations. For example, `Engineering Venture 2` was started upon completion of `Design phase` of either `Engineering Venture 3` or `Engineering Venture 1`.

4. `Design phase` *of a venture cannot be finished if any venture, it depends on, have not passed* `Design phase` yet. This rule minimizes chances of potential rework and wasted efforts. For example, `Design phase` of `Engineering Venture 2` can

be finished only after `Engineering Venture 4` switches to `Implementation phase`.

5. *If an engineering venture is in* `Implementation phase`, *and any of the engineering ventures it depends on has switched back to* `Design phase`, *then the venture should switch back to* `Design phase`. This rule covers possible redesign cases and ensures proper handling of late adjustments.

6. *For each analysis venture it is preferable to assemble a socially coherent team.* Given that a team can be represented as a graph, the extremum of social coherence is a clique, i.e., every two vertices in a graph are connected by an edge. In practice, however, cliques are seldom, and a modeler may want to relax constraints on social coherence by specifying a k-plex, where k-plex can be defined as a group of size n having each member connected to at least $n - k$ other members. Such coherent teams can be selected either for `Analysis Venture 1` or `Analysis Venture 2`. This rule tries to maximize communication within the team and good social atmosphere.

7. *If a software engineer stops working on an engineering venture during* `Implementation phase`, *it is necessary to replace her with a structural equivalent.* A structural equivalent is an engineer that has almost the same social neighborhood as the former engineer with respect to dependent project teams. For example, to replace `Engineer 2` it is preferable to find an engineer socially connected to both `Engineer 1` and `Engineer 3`. This rule aims at minimizing on-boarding time and restoring communication structures.

8. *If teams working on two interdependent ventures share no social connections, then it is necessary to find a liaison (broker), who is socially connected to more than 50% of members of each team.* For example, absence of social ties between groups `Experts 1` and `Experts 2` may hinder efficient communication thus delaying `Analysis Venture 2`. This rule tries to cover any structural holes in communication structures formed by collaborating teams.

As it can be seen from the examples above, CDRs allow to capture the knowledge about the impact of social and structural relations on collaboration processes. A formal specification can help to visualize and improve CDRs, thus reflecting management experience in an organization. We argue that a modeling approach, suitable for social collaboration processes, should encompass the following modeling principles that can be abstracted from the examples of CDRs above:

1. *Information-centric.* Collaboration processes should be represented by network of artifacts that originate from and evolve due to collaborative activities, following thus the information-centric perspective. Activity-oriented approaches are difficult to apply to collaboration processes, because it is hard to predefine exact steps to follow [Nur08]. For instance, people interactions, such as conversations and

transactions, in a collaboration process are rather unorganized and unpredictable, therefore, it is easier to capture collaboration artifacts and corresponding social and semantic relations as side effects of interactions. All the exemplified CDRs are based on the information-centric perspective on collaboration processes.

2. *Bottom-up and context-aware.* Modeling an evolvement of a network of artifacts in a holistic view can be a daunting task. Contrarily, neglecting relations completely and modeling the progress of artifacts in isolation leads to context tunneling [vdASW03], and, therefore ineffective models. A suitable modeling approach, therefore, should model the evolution of each artifact as an individual process explicitly influenced by its neighborhood (i.e., related artifacts), as it is shown in CDR examples 1 and 3-5. This approach allows to describe behavior at the macro level (network of artifacts) by means of modeling behaviors at the micro level (evolvement of a single artifact).

3. *Social.* Collaboration processes are influenced by social and communication structures formed by collaborators. Often, advanced non-routine activities are involved, such as discovery of socially coherent teams (CDR example 6) and structural equivalents (CDR example 7), or complex decision-making by exploiting social hubs (CDR example 8), and unbiased experts (CDR example 2). Therefore, the paradigm should promote not only the modeling of a network of evolving artifacts, but also of an evolving network of people. The modeling approach should be able to express not only social relationships between actors involved, but also complex patterns in social networks, such as k-plex, clique, structural equivalence, structural holes and so on.

Moreover, apart of incorporating the mentioned principles, the modeling approach should be backed up by a formal definition to support automatic reasoning, verification, and execution.

4.3 Related Work

In this section we discuss the related works with respect to modeling principles outlined in the previous section, and show their shortcomings with regard to their ability to model the exemplified CDRs. To the best of our knowledge, no existing framework is capable of capturing the CDRs defined in this work in a formal and visual manner. In the following, we will discuss Activity-oriented Business Process Modeling (Section 4.3.1), Artifact-centric Workflows and Case Handling (Section 4.3.2), Context-aware Workflows (Section 4.3.3), Visual Graph Query Languages (Section 4.3.4), and other approaches which influence our work (Section 4.3.5). In addition, Table 4.1 provides an overview of the general ability of the different related works to follow the modeling principles presented in Section 4.2.

4.3.1 Activity-oriented Business Process Modeling

Traditional activity-oriented business process modeling (BPM) approaches like the Business Process Modeling Notation (BPMN)[2] allow to model dependencies between processes via messages or events. Asynchronous messaging can be used to partially resemble CDRs, e.g., by sending notifications to related processes. However, it would not provide enough expressiveness and flexibility to capture such rules. Using external events is another way to model such logic, but, it would require the specification of events in natural language. Moreover, activity-oriented approaches are difficult to apply for collaboration processes, because it is hard to predefine exact steps to follow in collaborative workflows [Nur08]. In addition, explicit communication and coordination entities (i.e., events, message channels), intended for publishing information, do not convey any functional load and, therefore, complicate and encumber process models. Agent-based or agent-inspired approaches for coordination of business processes, such as [vdABEW00, HA99], also utilize explicit information publishing entities, thus sharing the same disadvantages.

The Web Services Business Process Execution Language (WS-BPEL or just BPEL) is an executable language standardized by OASIS[3], which allows to define business processes based on Web Services. This means that processes in BPEL export and import functionality by using Web Service interfaces exclusively [CGK+03]. Major IT companies realized the lack of human interaction support in service-oriented systems and proposed the WS-HumanTask [AAD+07] and BPEL4People [KKL+05] specifications. While these languages and their extensions (e.g., [SSP12]) allow for interaction with humans in the setting of an SOA, they are not designed to capture CDRs.

Table 4.1: Overview of Covered Aspects in the Related Work

	Information-centric	Bottom-up & Context-aware	Social
Activity-oriented BPM	☐	☐	☐
Artifact-centric Workflows	☑	☑	☐
Case Handling	☑	☑	☐
Context-aware Workflows	⊠	☑	☐
Visual Graph Query Languages	☐	☐	⊠
Multiagent Systems and Speech Acts	☐	☐	⊠

Legend: ☑ Supported ⊠ Partially Supported ☐ Not Supported

4.3.2 Artifact-centric Workflows and Case Handling

Information-centric modeling approaches, such as Artifact-centric workflows [Hul08, San11b] and Case Handling [vdAWG05], can capture the evolution of collaboration

[2] http://www.bpmn.org/
[3] http://www.oasis-open.org/

entities into formal models in order to provide a higher degree of flexibility than routing-based workflow descriptions are able to provide. Both approaches are examples of entity-centric modeling, which puts entities into the focus of processes and makes use of entity life cycles for dynamic modeling [San11b].

As the name implies, Artifact-centric workflows are based on (*business*) *artifacts* instead of the task-centric approach usually applied in business process modeling [Hul08]. Artifacts are important (business) objects which have a life cycle and provide information about their relationships to other artifacts as well as in what way and at what time tasks can be invoked on them [NC03]. Hull [Hul08] makes this more explicit by pointing out that artifact-centric workflows are not merely concerned with modeling process constructs and patterns, but take into account four explicit dimensions: the artifacts themselves, their (macro-level) life cycles, services ("tasks") running on artifacts, and associations/constraints.

For this, Artifact-centric workflows capture the relations on a conceptual level using Entity-Relationship models [BHS09], name-value pairs [NC03], or some Description [Hul08] or First-Order Logics [BHS09]. Life cycles are often depicted using some kind of finite state machines, but other approaches, like the Guard-Stage-Milestone meta-model for life cycles, which is based on Event-Condition-Action (ECA) rules, have also been introduced [Hul08, HDF$^+$10]. Most importantly, the association of services and artifacts can be done either in a *procedural* [NC03] (also: *imperative*) or *declarative* style [PvdA06, Hul08, FHS09, FMR$^+$09]. In contrast to the explicit modeling of process constructs as in, e.g., Petri nets, BPEL and BPMN, declarative languages (e.g., [PvdA06, BGH$^+$07, FHS09]) do focus on the goals of the process, i.e., *what* should be done instead of *how* it should be achieved [FMR$^+$09]. Example technologies to describe achievable goals using pre- and postconditions are the Ontology Web Language for Web Services (OWL-S) and the Web Service Modeling Ontology (WSMO) [FLP$^+$06, MBM$^+$07]. The *ConDec* language [PvdA06], which allows both an imperative and declarative modeling of business processes, makes use of Linear Temporal Logic to define declarative process constraints. Using an according model checker, it is possible to verify the correctness of processes and enact it by translating the process into an automaton. Comparable to our work, ConDec defines a graphical notation for such constraints. Due to the nature of the language, this notation is restricted to temporal constraints; social relationships are however not foreseen.

Case Handling distinguishes between the possibility to execute a business process fully automatic ("workflow management") and the necessity of human intervention during process runtime ("Case Handling") [vdAWG05], thus allowing a high degree of flexibility and variability. While the former is based on modeled process control structures, in the latter a *knowledge worker* is responsible for actively finding a way to reach the goal of a case. The Case Handling system is a dedicated assistant to the knowledge worker.

(Business) artifacts and cases are based on similar ideas, but cases are more focused on giving the structure and state of a case by data objects and therefore describing these objects in more detail [LBW07]. Data objects are also intended to represent pre- and postconditions. Cases are defined by the activities that need to be executed, data objects,

forms which provide activity-based views on data objects, actors, and roles grouping those actors [vdAWG05, vdASW03]. Associations between single activities are not explicitly modeled, but activities are attached to cases, and data objects are linked to activities. By defining mandatory and restricted data objects, conditions, and precedence relations for single activities, it is possible to model the process underlying a particular case. Conditions are based on data objects' states and values, and are bound to a particular activity. During design time, roles can be linked to both complex cases and activities; during runtime, concrete actors can be attached to a particular activity. With regard to the work at hand, the evolvement of collaboration entities is captured on a conceptual level using composite cases and 'is-a' relationships between roles.

To the best of our knowledge, *condition* elements in neither Artifact-centric workflows nor in Case Handling approaches do allow to specify CDRs. Conditions in Case Handling are defined as sets of bindings where a binding is a set of values for specific data objects. Therefore, it is not possible to define a condition which examines all the objects in a specific relation to the object at hand (CDR example 3), or to specify that *all* the related objects must reside in a specific state (CDR example 4). Conditions in Artifact-centric workflows may be specified in formulas written in First-Order Logic [BHS09]. However, the specification is restricted and does not allow to use quantifiers, which is crucial for expressing CDRs (e.g., CDR examples 3 or 4). Nevertheless, Artifact-centric workflows present an important foundation for our own work, as will be further discussed in Section 4.4.

4.3.3 Context-aware Workflows

Both Artifact-centric workflows and Case Handling are (amongst other reasons) motivated by the assumption that activity-oriented business process models do not capture the workflow context in enough detail and therefore may lead to inefficiencies [vdAWG05, Hul08]. Of course, this can be directly traced back to the fact that the modeling perspective in these approaches focuses on other aspects.

According to Dey [Dey01], "context is any information that can be used to characterize the situation of an entity". Following this definition, workflow context is any information that can be used to characterize the situation of a workflow. Accordingly, Rosemann and Recker define business process context as "The minimum set of variables containing all relevant information that impact the design and execution of a business process" [RR06].

As a similar notion is also underlying the work at hand, in particular regarding context data related to collaborations, it is worthy to discuss further approaches towards context-aware workflows with regard to the modeling of social collaboration processes. In general, context-aware workflow modeling approaches extend modeling and execution languages like BPMN and BPEL by the means to define context and make use of this knowledge in workflow execution. Very often, this is motivated by some specific domain, e.g., manufacturing [WKNL07, SSSA12] or e-health processes [AFG+12]. In the following, we will only discuss different approaches in the field which are important to the work at hand; for a thorough discussion we refer to the surveys by Baldauf et al. [BDR07] and Truong and Dustdar [TD09].

An early, Unified Modeling Language (UML) class diagram-inspired approach to a visual language for context-aware business process modeling has been introduced as part of the *Systemic Enterprise Architecture Methodology* (SEAM) [BW03]. As it is the goal of SEAM to support human reasoning, a formal reasoning framework is not provided and it is not possible to formally define rules. Instead, some very basic relationships inspired by the means to model composition and dependencies in UML class diagrams are provided. SEAM does not explicitly take care of collaboration between roles, i.e., it is not possible to model CDRs. Instead, different roles are related to each other through actions they are collaborating on.

Saidani and Nurcan [SN07, SN09] regard context-awareness in role-based business process modeling. They allow the definition of location-, time-, resource-, and organization-related context data aiming at the identification of fitting actors for the defined roles. Notably, the authors do allow to state social relationships between actors, but it is not possible to explicitly define CDRs. Comparable to the work at hand, the underlying context model is based on First-Order Logic. However, the authors do not make use of a formal definition of their modeling notation as it is provided in our work. Furthermore, queries are mentioned, but the topic is not discussed on a deep technical level. In general, Saidani and Nurcan do not focus on the actual modeling tasks and therefore do not provide a graphical modeling notation. Furthermore, like in SEAM, they follow a goal-oriented approach while in our work, we focus on information artifacts. Hence, their work should be rather seen as a complementary approach than as a foundation for our modeling notation.

Wieland et al. define context-aware workflows by advocating the augmentation of workflow modeling and execution with information about the physical world [WKNL07]. For this, context events, context queries, and context decisions (context-based transition conditions) are added to a workflow model, allowing to define and search context data as well as changes of the control flow. The authors make use of BPMN for process modeling and extend WS-BPEL 2.0 into *Context4BPEL*. An XML-based language is used to express context dependencies. Ardissono et al. present the *Context Aware Workflow Execution Environment* (CAWE), which is a complete Service-oriented Architecture extended by capabilities to achieve context awareness [AFG+12]. With regard to the work at hand, the context-based adaptation policies are the most interesting aspects of CAWE, as they allow to alter the flow of a workflow execution. These policies are modeled using declarative rules and based on boolean, context-based preconditions. Abstract activities are used in order to define the generic behavior of a task, thus resembling Artifact-centric workflow modeling languages and Case Handling as discussed above. Furthermore, the authors introduce dedicated models (Role Model, User Model, and Context Model), but do not take into account collaboration issues in them.

In theory, both the work by Wieland et al. and Ardissono et al. could be used as a foundation for drafting and implementing CDRs. However, both frameworks do not explicitly model collaboration dependencies. As a consequence, a resulting model would be somewhat confusingly extensive and therefore hardly intuitive to comprehend. Instead, we decided to draft an independent and therefore lightweight modeling notation as presented

in Section 4.4. The inclusion of explicit information about (social) collaboration allows a much more specific and therefore comprehensible modeling approach.

4.3.4 Visual Graph Query Languages

Conditions in CDRs can be intuitively represented as queries over graph-structured data. Over 25 years, graph query languages have been investigated for expressing graph patterns in various domains such as biological and transportation networks, Semantic Web and many others [Woo12a, AG08]. In recent years a number of graph query languages have been proposed also for the domain of social networks [RS09, DNDR09, BIJ02, EBGC09]. Graph query languages do not incorporate any control flow structures, being thus incapable of expressing (business) processes. We review, however, the expressiveness of various graph query languages with respect to complex structural formations.

Many prominent graph query languages, such as **G** [CMW87], GraphLog [CM90], Lorel [AQM+97], StruQL [FFLS00], UnQL [BFS00], NAGA [KSI+08], Cypher [Tea12] and SPARQL 1.1 [spa12], are based on *Conjunctive Regular Path Queries (CRPQs)*, which are in their turn based on *Conjunctive Queries (CQ)*. A simple example of CQ for finding persons who work on both artifacts Evaluation and Documentation is $x\ worksOn\ Evaluation \wedge x\ worksOn\ Documentation$. CRPQs extend CQs by allowing to query for nodes that are connected by a path satisfying a regular expression rather than relying solely on static paths. Being capable of expressing many graph patterns, CRPQs cannot capture groups of varying size or with inexact topology.

Among non-CRPQ languages greater variability can be observed with respect to graph patterns that can be expressed. For example, GraphQL [HS08] with repetition of graph motifs allows to define cycles and trees. PQL [Les], used for biological networks, can capture along nodes also their neighborhoods of a specified radius. BiQL [DNDR09] unifies nodes and edges, which makes it possible to capture even more complex patterns. In QGraph [BIJ02], a visual query language employed for social networks data mining tool Proximity [JN03], graph patterns can have numeric annotations, e.g., "Find all directors that had at least 2 movies each of them winning at least 3 awards". The Social Networks Query Language (SoQL) [RS09], is the only graph query language that can describe groups of flexible topology and varying size up to some degree. For example, it is possible to select a clique by specifying a condition on a group as depicted in Listing 4.1. This query language, however, is not capable to capture more advanced graph patterns, like k-plex. None of the discussed graph query languages has groups as first-class citizens, i.e., it is not possible to specify relations between two or more groups, nor are they designed for CDRs.

4.3.5 Other Noteworthy Approaches

Several frameworks have proposed for modeling functional or social relations between actors, for example DEMO [Die01], $I^\star$ [YM94], EKD-CMM [NB03], Speech acts [Win86, Kib06], or social commitments in multiagent systems [Sin99, CB06].

Listing 4.1: Selection of a Clique using SoQL

```
SELECT GROUP
       FROM GROUP(DISTINCT(X,Y,Z) IN G2)
       ...
WHERE
       ...
       ALL SUBGROUPS(U,V) IN G2 SATISFY
              (PATH(U TO V AS P1)
              COUNT(P1.edges.*)<=1)
```

Frameworks modeling functional relations between actors, such as $I^\star$ and EDK-CMM, typically rely on different pre-defined dependency models. $I^\star$ exploits intentional and strategic relationships among actors [YM94]. It supports a Strategic Dependency model for capturing dependencies among actors for a specific business process design. Dependencies can be due to tasks, resources, goals and soft-goals. Based on that, one can reason how to improve the process/activity. $I^\star$ enables functional relations among actors and it could be used to model the relationships between actors and artifacts but it does not really consider dynamic and social context in our scenario, such as actors are working in the same projects or the evolution of artifacts in connection to other ones. The enterprise model of EKD-CMM supports business processes built atop three main (sub)models: actor/role, role/activity and objects. It focuses on functional relations so static dependencies among actors and artifacts could be modeled. But it does not support social relations and context that links actors and artifacts in particular collaborative tasks. Furthermore, pre-defined actor/role and role/activity models are not well-suited for dynamic relations that we also support in our framework.

Speech acts and social commitments in multiagent systems could be used to model functional relations taken by actors in the form of high-level communication actions. They aim to support both human actors and intelligent software agents so specific languages for human cooperative tasks [Win86], communicative acts [Die01], or social commitment protocols [Sin99, CB06] have been introduced. But, they are mainly designed for modeling actions, via communication messages, between two individual collaborators. Thus, they do not support well high-level, context-aware interaction patterns among groups like our approach. For example, we could use them to model the request from an actor to another actor, but we could not use them to model the context in which several actors are working on the same document (artifact). Furthermore, they require precise modeling of semantics and action flows to be carried out by collaborators. Due to the inherent ad hoc nature of communication and interactions in collaborations, this would prevent us from modeling dynamic information-centric collaboration actions.

In [BW95], so-called *batch-tasks* were proposed to allow for a task that is executed for multiple workflow instances at the same time. Other similar approaches can be found in [CCPP95]. Some simple CDRs can be covered by batch-tasks, e.g., CDR example 4. For more complex rules, however, this approach is not flexible enough, e.g., because

they do not consider artifacts. Team Automata [Ell97, EG02] use communication via shared action spaces. Transitions, which include the same external action, are fired simultaneously in these Automata. Alike to batch-tasks, it does not provide the needed flexibility.

While not directly related to our work, the PENELOPE (Process ENtailment from the ELicitation of Obligations and Permissions) language [GV06] allows to define timing constraints in a manner that could be helpful to define CDR examples 3-5. Interestingly, PENELOPE also allows to automatically generate a state space from the defined timing constraints – this feature has not been foreseen in our work, but would be an interesting aspect for the future work. Finally, the COREPRO modeling framework [MRH07] proposes to model the dependencies between states of related processes via so-called external state transitions. Again, it provides limited expressiveness for describing the dependencies, as it allows to specify only exact external state transitions.

4.4 Modeling Paradigm

As discussed in the previous section, related works provide some interesting links and foundations, but none of them provides a holistic approach encompassing all modeling principles and CDR examples discussed in Section 4.2.

4.4.1 Modeling Framework

Our modeling framework is defined as a set of basic modeling elements that a business process modeler can operate with in order to reflect CDRs within business process models:

1. *Collaboration artifacts* and their *states*. Artifacts should represent various aspects and deliverables of collaboration process (e.g., a software component, or a technical design). The states should represent the possible phases of collaborations. *Artifacts* and their *states* may be modeled using existing information-centric approaches, such as Artifact-centric workflows [BHS09], making thus our approach rather complementary, than stand-alone. Actors in the modeling framework are modeled as collaboration artifacts as well. This unification simplifies modeling of collaboration processes, as it is easier to predict types (or roles) of involved actors and their states rather than possible actions that comprise collaboration.

2. *Relations*. Relations can be pre-defined (e.g., functional or structural dependency) or dynamic (e.g., temporal or social relations), i.e., produced as side effects of interactions and transactions. Proliferation of groupware and social software boosts the quantity and quality of dynamic relations data, thus empowering process modelers.

3. *Groups*. Groups can be defined as sets of *artifacts* or people shaped by *relations* into formations exhibiting complex structural characteristics.

4. *Context-aware state transitions.* Context-aware state transitions define what *Relations*, *Artifacts* and *Groups* are relevant for a business process at various steps of its execution.

In order to better demonstrate how the framework's basic modeling elements can be put together to model a business process, we present a graphical notation for the modeling framework. The notation is an extension of the conventional statecharts visual formalism [DH87]. The choice of statecharts is justified by their information-centric nature and widespread adoption as part of UML[4]. Being a natural visual representation of the state machine mathematical model, statecharts include the following basic elements: (i) Clustered and refined *states*; (ii) State *transitions* comprised of *events* (external happenings such as user input or timeout), *conditions* (boolean expressions over events and state) and *actions* (e.g., sending an e-mail, or assigning a person to a task).

Our graphical notation, dealing with explicit modeling of relations, extends conventional statecharts with a new element `Context`, graphically depicted as a hexagon. A `Context` element, being inseparable to a `State` element, defines relations and artifacts, relevant to a particular state. Each `Context` element contains a specification of the neighborhood of the artifact (i.e., related artifacts and people) describing the presence of a specific pattern. Essentially, a `Context` element is a formalization of statecharts' *conditions*, usually expressed in free-text. Each `Context` can have several `Transition` elements attached: If the pattern, described in the context specification, is found in the neighborhood, then all transitions attached to the respective `Context` element are enabled, otherwise disabled. Similarly to `State` elements in statecharts, `Context` elements can be clustered using logical AND/OR/XOR operations.

Figure 4.2 demonstrates the overall integration of `Context` element into statecharts (the context specifications are omitted in this figure for the sake of simplicity). Two of three transitions in the figure are enabled by `Context` elements. By default, we assume that transitions attached to `Context` elements have a higher priority over other transitions, but generally it is up to a modeler to define the priorities. Below are enlisted possible transitions in the default prioritization order:

1. If `Event 1` is fired and a pattern described in `Context 1` is found, then the state machine switches to state `B`.

2. If `Event 1` is fired and a pattern described in `Context 1` is not found, then the state machine switches to state `C`.

3. If a pattern described in `Context 2` is found, then the state machine switches to state `D`. Here we can see that an *event* element is optional, and if absent, then the *transition* is activated at once.

When modeling the behavior of multiple interdependent concurrent process instances, a modeler should assume that state transitions are synchronized, i.e., *every* `Context`

[4] `http://www.omg.org/spec/UML/`

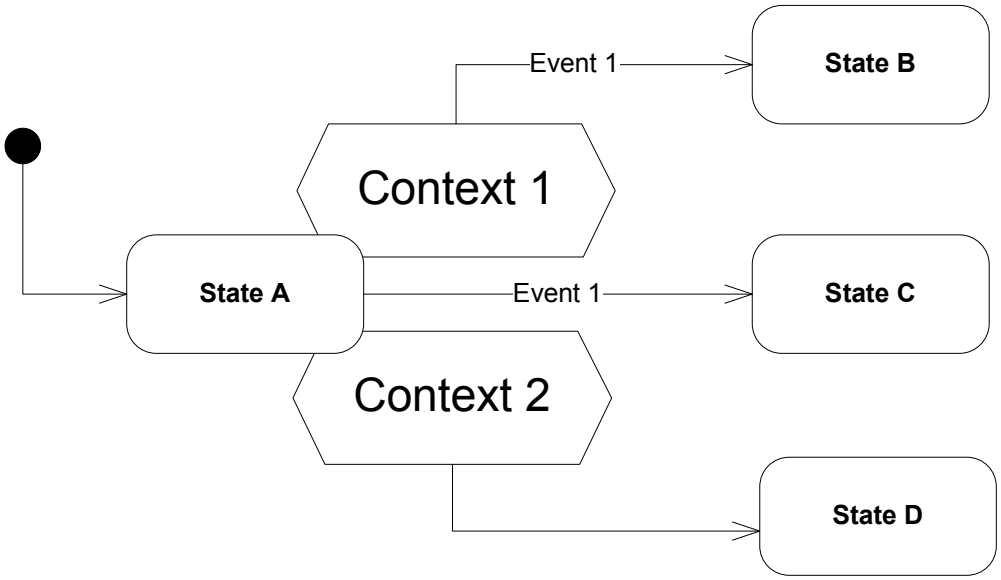


Figure 4.2: Integration of `Context` elements into statecharts

element is evaluated before activation of *any* state transition in any process. Thus, if a process switches to state `A` and then instantly to some other state, the fact that it has been in state `A` will be considered.

We believe that graphs *a priori* are rather a natural visual medium for describing artifact networks and relations. Therefore, we define a visual graph query language, which is used to define neighborhood specifications in `Context` elements. A query in the visual language is a directed connected multigraph with labeled edges and nodes. Labels can either denote atomic relations/states/types, or expressions over atomic entities based on propositional calculus expressions. Additionally, labels may be absent in general, denoting a placeholder (e.g., any relation/state/type). An edge direction in a graph is used to depict a non-commutative relation. Query graphs always have one initialized primary element, therefore, graph queries should be interpreted outwards: starting from the central primary element towards most distant nodes. For example, context specifications depicted in Figure 4.3 can be interpreted as follows:

- `Context 1`: If the primary document is in state `A`, and there are no documents, related by content or author to the primary one, residing either in state `A` or state `B`, then the attached transition is enabled.

- `Context 2`: If the primary document is in state `A`, and every single document, related by content to the primary one, must reside in state `B` and have two socially unrelated Authors that contributed to it, one of which is `Active`, then the attached transition is enabled.

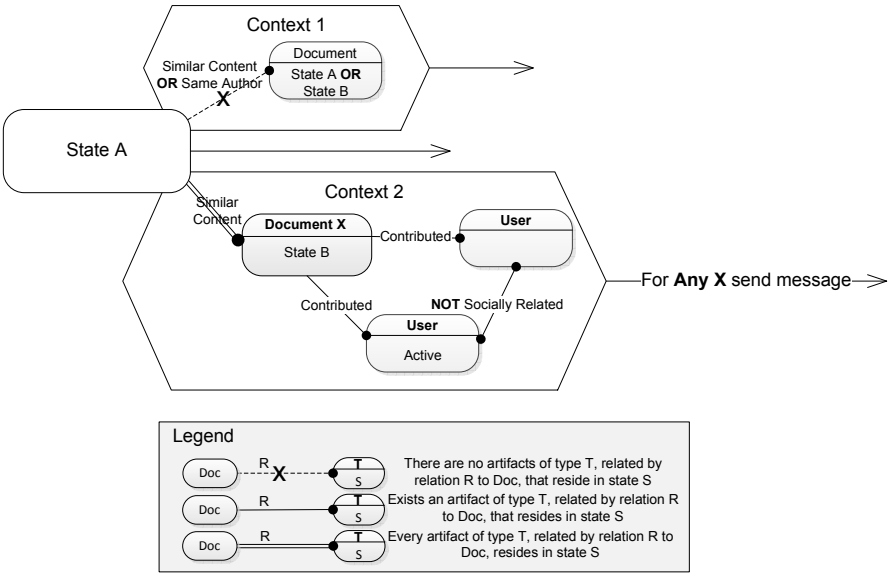


Figure 4.3: Example of context specifications in `Context` elements

As depicted in Figure 4.3, single line edges correspond to existence quantifiers, while double line and crossed dashed edges correspond to universal quantifiers. Nodes in query graphs may be labeled with variables that can later be reused in Conditions and Activities of corresponding Transitions. Since multiple occurrences of a context pattern may be found in the neighborhood, Activities/Conditions may be also extended with quantifiers, i.e., send e-mail to any/every related contributor. Interpretation of the exemplified graph query naturally corresponds to the way we read *First-Order Logic* expressions. *First-Order Logic* with its subsets form a solid foundation for many modeling frameworks and query languages. For our purposes we, however, introduce two extensions that allow for expressing CDRs as introduced in Section 4.2.

First, we introduce counting and fractional quantifiers that may annotate double line edges. Fractional quantifiers define ratio, while counting quantifiers define exact number of artifacts satisfying given condition. Counting and fractional quantifiers should always be defined with comparison operators $\{\geqslant, \leqslant, =\}$, which define, respectively, *at least*, *at most*, and *exactly* conditions. For example, double line edge in `Context 2` in Figure 4.3 annotated with fractional quantifier $\geqslant 70\%$ would be interpreted as "...at least 70% of documents, related by content or author...".

Universal and existential quantifiers can be considered as special cases of fractional and counting quantifiers respectively, i.e, at least 100% artifacts and at least 1 artifact. Since counting quantifiers are inherently similar to existential quantifiers, one might think of having only one type of edges. We, however, for the sake of simplicity and clarity, consider double line edges to correspond to plural existence of artifacts, while single line edges to singular existence. Similarly, we introduce crossed dashed edges as a special case of double line edges with $= 0$ countable quantifier.

Second, as a means of modeling complex structural formations we introduce the `Group` element, which defines a set of artifacts or people. `Group` elements naturally extend graph query notation exemplified so far in the same way as *Monadic Second-Order Logic (MSOL)* extends *First-Order Logic*. *MSOL* allows only existential quantifiers to be applied to set variables. Likewise, groups in our notation can only be defined existentially. Therefore, quantifiers, correspondent to single line, double line and crossed dashed edges adjacent to `Group` element, are applied to elements of the corresponding group, but not to the group itself. Similarly, state and type labels annotating `Group` element are also applied to group members (e.g., all group members reside in state `A`). Double line edges connecting two `Group` elements may be annotated with two quantifiers as they are adjacent to two plural entities. Along with set variables *MSOL* introduces the atomic formula $t \in S$, where t is a first-order term and S is a set variable. Considering practical usefulness of this formula, we introduce an additional edge type, which defines membership. The shape of such edges resembles aggregation association in UML, and the meaning of this type of edges is similar to a weak "has a" relationship.

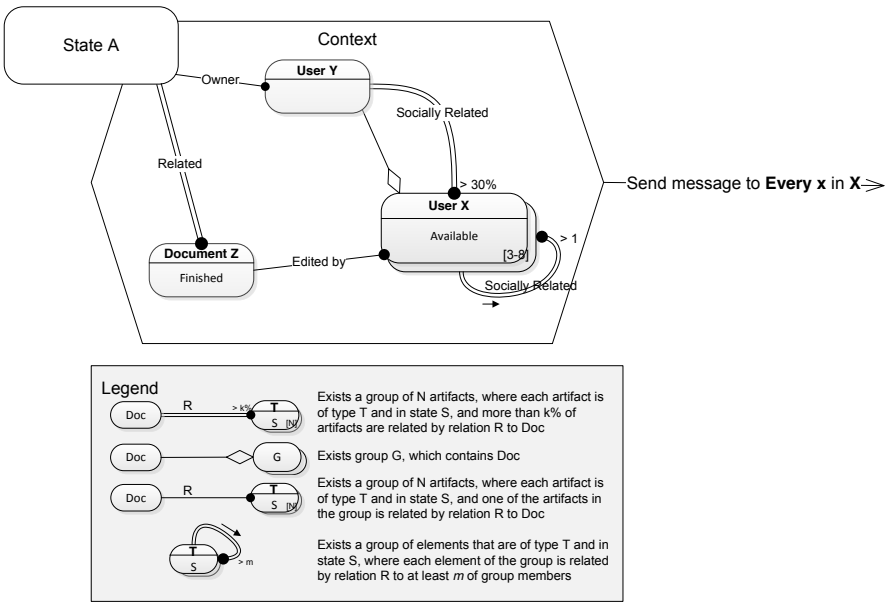


Figure 4.4: Example of using `Group` elements

Let us consider the example depicted in Figure 4.4, which shows the selection of a group exhibiting certain structural characteristics. The query described by the example can be interpreted as a conjunction of the following three statements:

- Exists a group `X` of size 3 to 8 members, such that every member of the group is of type `User`, and in state `Available`, and is socially related to at least 2 other group members.

- Every document `Z` related to the primary one is in state `Finished` and was edited by a member of the group.
- An owner of the primary document is a member of the group `X`, and is socially related to at least 30% of the group members.

Interpretation of the exemplified graph query naturally corresponds to the way we read *Monadic Second-Order Logic* expressions. For simplicity, during interpretation it is necessary to define groups before the edges adjacent to them. Quantifiers annotating a loop edge should be interpreted in the order specified by a small arrow attached to the edge.

The success of a modeling approach depends, to a great extent, on the level of simplicity offered. Therefore, we favor simplicity over completeness and impose the following constraints on the queries expressed in the visual language:

- Unlike artifacts, which can be defined with universal quantification by double line edges, groups can be only defined with existentional quantification. Being more expressive, universal quantification for groups is rather complex to comprehend.
- Only basic operators from proposition calculus are allowed as literal expressions attached to edges and nodes: conjunction, disjunction and negation. Even though, conditional and biconditional operators may be expressed via the former ones, more complex operators may decrease understanding and make reasoning about the model more difficult.
- Under the Open World Assumption [Rei87], negation may introduce ambiguity, therefore only negation as a failure is allowed, i.e., negation on an edge can be used only if nodes connected by the edge are transitively connected to the central node with non-negative edges.
- During our experiments with the modeling notation we observed that edges with universal (fractional) quantification adjacent to `Artifact` elements may introduce ambiguity in query graphs with cycles. We can define node level as a length of the shortest path from the node to the primary element. If a double line edge is part of a cycle, then one of its adjacent `Artifact` nodes should have the lowest level among the nodes in the cycle. In other words, since query graphs are interpreted outwards starting with the primary element, we can say that double line edges adjacent to `Artifacts` can appear only in those places where they can be interpreted first among edges in the cycle. This rule is not applied to edges with universal quantification adjacent to `Group` elements, as groups are always defined with existential quantification avoiding thus any ambiguities.

4.4.2 Formal Definition

A formal definition of our modeling notation is given below. In order to keep the definition succinct, we omit a formal definition of statecharts, as it is available elsewhere, e.g., in

[LMM99]. The formal definition supports automatic reasoning about consistency and correctness of a process at design time, e.g., detection context specifications that can never be reached. Moreover, it enables various optimization techniques, e.g., conversion of a workflow definition to a more compact and simple one.

Labels L in a query graph representing relations R, types T and states S of artifacts are defined as:

$$\begin{aligned} &\textit{Label } L \stackrel{\textbf{def}}{=} \textit{Atomic Condition} \mid \textit{Placeholder} \\ &\qquad \mid L \ \wedge \ L \mid L \ \vee \ L \mid \neg L, \\ &\qquad \textit{Placeholder} \text{ denotes } \textbf{any} \text{ value (no condition)} \end{aligned} \tag{4.1}$$

Edges in a query graph can have one or two attached quantifiers. Beyond standard quantifiers $\forall$ and $\exists$ the modeling notation allows generalized quantifiers, similar as proposed in [Mos57]. All extended quantifiers refer to a certain set, which is a whole domain of discourse or a single group in case of `Artifact` or `Group` elements respectively. In the following, we define the quantifiers we use by showing the mapping they signify with relation to some arbitrary set M.

$$\begin{aligned} &\textit{Universal} : \forall_M = \{M\} \\ &\textit{Existential} : \exists_M = \{S \subseteq M : S \neq \varnothing\} \\ &\textit{Counting} : \exists_M(\odot n) = \{S \subseteq M : |S| \odot n\}, \\ &\qquad\qquad \odot \in \{>, \geqslant, =, \leqslant, <\}, n \in \mathbb{N} \\ &\textit{Fractional} : \exists_M(\odot p\%) = \{S \subseteq M : |S| \odot p|M|/100\}, \\ &\qquad\qquad \odot \in \{>, <\}, p \in [0, 100] \end{aligned} \tag{4.2}$$

We use capital Greek letters Ξ and Ψ as placeholders for universal, counting, and fractional quantifiers, i.e., $\Xi, \Psi \in \{\forall, \exists(\odot n), \exists(\odot p\%)\}$.

Edges in a query graph, along with adjacent `Artifact` elements, are interpreted in First-Order Logic, extended with generalized quantifiers, as follows:

$$(a) \ \overset{R}{\multimap\!\bullet} \ (T, S) \stackrel{\textbf{def}}{=} \exists x : R(a, x) \wedge T(x) \wedge S(x) \tag{4.3}$$

$$(a) \ \overset{R}{\Rightarrow\!\bullet} \ \Xi \, (T, S) \stackrel{\textbf{def}}{=} \Xi x : R(a, x) \wedge T(x) \rightarrow S(x) \tag{4.4}$$

$$(a) \ \overset{R}{\text{-}\!\times\!\bullet} \ (T, S) \stackrel{\textbf{def}}{=} \nexists x : R(a, x) \wedge T(x) \wedge S(x) \tag{4.5}$$

Where, given that graph queries are interpreted outwards from the central primary element (vertex), a denotes an already interpreted vertex. Predicates T and S describe type and state of suitable artifacts respectively. The result of a query graph interpretation is a logical conjunction of the First-Order Logic formulas corresponding to graph edges. Higher priority of edges with universal quantification ensure that the formulas corresponding to these edges always appear at the beginning of the resulting logical conjunction. For double line edges $\forall$ quantification should be assumed as a default one, i.e., an absence of quantifier annotation is interpreted as $\forall$ quantifier.

Query graph nodes representing `Group` elements can be interpreted as follows:

$$((T,S))_{[m-n]} \stackrel{\textbf{def}}{=} \exists G : \forall g(g \in G \rightarrow T(g) \wedge S(g)) \\ \wedge\, n < |G| < m \tag{4.6}$$

$[m-n]$ annotation defines constraints on the group size, where both n and m can be optional defining thus absence of upper or lower limits. We use capital letters for variables identifying sets (e.g., G), and lower case for variables denoting single objects (e.g., g).

Edges in a query graph, adjacent to `Group` elements, are interpreted in Monadic Second-Order Logic, extended with generalized quantifiers, as follows:

$$(a) \stackrel{R}{\multimap\!\bullet} ((G)) \stackrel{\textbf{def}}{=} \exists g \in G : R(a,g) \tag{4.7}$$

$$((A)) \stackrel{R}{\multimap\!\bullet} ((G)) \stackrel{\textbf{def}}{=} \exists g \in G, \exists a \in A : R(a,g) \tag{4.8}$$

$$(a) \stackrel{R}{\Longrightarrow\!\bullet} \Xi((G)) \stackrel{\textbf{def}}{=} \Xi g \in G : R(a,g) \tag{4.9}$$

$$((A))\Psi \stackrel{R}{\Longrightarrow\!\bullet} \Xi((G)) \stackrel{\textbf{def}}{=} \Xi g \in G, \Psi a \in A : R(a,g) \tag{4.10}$$

$$(a) \stackrel{R}{\text{-}\times\!\bullet} ((G)) \stackrel{\textbf{def}}{=} R(a,g) \rightarrow g \notin G \tag{4.11}$$

$$((A)) \stackrel{R}{\text{-}\times\!\bullet} ((G)) \stackrel{\textbf{def}}{=} R(a,g) \rightarrow a \notin A \vee g \notin G \tag{4.12}$$

Here, similarly to previous definition, a enclosed into single and A enclosed into double round brackets denote an already interpreted artifact or group respectively. Also, $\forall$ is a default quantifier for double line edges.

In addition to edges defined above, membership and subgroup relations in a graph query can be defined as:

$$(a) \multimap\!\diamondsuit ((G)) \stackrel{\textbf{def}}{=} \exists G : a \in G \tag{4.13}$$

$$((A)) \multimap\!\diamondsuit ((G)) \stackrel{\textbf{def}}{=} \exists G : A \subset G \tag{4.14}$$

Query graph Q is a triple defined as follows:

$$\begin{aligned} Q \stackrel{\textbf{def}}{=}\ & (a, E, V), \text{graph } Q \text{ is connected,} \\ & a \text{ is the predefined central primary vertex (artifact),} \\ & V \text{ is a set of vertices } (T,S) \text{ and } ((T,S))_{[m-n]}, a \notin V, \\ & E \text{ is a set of edges } E \subseteq \{a\} \times \{\multimap\!\bullet,\ \Longrightarrow\!\bullet,\ \text{-}\times\!\bullet\} \times (V) \\ & \quad \cup\, V \times \{\multimap\!\bullet,\ \Longrightarrow\!\bullet,\ \text{-}\times\!\bullet\} \times V \end{aligned} \tag{4.15}$$

Context element CTX in the modeling notation is a composition of query graphs CQ:

$$\begin{aligned} CQ \stackrel{\mathbf{def}}{=} & Q \mid CQ' \textbf{ AND } CQ'' \mid CQ' \textbf{ OR } CQ'' \\ & \mid CQ' \textbf{ XOR } CQ'', \\ & CQ' = (a', E', V'), \\ & CQ'' = (a'', E'', V''), \\ & a' = a'', E' \cap E'' = \varnothing, \\ & V' \cap V'' = \varnothing \end{aligned} \tag{4.16}$$

Transition element CT, attached to Context element CTX, can be defined as:

$$\begin{aligned} CT \stackrel{\mathbf{def}}{=} & (CTX, E, C, AC), \\ & E \text{ is an external event}, \\ & C \text{ is a condition}, C : QU \times ID \rightarrow \{\textbf{true}, \textbf{false}\}, \\ & AC \text{ is an activity}, AC : QU \times ID \rightarrow \varnothing, \\ & ID \text{ is a set of identifiers attached} \\ & \text{to vertices in } CTX \text{ graph}, \\ & QU \text{ is a set of quantifiers}, QU = \{\textbf{Any}, \textbf{Every}, \textbf{All}\} \end{aligned} \tag{4.17}$$

In the next section we demonstrate the expressiveness of the defined modeling notation by means of several use cases.

4.5 Evaluation

This section describes four collaboration process use cases which demonstrate the application of our modeling approach to various collaboration issues. As it can be seen, the approach allows to easily express the dependency of a process on complex relations in its environment, and to compactly capture the dynamic co-influence between instances of the same process in one model. For clarity, in the use cases we attach to each `Context` element a free text description of its specification.

4.5.1 Use Case – Design Game

Goal. The goal in this use case is to coordinate a design of a complex system consisting of interrelated projects. A set of expert virtual teams thus collaborate to reach a consensus. The assignment relation between teams and projects is one-to-one, but teams can share members. As some projects are dependent, it can happen that changes in the design of one project can be the reason for changes in the design of other ones. Finally, all project designs should be consistent with their dependent ones.

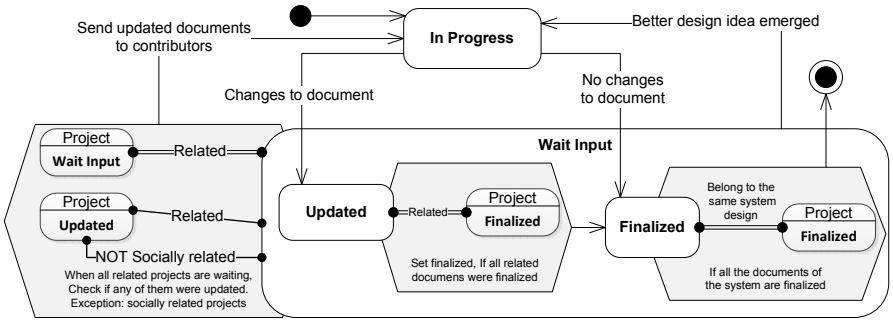


Figure 4.5: Use case – design game

Model. Each project of this system is regarded as a separate process (see Figure 4.5). In the beginning, it is in `In Progress` state, indicating that the team is currently working on its design. When the team makes some changes to the design and commits it, the process goes into `Updated` state. If no changes to the design were made, i.e., the existing version was examined and considered valid, then the process switches to `Finalized` state. The states `Updated` and `Finalized` together represent superstate `Wait Input`, which means that the project design is currently awaiting for some external actions. If the team suddenly decides to update the design (e.g., a better idea emerged), the process goes back into `In Progress` state.

Now, if the process is in `Wait Input` state, and if all the related projects are also in `Wait input` state and at least one is `Updated`, then the team should check the design of their project against inconsistencies with updated projects. Thus, the updated documents are sent to the team and the state is switched to `In Progress`. An exception is the case when the project team shares a common expert with the team of an updated project (relation *Socially related*), who is expected to foresee any inconsistencies beforehand. Waiting the related projects to be in `Wait Input` ensures that all the updates of related documents will be taken into account.

When in `Updated` state, and if all the related projects are finalized, the process goes into the finalized state, which ensures that if a document spawned no updates among related documents, it will not stay in `Updated` state.

The system may be considered in the final state when all the projects are in `Finalized` state.

Advantages. This use case demonstrates the modeling of collaboration as ordered iterative communication of project teams towards reaching a consensus. It shows that our modeling approach, as opposed to existing modeling approaches (see Section 4.3), is capable of expressing universal and existential quantification.

4.5.2 Use Case – Social Selection

Goal. The goal of this use case is to support a software development process with the selection of appropriate actors (e.g., developer, adviser, reviewer) based on relations with

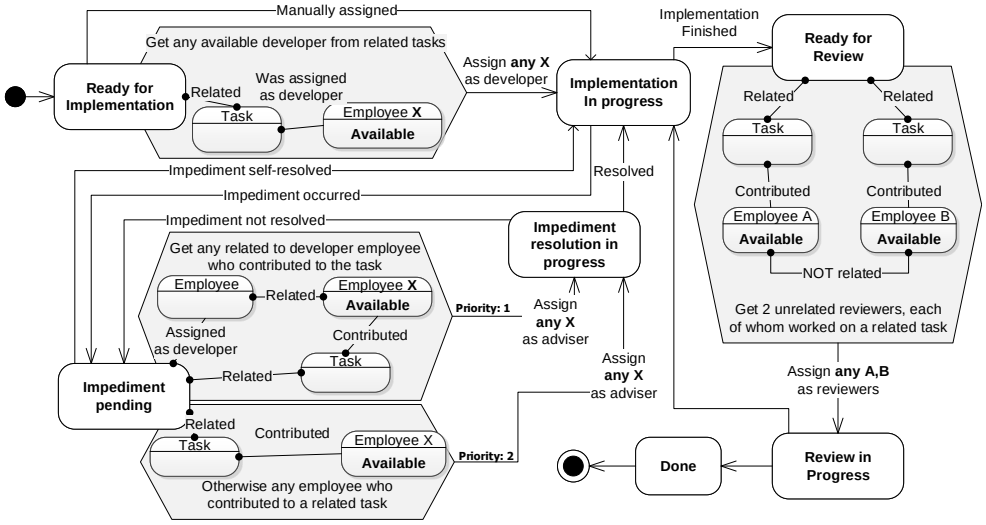


Figure 4.6: Use case – social selection

the other tasks and among the actors. Tasks are related if they belong to the same project, employees are related if they collaborated before.

Model. Figure 4.6 depicts the software development process. At first, the task is in the `Ready for Implementation` state and is waiting for an appropriate developer to be assigned. Any available developer from a related task is assigned for this role, as he/she is expected to be more productive because of being familiar with some related concepts. Alternatively, a manual assignment is performed. In either case, the process goes to the `Implementation in Progress` state. An impediment can occur during the implementation (`Impediment pending` state), in which case an adviser is needed for assistance. An adviser is preferably selected as being related to the developer employee who contributed to a related task, because of joint work experience. Otherwise, any related task contributor is chosen. If the adviser is found, the process goes into `Resolution in Progress` state, from where it can either go either back to `Implementation in Progress` or `Impediment Pending` states, depending on whether the impediment has been resolved. Also, the developer can resolve the impediment by herself if no adviser was found. After the implementation is finished, the reviewers are selected (`Ready For Review` state): they are desired to have experience with related tasks but be unrelated to each other, which assures unbiased reviews. After the review process (`Review In Progress` state), either the implementation needs to be revised, or the task is considered finished.

Advantages. This use case demonstrates expressiveness of the modeling approach when visualizing a social network environment, allowing thus to model processes that require discovery (e.g., compose a socially coherent team), unbiasedness (e.g., involve independent people), and negotiation (e.g., by exploiting of social hubs). It shows expressiveness of the graphical notation with regards to modeling discovery in a surrounding

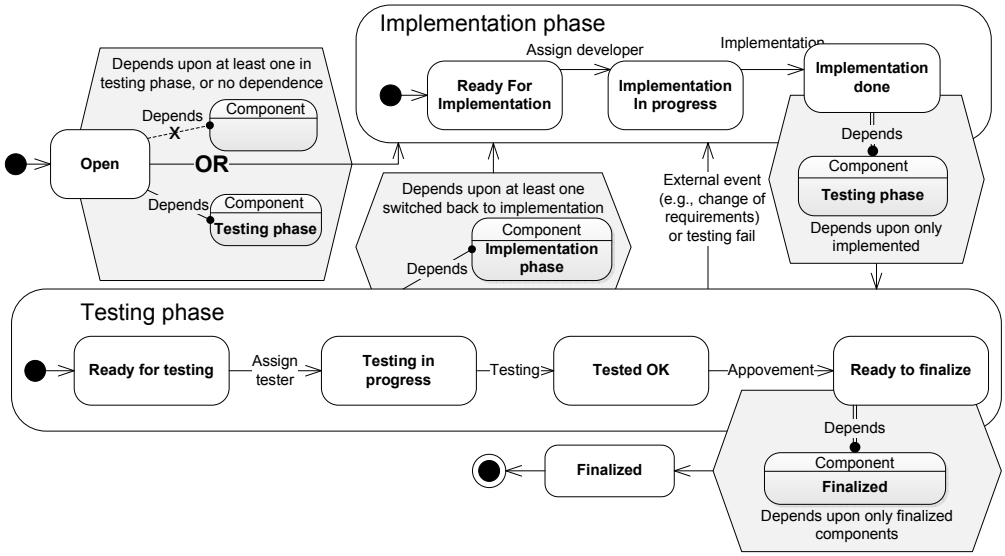


Figure 4.7: Use case – dependent components

social network. Contrarily, existing modeling approaches fall short of expressing such patterns in a visual and formal manner (see Section 4.3).

4.5.3 Use Case – Dependent Components

Goal. The goal is to coordinate the development and testing of a software product, which consists of manifold components, some of which depend on others (we assume no cyclic dependencies). The development a component should proceed only when the components it depends on have reached certain progress.

Model. Figure 4.7 depicts the process which corresponds to a single component. It starts in `Open` state and switches over to `Implementation Phase` in either of two cases: it does not depend on any components, or at least one component which it depends on is in `Testing Phase`. This ensures some minimal basis for the development. After `Implementation Phase`, the component is ready to switch over to `Testing Phase`, but, first, it should wait for all the components it depends on to be implemented, so the testing covers the combined functionality. The testing phase can reveal some flaws so the component will return into `Implementation Phase` for fixing those. If, while the component is in `Testing Phase`, any of the components it depends on suddenly goes into `Implementation Phase`, then the testing should be stopped in order not to waste the testing effort on outdated components. Lastly, if the component is in `Ready to Finalize` state, and all the components it depends on are `Finalized`, then the component can be finalized.

Advantages. This use case demonstrates the suitability of the modeling approach for expressing the coordination of project teams towards ensuring consistency and correctness of a complex product. It shows the expressiveness of our modeling notation if comparing

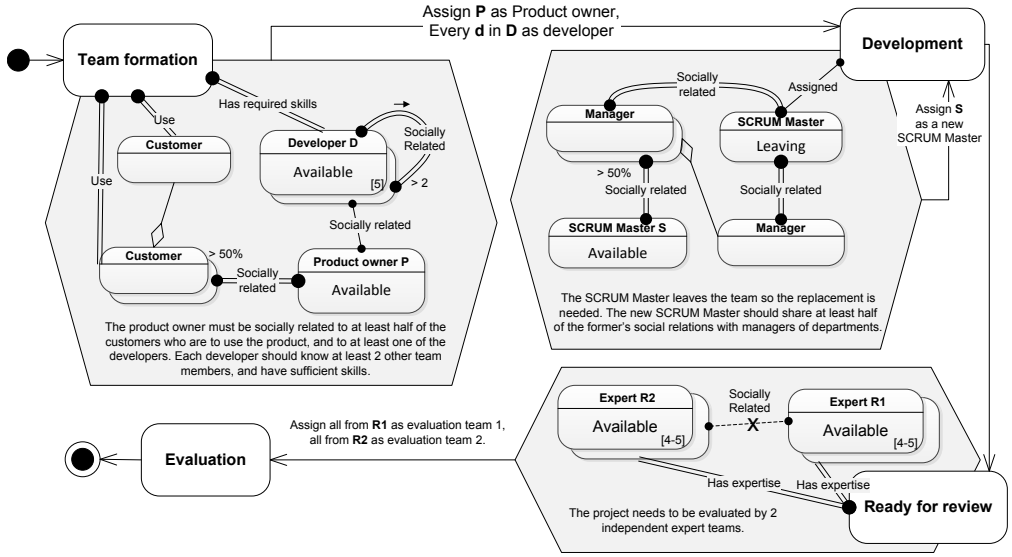


Figure 4.8: Use case – teams and groups

it to existing modeling approaches that would capture process coordination either in a text form or via events (see Section 4.3).

4.5.4 Use Case – Teams and Groups

Goal. The goal of this use case is to compose effective teams based on social connections and internal company structure. This use case exemplifies a composition of a development team, a replacement search for a key role (here: SCRUM Master[5]), and the formation of independent expert groups.

Model. Figure 4.8 shows a simplified software development process with focus on team creation and support. The process starts in `Team Formation` state where a development team of five people and a product owner are chosen. The product owner should be socially related to at least half of the future users of the product at hand, which ensures more efficient communication of requirements and feedback. The product owner should also know at least one of the developers, so a better contact with the team can be established. Developers in turn should have the skills necessary for the project and should know each other to some extent for easier integration, so the rule states that each developer in a team should be related to at least two others.

To specify the requirement that *all* the customers that will use the product must be included in the group, we need to use an advanced pattern, because having only a double edge from the primary element is insufficient: that would mean that there *exists* a group of customers, all of whom will use a product, but does not imply that all such possible customers will fall into this group. We thus need to use an additional single `Customer`

[5] `http://scrummethodology.com/scrum-effort-estimation-and-story-points/`

element defined with a universal quantification, which means that each customer who uses the product is included in the group. The double edge from the primary element should remain to state that the group is restricted only to the customers who will use the product. The similar technique is used in the context with SCRUM Masters.

Once the team is formed, the development phase starts and the process goes into `Development` state. During this phase it might happen that some member leaves a team for an arbitrary reason, and a replacement has to be found. The use case illustrates such a situation with the team's SCRUM Master. A criterion for the new SCRUM Master is that she should share at least half of connections of the leaving SCRUM Master to managers of collaborating departments. This rule aims to retain the pace of issue resolution, should any occur while collaborating with the other teams. After the development phase is over, the process goes into `Ready for review` state where evaluation teams are composed. Two teams of four to five people should be independent, i.e., a member of one team should not have any connections to the members of the other team, to assure unbiased evaluation. The process can then switch to `Evaluation` phase and eventually be finished.

Advantages. This use case demonstrates the capabilities of the framework to express advanced patterns in social networks, such as 2-plex (development team), broker (product owner), and structural equivalence (new SCRUM Master), as well as conditions involving multiple teams.

4.6 Discussion

Compared with existing approach, the main strengths of our modeling notation are its expressivity and flexibility. First, the modeling notation is capable of capturing complex graph patterns inherent to social networks. Second, the modeling notation goes in line with statecharts by avoiding any domain-specific constructs, making it applicable outside of the social networks domain. Third, it allows to capture the evolution of a network of artifacts, as well as a network of people. Our modeling approach is supported by a formal definition, enabling thus design time reasoning, verification, optimization and efficient execution.

The absence of explicit communication entities (events or messages) in the modeling approach is a strength regarding the clarity of the resulting model, but also a weakness. It allows to provide simple processes coordination and secure encapsulation: a process can modify only its own state, it cannot impact related processes explicitly, similarly to Cellular Automata (CA) [Neu66]. However, a modeler cannot immediately see what parts of a business process (states) other processes rely upon. Given that definitions of events and messages represent a process interface, a modeler will not be able to remove or change process states without a certain risk of affecting other models. However, this problem can be remedied with state clustering available in statecharts.

Unlike CRPQ-based languages (see Section 4.3.4), our visual notation does not have notion of paths defined with regular expressions. Seamless integration of paths requires further investigation with respect to usefulness in the scope of context-aware processes,

and is part of our future work. Also, we envision that other additional elements might be added, like aggregation operators to describe the accumulated state of the entire neighborhood. Moreover, among our major interests are the possibility of sharing context elements between parallel processes along with zoom in and zoom out capabilities for *group* elements.

Our modeling framework unifies active and passive entities, i.e., actors and artifacts, and considers them from the perspective of classification (type) and possible states. Correspondingly, our graphical modeling notation employs a single type of shapes for both actors and artifacts in line with statecharts. This approach emphasizes the viewpoint of groupware and collaborative software, where actors are represented simply as user profiles, which are, essentially, also documents. The unification affects slightly the intuitiveness of the modeling notation, as it is not immediately visible which entities are active and which are passive. However, this unification allows for greater flexibility, e.g., it is possible to specify an actor as a central element, modeling thus evolvement of a user profile, or express a semantically coherent group of artifacts. Moreover, it enables a broader use of the framework and potential application to other domains. For example, it is possible to introduce actors that represent software agents, or other types of social entities, such as organizations.

According to the formal definition (see Section 4.4), our modeling notation incorporates *relation* as a modeling element, but neither types nor semantics of relations are formalized. This makes sociality of the modeling notation somewhat implicit, coming rather from the ability to express common patterns in social networks and their influence on collaboration processes. Being highly dependent on the target domain, semantics of relations between collaborators are left to be defined by the modeler, as well as possible problems to infer those relations. Absence of specific semantics behind relations, again, allows for greater flexibility, enabling a modeler to define and adjust many specific types of relations, such as colleagues, acquaintances, relations denoting mutual dislike or past conflicts and so on. To avoid this, a context taxonomy [SN07, RRFA06] could be extended to incorporate information about different social relationships.

4.7 Conclusion

This chapter proposes a modeling approach and a corresponding graphical notation for creative human collaboration processes. The applicability of the approach was demonstrated through several use cases, and its strengths and weaknesses were discussed.

Comparing to existing approaches, our contribution has two main distinguishable features: it is capable of capturing complex patterns in network of artifacts and people, and it advocates a communication model where a process can modify only its own state and cannot explicitly impact related processes. We have shown that these features are naturally suitable for modeling of social collaboration processes. Although our approach was designed with this focus, we do not exclude its applicability in other areas.

In the next chapter, we first present an execution framework for our modeling approach in the form of a coordination language.

CHAPTER 5

Implementing Collaboration-assisted Computation

Today people work together across time, space, cultural and organizational boundaries. To simplify and automate the work, collaboration employs a broad range of tools, such as project management software, groupware, social networking services, or wikis. For a collaboration to be effective, the actions of collaborators need to be properly coordinated, which requires taking into account social, structural, and semantic relations among actors and processes involved. This information is not usually available from a single source, but is spread across collaboration systems and tools. Providing a unified access to this data allows not only to establish a complete picture of the collaboration environment, but also to automate the coordination decision making by specifying formal rules that reflect social and semantic context effects on the ongoing collaboration processes. In this chapter we present Statelets, a coordination framework and language for support and coordination of collaboration processes spanning multiple groupware tools and social networking sites, and demonstrate its suitability in several use cases.

5.1 Introduction

Groupware and social software foster collaboration of individuals who work across time, space, cultural and organizational boundaries, i.e., virtual teams [PPI04]. Problem of people coordination in collaborative processes has been already extensively studied in academia, (e.g., in [Dus04, FBG96]), and addressed in industry with ever more groupware products incorporating workflow and orchestration mechanisms (e.g., Microsoft Sharepoint). However, in many cases, people interact and contribute in divergent commercial or non-profit on-line collaboration platforms, such as social networks, open

source development platforms, or discussion forums, that remain decoupled, isolated and specific to their domains. The problem of coordination in such a setup gets a new look, where processes that need to be coordinated are decentralized and distributed across different specialized tools and online services.

Social network context is an integral part of human coordination. For example, the following context aspects have an impact on the behavior of collaborating individuals: actions taken by neighbors in social network [GGJ+10], social neighbors' preferences [Chw00], and the social network structure itself [ZB11]. The degree of the impact varies from network context simply 'carrying' the information that can be used in a process to forcing adjustment or even cancellation of ongoing actions. Also, social context can imply mutual dependency between processes, reflected by such common coordination mechanisms in social networks as collective actions [Chw00], i.e., 'I'll go if you go'. Social network context can be used for such advanced activities as expertise location [MA98], composition of socially coherent collaborative teams [DB11], discovery of unbiased reviewers, and so on.

Along with the social component of the network context, semantic relations between processes may affect coordination decisions as well. In groupware and wiki-like platforms, processes are reflected as incremental changes of common deliverables (e.g., documentation of an idea, a technical specification, or a source code file) connected into dependency and semantic networks. Relations between these artifacts may influence the collaboration process. For example, actions on a document should not be performed before related documents reach a certain condition, or a change in a related document might force to re-do an activity.

Due to an information-centric nature of both social and semantic contexts, we combine these notions together and define *network context of a collaboration process* as *information about related processes and people, their actions and states.* In our previous work [LKTD12a] we discussed network context effects on collaboration processes, and presented an approach for modeling them.

In spite of growing interest to social network effects in academia [Chw00, GGJ+10, ZB11], the problem of network context-based coordination has not been properly addressed by coordination languages and frameworks. As examined in the chapter, existing coordination languages lack necessary features to enable efficient programming of coordination based on network effects. We refer here to suitability as an amount of efforts a developer needs to spend to express such coordination rules. Also, supplying the developer with social and semantic network context requires horizontal composition of groupware and social networking sites, which imposes yet additional challenges [DG11, KCSS10], which are not addressed properly by existing frameworks as well.

In this chapter we present Statelets, a programming language for coordination of social collaboration processes spanning multiple software systems. A distinguishing characteristic of Statelets is the support for coordination based on social and semantic network effects. Although the primary focus of the chapter is the programming language, our contribution also includes a conceptual architecture of the underlying framework that aims at integration of groupware and social networks to extract social and semantic

contexts. To evaluate Statelets, we have implemented use cases that show its advantages and suitability to the domain.

The rest of this chapter is structured as follows: Section 5.2 provides a motivating example and identifies the features that are crucial for a network context-based coordination language. In Section 5.3 we explore the suitability of existing coordination languages for the problem at hand in the perspective of these features. Sections 5.4 and 5.5 describe the Statelets coordination language and the conceptual architecture of the underlying framework respectively. Section 5.6 demonstrates the usage of the language with use cases. The chapter is concluded in Section 5.7.

5.2 Motivation

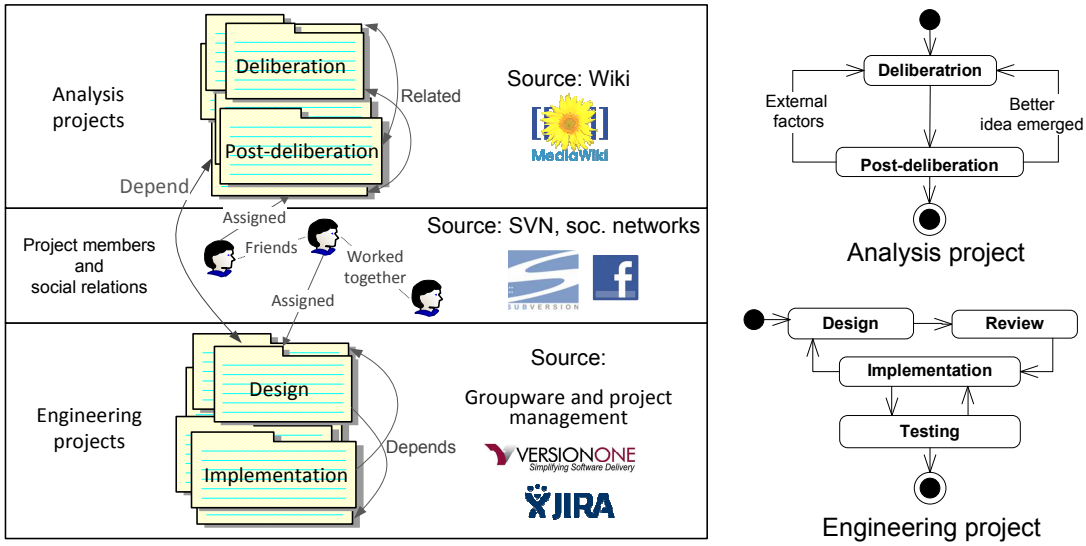
As a motivating example, let us consider open source software engineering. Projects in software engineering can be classified into analysis projects and engineering projects (See Fig. 5.1b). An analysis project represents a non-routine and changeable process, whereas an engineering project represents a rather routine and stable process. Both types of projects produce deliverables, such as source code or technical documentation. Projects get assigned to members of open source communities, who are located via social (professional) networks and online collaboration services, and are then hold responsible for the progress of corresponding activities.

Projects can be related to or depend on each other. For example, two projects are related if they contribute to the same software product, are functionally interdependent, or share components, goals, or resources. Similarly, social and professional relations and technical dependencies exist between project members, e.g., a software engineer depends on engineers who wrote previous versions of the component or worked on the code in the past. Figure 5.1a depicts various relations between projects and their members.

The key to success of such engineering and analysis projects are advanced activities, such as expertise and resource discovery. Such activities are not possible without integration of professional (e.g., XING, LinkedIn) and private (e.g., Facebook, MySpace) social networks, and online collaboration tools (e.g., SourceForge). Figure 5.1a depicts integration and execution environment of processes that correspond to analysis and engineering projects. Engineering projects are more specific to the domain, and, therefore, require more specific groupware, e.g., VersionOne, or Jira. Analysis projects, on the contrary, require more flexible and wide-spread groupware, such as MediaWiki (engine for Wikipedia).

Given the setup described above, let us consider the following possible coordination rules:

1. *If an Analysis project is in* `Post-Deliberation phase`*, and all its related Analysis projects have transitioned to* `Post-Deliberation phase`*, then, if any changes have occured among solutions in those projects during the transition, the project should be switched back to* `Deliberation phase` *and the changes should be communicated to the project's team.* This rule ensures proper communication



(a) Integrated relations between projects and actors

(b) Project types

Figure 5.1: Projects in open source software engineering

of new or adjusted solutions between teams of interrelated Analysis projects and allows a collaboration team to produce solutions that are not affected by possibly incorrect solutions produced by other teams. Similar strategies were adopted in agile software engineering methodologies, e.g., in SCRUM estimation game[1].

2. *An engineering project design should be reviewed by an expert from a functionally dependent project. Moreover, it is preferable to assign an expert socially unrelated to the project team members.* This rule tries to avoid biased reviews by finding socially unrelated experts.

3. *In case of an expertise request, an appropriate expert should be socially connected to one of the project team members, or work on a related project.* This rule ensures faster expert onboarding.

4. *When starting an engineering project, a socially coherent team of qualified experts should be assembled, which has connections to members of related projects.* This rule tries to maximize probability of a project success by ensuring a good social environment in advance.

5. *An engineering project can be started, if at least one project it depends on has passed* `Design phase`. This rule defines a balance between total serialization of dependent projects `Design phases`, which results in a longer time-to-market, and total parallelization of `Design phases`, which results in more iterations.

[1] http://scrummethodology.com/scrum-effort-estimation-and-story-points/

6. `Design phase` *of a project cannot be finished until all projects it depends on pass* `Design phase`. This rule minimizes chances of potential rework and wasted efforts.

7. *If an engineering project is in* `Implementation phase`, *and any of the projects it depends on has switched back to* `Design phase`, *then the project should switch back to* `Design phase`. This rule covers possible redesign cases and ensures proper handling of late adjustments.

8. *All impediments in a project should be communicated to any engineer in every related project.* This rule ensures timely communication between project teams.

Let us consider the challenges that a developer faces when implementing the aforementioned rules. Based on the challenges, we further draw conclusions and identify the most important features that reflect the effectiveness of a coordination language and its underlying framework.

1. *Optimized horizontal integration of external collaboration projects.* The motivation scenario involves integration of many social networks and groupware products, such as MediaWiki, Subversion, LinkedIn, Facebook, and VersionOne. The developer should concentrate on the coordination logic, and not on how to extract the needed information from external sources. As APIs of collaboration platforms could not provide all the needed information in the right form, the framework needs to decouple the concepts perceived by the developer from representation and transformation issues and take care of the optimizing the data exchange seamlessly for the developer. Different authorization mechanisms and the necessity for identity mapping between entities coming from different sources makes integration even more complex. The coordination language should in turn support the unconditioned access to externally provided data in a manner that enables the optimization, and the language's semantics should reflect the nature of external APIs, i.e., consider distinct behavioral classes of APIs' methods (e.g., methods with and without side-effects).

2. *Condition-Action rules.* Rules 5, 6, and 7 take the declarative *condition-action* form, as opposed to more common *event-condition-action* rules, because the developer is interested in situations or patterns that need to be managed rather than in events that lead to these situations. When a condition depends on external data sources, problems of continuous checking and polling arise. Additionally, when a condition depends on time (e.g., escalation), timers get involved as well. These problems should be abstracted away from the developer and be handled by the framework, while the coordination language should support *condition-action* expressivity.

3. *Network context querying and processing.* Integration of groupware and social software enables social resource discovery and process coordination based on rich network context. Manipulations with network context, as it can be seen from most

of the rules above, can be significantly simplified with quantifiers (as in Rules 5, 6, or 8), and disjunctions (as in Rule 3), as they naturally fit for expressing the coordination logic.

4. *Network context synchronization.* As depicted by Rule 1, when multiple related entities fulfill a rule, the action should be taken for all such entities simultaneously to avoid a situation when the action for one entity discards the condition for other related entities. Such synchronization issues should be handled at the framework level and be taken into account by the language design.

5.3 Related Work

In this section we examine existing coordination and orchestration languages with respect to the features outlined in the previous section. Table 5.1 summarizes the suitability of considered languages to network context-based coordination. The suitability of a language can be characterized as the amount of efforts a developer needs to spend on a task at hand. We therefore regard the native support of the aforementioned features, i.e., when no additional effort is needed for their realization.

Table 5.1: Natively supported features in selected coordination languages

	Seamless integration	Condition-Action	Context queries	Context synchronization
Control-driven languages	⊠	☐	☐	☐
Linda-based languages	☐	☑	☑	☐
Reactors	☐	☑	☑	☐
CEP languages	☐	☑	☑	☐
BPEL4Data (BEDL)	☐	☑	☐	☐

Legend: ☑ Supported ⊠ Partially Supported ☐ Not Supported

Control-driven coordination and orchestration (workflow) languages based on messages (channels), such as BPEL [AAB+05], Orc [KCM06], or Workflow Prolog [GP07] are specifically designed for integration of services like those in external APIs. They can also simulate network effects via messages or events, i.e., by notifying related processes. However, context querying using point-to-point messages would result in "chatty" communication, and context synchronization would require the implementation of complex protocols similar to two-phase commit. Also, support for integration is limited, as difference between methods w/o side-effects is not considered.

Data-driven (Linda-like [GC92]) coordination languages (for example, [BFLM01]) express coordination as dependencies between removal/reading and insertion of atoms from or into a shared space. However, groupware APIs are often assymetric and do not provide *insert/remove* operations for each *read* operation. It is therefore hard to

align API method calls with the removal and insertion of atoms, because the actual changes made by API calls are not explicit and occur rather as side-effects. Basic Linda operators provide only limited expressivity of conditions expressing network context, unlike reactive Linda extensions [BZ05] that introduce additional *notify* operation. Two coordination approaches are used in reactive extensions [BZ05]: parallel (e.g., JavaSpaces, WCL) and prioritized (e.g., MARS, TuCSoN (ReSpecT)). In order to express network context synchronization, parallel reactions require implementing two-phase synchronization protocols, similarly to control-driven languages. Prioritized extensions make the implementation even more difficult by restricting the usage of coordination operators within reactions.

Reactors [FMS09] is a coordination language where networks of reactors can be defined by means of relations. The behavior of reactors in the neighborhood is observed as sequences of their states, which can be queried with Datalog-based language, thus allowing the context querying. Also, Reactors eliminate the distinction between events and conditions. Reactors react to stimuli defined as insertion or removal of relations. This is suitable for integrating RESTful APIs, but is limited to them, as many groupware APIs are coarse-grained and it is not intuitive to map insertion and removal of tuples to API calls. In general, reactors are executed concurrently and independently. Synchronous execution can only be achieved through a composition of reactors, which is not intuitive to implement.

Given that processes can publish their states as events, modern Complex Event Processing languages (e.g., [PE10]) can express conditions on network context using event correlation and predicates. However, representation of external data retrieved from request-response web APIs in the form of events is not intuitive. Moreover, the recursiveness [MM06] (See Rule 1) of collaboration processes can significantly complicate the definition of network context queries.

Typically, rules in Rule-based languages fire non-deterministically, thus complicating the network context synchronization. However, two notably different approaches here are: (i) to derive dependencies from postconditions (e.g., [SC05]), which in scope of external APIs integration might be not known, or not possible to define; and (ii) by explicit operators (e.g., [NN08]), which do not allow to specify dependencies based on relations between events.

In XML-based language BPEL4Data [NKM$^+$10] processes can communicate via shared business entities, resembling thus a shared-space paradigm. Business entities are represented as XML documents. Simple conditions can be expressed as guards on Business entities using XPath/XQuery. However, it is not intuitive to describe network context querying, i.e., conditions on a graph of related XML documents. Synchronization between processes is achieved through additional processes and locks. Similarly to CEP languages, integration with BEDL requires representation of external data changes in the form of CRUDE notifications or invocations, which is not always intuitive.

As it can be seen, existing approaches partially support requirements outlined in the motivating scenario, but none of them provides a full spectrum of features necessary for efficient programming of coordination based on network effects.

5.4 Statelets Coordination Language

In this section we present Statelets, the coordination language designed for orchestration of activities in groupware and social software systems. The language natively supports all four features outlined in the previous section. However, native support of the *'Seamless integration'* feature requires additionally implementation of an extensible framework, conceptual design of which is discussed in the next section. The main building block of Statelets is *statelet* - a construct that corresponds to a state of a process and denotes coordination rules that should be fulfilled when the process resides in this state. Statelets do not completely describe collaboration processes, but rather are complementary reactions to workflows defined in groupware systems and human collaboration activities. A statelet consists of mainly two parts: a condition(s) that formally describes an anticipated situation and an action(s) which have to be undertaken if such a situation is detected. Conditions are given in a form of *context queries* against the data integrated from external collaboration projects, and the actions are given as either *triggers* that correspond to external commands in collaboration software, or *yield* constructs that activate other statelets. All the data integrated from external sources by the framework is accessed as *relations* in the language. This allows the developer to easily design the coordination rules by seamlessly combining the relations originating from diverse platforms into single conditions.

5.4.1 Context Queries and Commands

Assymetric nature of many collaborative software APIs is reflected in Statelets as segregation of operations[2] to read (side-effect free) operations, i.e., queries, and modify operations, i.e., commands. Such segregation allows a programmer to specify what API methods are side-effect free and what are not, enabling thus the framework to treat them differently.

Queries. Read operations define data models, which in Statelets are represented as a unified hypergraph comprised of overlay networks. Even though the data model is defined by collaboration software adapters, additional relations may be integrated, (e.g., Core Relations Library), denoting side-effect free external computation. For example, querying the *Factorial(X, Factorial)* relation results in computation of a factorial by an integrated component. Also, additional virtual relations can be defined on top of the basic data model. For instance, a SocialRelation virtual relation below is defined by means of relations coming from Facebook and MySpace.

Listing 5.1: Definition of social relation

```
relation SocialRelation(User1, User2):
 Facebook.Friends(User1, User2) || MySpace.Friends(User1, User2);
```

Querying a hypergraph relation at runtime creates a *data stream*, i.e., a lazy sequence of records, which is gradually initialized by the framework with each set of vertexes matching

[2] http://martinfowler.com/bliki/CQRS.html

the given relation found. Given that relations in hypergraph constitute predicates, data streams can be formed by expressions using the following binary operators based on the First-order logic:

- Operators `&&`, `||`, `not`, and `->` correspond appropriately to $\wedge$, $\vee$, $\neg$, and $\rightarrow$ first-order logic connectives with implicit existential quantification attached to all variables within the expression.
- Operators `=>`, `-!`, and `-x` correspond to conditional ($\rightarrow$) connector with implicit universal quantification over the variables present in the left part of the expression. Variables in the right part of the expression, that are not present in the left part, are quantified as $\exists$, $\exists!$, and $\neg\exists$ appropriately. Clearly, second and third operators can be expressed using the first one.

Basically, a query expression describes a pattern (a subgraph) within a hypergraph. Appropriately, a data stream resulted from evaluation of this query contains all occurrences of the pattern.

Queries in Statelets can be evaluated using `define` and `wait` operations:

- `define` operation simply evaluates a query expression and searches shared space hypergraph for pattern instances. Each pattern instance found along the hypergraph search is pushed into the data stream. If no instances are found, then `define` returns an empty data stream.
- `wait` operation continuously evaluates a query expression until at least one pattern instance is found. Therefore, `wait` operation always returns non-empty data stream.

For instance, if it is necessary to wait until all related to the project documents are completed, then we can use the following code snippet:

Listing 5.2: Queries in Statelets

```
wait Related(Project, Document) => Status(Document, 'Completed');
```

Here a data stream is created that remains uninitialized until the condition is satisfied. However, if it is simply necessary to check if all related documents are completed, then the following code snippet can be used:

Listing 5.3: Relations in Statelets

```
define Related(Project, Documents) => Status(Document, 'Completed');
```

Here an uninitialized data stream is created, which either is initialized with all related documents if all of them are completed, or is initialized as empty. A statelet can run many queries, getting thus many data streams. If query expressions within a statelet share variables, then resulting streams are joined by those shared variables.

Commands. Commands represent groupware API methods with side effects, for example, send an e-mail, or delete a document. Commands in Statelets are executed using `trigger` keyword:

Listing 5.4: Commands in Statelets

```
trigger AssignReviewer(Document, Reviewer);
```

Commands in Statelets are used to process or handle records of data streams defined by query evaluations. If a data stream is not yet initialized, then a command is suspended until it is initialized (similar to lists with unbound dataflow tail [VRH04]). However, if a data stream is empty, then the command is not executed at all. A command can be executed for `any` or for `every` record in a data stream, or for the whole collection of records. `Any` quantifier is a default quantifier, which is implicitly attached if no quantifiers are specified. Consider the example below:

Listing 5.5: Commands with quantifiers

```
trigger SendForReview(every Team, any Programmer, all Documents);
```

This reads as follows: send a list of Documents (all Documents) for a review to any Programmer in every Team.

5.4.2 Programming Coordination

Coordination is managing dependencies between activities. Apart of being able to express basic dependencies between human activities, Statelets also support network context-based coordination.

Dependencies between Activities. A statelet by itself describes precedence dependency: once completion of a human activity is registered in a shared space, a succeeding activity is triggered by a command. Statelets can be composed using `alternative` keyword expressing thus multiple different outcomes of a manual or automated activity. We exemplify usage of such composition in the use case scenarios. The statelet in the example below describes dependencies between design activity, project owner notification activity, and assignment of multiple experts activity:

Listing 5.6: Design phase

```
statelet DesignPhase(Project):
{
 wait DesignDocument(Project, Document) && Status(Document, 'Completed');
 trigger NotifyProjectOwner(Project);
 define ExpertiseKeywords(Document, Keyword) && FindEngineers(Expert, Keyword);
 trigger Assign(every Keyword, any Expert, Project);
};
```

Dependencies between processes. A process in Statelets is comprised of a sequence of statelets that produce each other by using `yield new` operation, i.e., a sequence of states. A process may reside in multiple orthogonal states, requiring thus presence of many statelets in parallel. Therefore, a statelet is technically a coroutine: it can produce multiple new statelets along its execution. Statelet by itself complements shared space hypergraph at runtime, simulating thus a relation. In other words, a statelet can query existence of other statelets in its neighborhood similarly to how it queries for existence of specific relations and nodes in a shared space hypergraph. A process

in Statelets thus communicates with its neighborhood by changing its own state. In other words, observable behaviors of Statelets processes are sequences of *states*, rather then *messages*. This behavior was inspired by Cellular Automata [Neu66], a popular abstraction for modeling complex behaviors in social and biological networks. If a statelet queries for the presence of another statelet, then such situation is treated by the framework as dependency, i.e., the assumption is that any actions triggered by a statelet can discard conditions of dependent statelets. Therefore, the framework ensures that actions of a statelet are triggered after conditions in dependent statelets are checked. Appropriately, if two statelets are mutually dependent, then the framework executes their actions simultaneously, allowing thus for expressing simultaneity dependencies, i.e., network context synchronization and collective actions (see Sec. 5.2). Lifetime of a statelet is bound to the data streams defined within it. A statelet is visible in shared space hypergraph until all its data streams are initialized. Once the statelet starts processing data streams by triggering actions, it becomes invisible to other statelets, i.e., queries being evaluated within `wait` operations of all other statelets will not consider presence of the relation correspondent to the statelet. Let us consider an example: an engineering project can be started if design of all projects it depends on is finalized, and if at least one of them is in the implementation phase. The following code snippet implements this rule:

Listing 5.7: Design finalized phase

```
statelet DesignFinalizedPhase(Project):
{
 wait Depends(Project, DepProject) => (DesignFinalizedPhase(DepProject)
    || ImplementationPhase(DepProject));
 yield new ImplementationPhase(Project);
};
```

5.4.3 Feature Support and Prototype Implementation

All four features outlined in Sec. 5.2 are integral part of and natively supported by Statelets. Data streams and segregation of operations realize the horizontal integration feature. `Wait` operation enables *condition-action* rules. Implicit quantifiers in queries along with explicit quantifiers in commands allow for easy network context querying and processing. Statelet dependency solves the synchronization problem.

Statelets employ accustomed C-based syntax. Prototypes of the Statelets interpretor and the initial version of the language runtime are implemented in the functional programming language F#, and are publicly available for download[3].

The complete abstract syntax tree of the Statelets coordination language is provided below:

Listing 5.8: Statelets syntax tree

```
Quantifier Q ::= any | every | all
Constant C ::= boolean | number | string
```

[3] http://sourceforge.net/p/statelets/

Identifier ID ::= string without spaces
Expression variables EVARS ::= (ID | C | _) list
Command variables EPARS ::= (Q ID | C) list
Expression E ::= EVARS | (ID, EVARS) | (E && E) | (E || E) | (**not** E) | (E −> E)
| (E => E) | (E −! E) | (E −x E)
VirtualRelation VR ::= (ID, ID list , E)
Statement S ::= **define** E | **wait** E | **trigger** ID EPARS | **yield new** ID EPARS
Statelet ::= (ID, ID list , S list)

5.5 Statelets Framework

In this section we present the conceptual architecture of the Statelets framework that enables horizontal integration of collaborative software systems. The focus of this chapter is on the coordination language, therefore technical details are not provided. Figure 5.2 shows the high level design of the framework comprised of the following layers:

Connectors. Groupware APIs are diverse by their nature and employ distinct protocols. This requires creation of fine-tuned integration points, i.e., connectors. Connectors define supported relations and commands, and adapt object models of groupware APIs to fit Statelets semantic model. Connectors may support not only initialization of data streams corresponding to atomic relations, but also interpretation of queries on relations in order to better utilize flexibility of APIs and improve efficiency.

Authentication and Authorization. User-centric APIs are designed for vertical composition [DG11], and often require authorization and authentication mechanisms with direct user involvement (e.g., OAuth 1.0/2.0). This complicates traversal of social graphs, and imposes needs to store and maintain certificates, application and user tokens, or even credentials. Moreover, a mechanism to update or collect new tokens should be present as well.

Entity mapping. Many user accounts and entities map to the same entity in the real world. For instance, users usually have different accounts per each collaboration tool they use, and two files in different tools may represent the same research paper. Typical approaches to entity mappings [KCSS10] are attribute-based identity, by e-mail address, by custom metadata, or even direct mappings (e.g., based on Facebook Open Graph or OpenID).

Optimizations. Authentication and authorization mechanisms together with identity mappings algorithms may introduce high latency. Additionally, some data in social networks, like a friendship connection, or a user profile, change rarely. This introduces unnecessary overhead for queries with existential quantifiers, i.e., 'find any socially related expert in given area'. In this case, caching and heuristic approaches may bring substantial value.

Language Runtime. The language interpretor is responsible for code parsing and interpretation of the language semantic model. The scheduling component is responsible for polling graphs of artifacts and user profiles. The coordination component is responsible for enforcing dependencies between activities and processes at runtime.

The multi-layer design decouples integration and optimization issues from the coordination logic. The developer therefore only operates with entity abstractions and is not

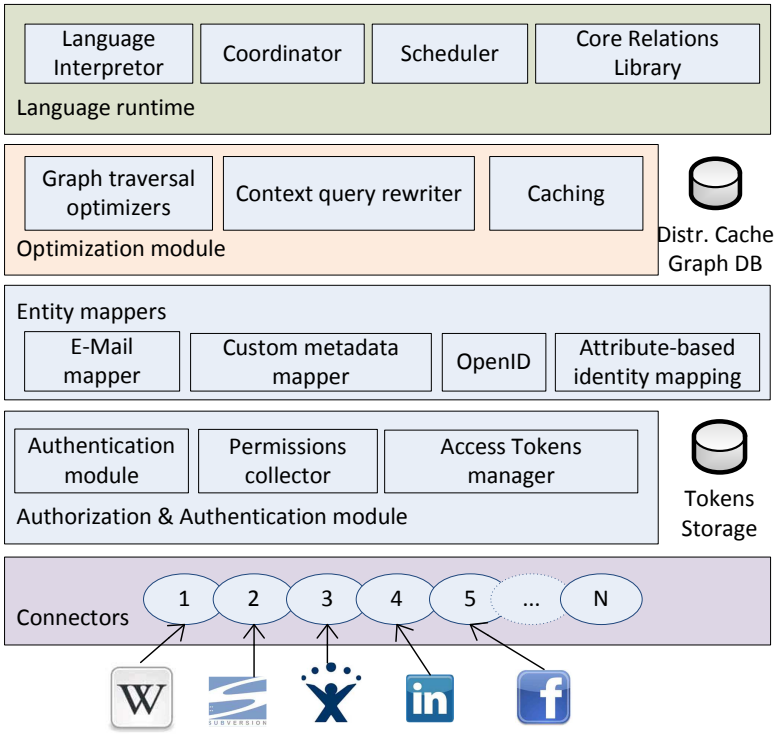


Figure 5.2: Statelets framework architecture

required to comprehend technical details of data access, whereas the other layers are handled by appropriate integration experts.

5.6 Evaluation

This section demonstrates the implementation of two process types considered in Sec. 5.2, namely Analysis and Engineering projects. The use cases exemplify main language features and implementation of coordination based on network effects. More use-cases can be found online[3].

5.6.1 Analysis Projects

MediaWiki engine used in Wikipedia is used as an underlying groupware platform. Typically, work on wiki pages is coordinated by non-functional attributes, for example, 'Category:All articles with unsourced statements'. Similarly, we add a special marker category which is used to denote `Post-Deliberation phase` of a project. Two analysis projects are considered to be related, if one of the project wiki pages contains a link to a wiki page from the other project. Synchronization between related projects is achieved in two steps: (i) residing in the Post-Deliberation phase, a process waits until all

related processes switch to the Post-Deliberation phase; (ii) all changes made in related projects since last synchronization are communicated to every team member in related projects, and related projects switch back to the Deliberation phase simultaneously.

Listing 5.9: Analysis projects

```
statelet AnalysisProject.Deliberation(WikiPage, Timestamp):
{
 wait Wiki.Categories(WikiPage, "PostDeliberation");
 yield new AnalysisProject.PostDeliberation(WikiPage, Timestamp);
};

statelet AnalysisProject.PostDeliberation(WikiPage, Timestamp):
{
 wait
  ((Wiki.Links(WikiPage, RelatedPage) => AnalysisProject.PostDeliberation(RelatedPage, _))
  -> Wiki.Revisions(RelatedPage, _, RelRevTimestamp))
  && >(RelRevTimestamp, Timestamp) && System.DateTime.Now(now);
 define Wiki.Revisions(WikiPage, Contributor, _);
 trigger Wiki.EmailUser(every Contributor, every RelatedPage, all RelRevTimestamp);
 trigger Wiki.DeleteCategory(WikiPage, "PostDeliberation");
 yield new AnalysisProject.Deliberation(WikiPage, now);
}
alternative
{
 wait WikiPage -x Wiki.Categories(WikiPage, "PostDeliberation");
 yield new AnalysisProject.Deliberation(WikiPage, Timestamp);
};
```

This use case exemplifies simplicity of network context synchronization and collective actions implementation in case of recursive collaboration processes.

5.6.2 Engineering Projects

To save space, we exemplify only expertise discovery in social neighborhood. The algorithm combines two ideas: (i) try to find a reviewer from a related project, which is not socially related to any of the project team members; (ii) try to find any reviewer who has appropriate expertise. In this example, social context is retrieved from Facebook, LinkedIn, and Subversion (two engineers are socially related if they committed to the same project in subversion). Project data is retrieved from the VersionOne groupware. Subversion and VersionOne are depicted in the code snippet as `SVN` and `V1` respectively.

Listing 5.10: Engineering projects

```
relation SVN.Related(User1, User2):
 SVN.Logs(Path, User1, _, _, _) && SVN.Logs(Path, User2, _, _, _);

relation SocialRelation(User1, User2):
 SVN.Related(User1, User2) || Facebook.Friends(User1, User2);

statelet EngeneeringProject.InProgress(Story):
{
 wait V1.Attribute(Story, "Status", "Completed");
 yield new EngineeringProject.ImplementationFinished(Story);
}
alternative
{
```

```
 wait
  V1.Attribute(Story, "Status",  "Review") && not V1.Relation(Story, "Reviewer", _)
  && V1.Relation(Story, "Developer", Dev)
  && V1.Relation(Story, "FunctionalRelation", RelStory)
  && V1.Relation(RelStory, "Developer", RelDev)
  && LinkedIn.Profile(RelDev, Profile) && ExpertiseFits(Profile, Story)
  && (not SocialRelation(Dev, RelDev) || RelDev);
 trigger SetRelation(Story, "Reviewer", any RelDev);
 yield new EngineeringProject.InProgress(Story);
};
```

The use case exemplifies implementation of such advanced activities as location of socially connected experts, unbiased reviewers, and so on. The use case also shows benefits arising from horizontal composition of social networking sites.

5.7 Conclusions

This chapter proposes a novel coordination language for network context-based coordination, and demonstrates its suitability through use cases. Compared to existing approaches, our contribution provides a full spectrum of features that are crucial for network context furnishing and coordination based on it. We have shown that these features are necessary for an effective coordination of social collaboration processes. Although Statelets was designed with the focus on collaboration, we do not exclude its applicability in other areas.

CHAPTER 6

Scaling Collaboration-assisted Computation

Existing crowdsourcing database systems fail to support complex, collaborative or responsive crowd work. These systems implement human computation as independent tasks published online, and subsequently chosen by individual workers. Such pull model does not support worker collaboration and its expertise matching relies on workers' subjective self-assessment. An extension to graph query languages combined with an enhanced database system components can express and facilitate social collaboration, sophisticated expert discovery and low-latency crowd work. In this chapter we present such an extension, CRowdPQ, backed up by the database management system Crowdstore.

6.1 Introduction

Crowd-powered hybrid databases have gained momentum in recent years [FKK+11, PP11, PPGM+12] due to their ability to combine human and machine computation. These database engines allow the specification of human-computable predicates that transform into Human Intelligence Tasks (HITs), which are posted online and are expected to be picked up by workers. In spite of being cumbersome for workers, as browsing HITs is time-consuming [KNB+13], such pull model has limitations for collaborative and expert work. The better-suited push model requires the crowd platform to support sophisticated worker discovery capabilities in order to assign or recommend tasks to workers.

Plethora of tasks require synchronous collaboration [IK92, KNB+13]. Successful collaboration can be largely influenced by social relations between human workers. Moreover, reusing teams that exhibited successful collaboration in the past can greatly increase chances of success for new assignments.

Realtime crowdsourcing is based on the concept of flash crowds [KNB+13]: groups of individuals who respond moments after a request and can work synchronously. The

benefits of realtime crowdsourcing have been shown in [BBMK11, BKMB12]: paying workers a small wage to stay on call is enough to draw a crowd together within seconds.

In this chapter we show how a graph query language can be extended to express synchronous collaboration, social formations (teams) of crowd workers, sophisticated worker discovery, as well as complex crowdsourcing patterns, such as iterative computation, control groups, ranking, etc. Also, we show how classical database engine components, such as indexes, caches, and the buffer pool manager, can be extended to improve worker discovery, team formation, and flash crowds management.

6.2 Motivation

Consider following examples of crowdsourcing tasks:

1. Implement a web page, and create images for it. These two tasks require distinct skill sets. Also, to improve collaboration, workers assigned to the tasks should be socially related, or at least in close social vicinity.

2. You have a recorded melody fragment and want to know what song it belongs to. You want only those workers to work on it who like rock music.

3. You need a set of hand-drawn paintings. You ask crowd workers to draw and rate the paintings. To avoid biased ratings, workers rating the paintings should not be socially related to workers drawing the paintings.

Existing crowdsourcing query languages fall short of expressing complex relations between crowd workers working on related tasks, e.g., in the first example. Moreover, discovery of proper workers might by a crowdsourcing task itself, e.g., in the second example. Finally, even trivial crowdsourcing patterns, as in Example 3, are cumbersome to express in existing crowdsourcing query language adaptations.

In the next section we review existing query languages employed for crowdsourcing database scenarios.

6.3 Related Work

Table 6.1 provides an overview of existing hybrid human-machine databases. We analyze their ability to express complex crowdsourcing workflow patterns (such as control groups and ranking), social formations between workers, and sophisticated worker discovery. Also, we overview techniques they employ for query optimizations, i.e., to minimize the number of generated HITs under cooperative/collaborative scenarios, and approaches they utilize to enable realtime crowdsourcing.

Qurk [MWK+11a, MWK+11b] and hQuery [PP11] were among the first attempts on expressing crowdsourcing tasks as declarative queries. The SQL-based query language in Qurk [MWK+11a] exploits user-defined scalar and table functions (called *TASK*) to retrieve, join, sort and filter data from the crowd. Qurk also extends SQL with

Table 6.1: Supported features in selected crowdsourcing databases

	Workflow	Query optimizations	Worker discovery	Social formations	Realtime crowdsourcing
Qurk	⊠	⊠	☐	☐	☐
hQuery	⊠	⊠	☐	☐	☐
CrowdDB	⊠	☐	☐	☐	☐
CrowdSPARQL	⊠	☐	☐	☐	☐
Deco	⊠	☐	☐	☐	☐
CrowdSearcher	⊠	☐	☐	⊠	☐
Join optimizations	N/A	⊠	☐	☐	☐

Legend: ☑ Supported ⊠ Partially Supported ☐ Not Supported

a *POSSIBLY* clause to reduce the number of join candidates. Join optimizations in Qurk consider batching of items, thus minimizing the number of generated HITs. hQuery [PP11], a Datalog-like declarative model, features human-based and algorithm-based predicates. Authors focus on presence of uncertainty in the result set as well as optimization challenges, such as trade-offs between the number of certain answers, time allocated and monetary cost.

CrowdDB [FKK+11] introduces extensions to the SQL data definition language to define *CROWD*-enabled columns and tables, i.e., which should be fetched from an underlying crowdsourcing platform. Also, it introduces CROWDEQUAL and CROWDORDER extensions to the SQL data modification language.

CrowdSPARQL [ASFN12] introduces a hybrid query engine that allows executing SPARQL queries as a combination of machine- and human-driven functionality. Similar to *CROWD*-enabled columns and tables in CrowdDB, CrowdSPARQL defines crowdsourced *classes* and *properties* in VoID (Vocabulary of Interlinked Datasets). Also, CrowdSPARQL defines an *ORDER BY CROWD* operator.

The Deco [PPGM+12] database semantics are defined based on the so-called *Fetch-Resolve-Join* sequence, i.e., data is fetched using *Fetch* rules, then data inconsistencies are resolved using *Resolution* rules and afterwards conceptual relations are produced by outer-joining the resolved tables.

CrowdSearcher [BBC12] allows putting constraints on crowd workers via a mapping model, e.g., friends of a specific user, geo-localized people, workers on a selected work platform. However, it is not possible to specify either relations between workers, or social formations.

Neither of the query languages above allow specifying *CROWD*-enabled constraints on workers, nor relations between crowd workers themselves. Hence, these query languages cannot support examples provided in the previous section. Moreover, they lack capabilities to express complex workflows in a natural way.

Multiple papers discuss the problem of minimizing the number of HITs required

to resolve JOIN operations. We have grouped those papers in the table under the "Join Optimizations" row. CrowdER [WKFF12] suggests using a hybrid human-machine approximation approach to filter out non-matching join pairs (with similarity ratio below certain threshold), aiming to minimize the number of HITs required to join entities. Wang et al. [WLK+13] discuss join optimization based on transitive relations. Contrary, in [MKM+13] authors discuss selectivity estimation performed by the crowd, which implies optimal join ordering. All these approaches focus on crowd-based and automatic join resolution, neglecting a query writer, who can have better insight into selectivity of the data queried and optimal join ordering.

While simple caching of results produced by HITs has been discussed (e.g., [MWK+11b]), it has not been discussed how to cache successful workers and social formations (teams). Also, to the best of our knowledge, no papers have suggested application of classical database techniques and algorithms for realtime crowdsourcing.

6.4 Query Language

In this section we show how Conjunctive Regular Path Queries, a formalism behind many graph query languages [Woo12b], can be extended to overcome their shortcoming for incorporating free-text conditions and relations between data to be fetched and workers who fetch the data.

6.4.1 Preliminaries

A database is defined as a directed graph $K = (V, E)$ labeled over the finite alphabet Σ. If there is a path between node a and node b labeled with $p_1, p_2, ..., p_n$ we write $a \xrightarrow{p_1 p_2 ... p_n} b$. In the remainder of this section we give definitions of (conjunctive) regular path queries, similar to other works, like [CGLV00b].

Definition 1 (Regular Path Queries). A regular path query (RPQ) $Q^R \leftarrow R$ is defined by a regular expression R over Σ. The answer $ans(Q^R, K)$ is the set connected by a path that conforms to the regular language $L(R)$ defined by R:

$$ans(Q^R, K) = \{(a, b) \in V \times V \mid a \xrightarrow{p} b \text{ for } p \in L(R)\}.$$

Conjunctive regular path queries allow to create queries consisting of a conjunction of RPQs, augmented with variables.

Definition 2 (Conjunctive Regular Path Queries). A conjunctive regular path query (CRPQ) has the form

$$Q^C(x_1, ..., x_n) \leftarrow y_1 R_1 y_2 \wedge ... \wedge y_{2m-1} R_m y_{2m},$$

where $x_1, \ldots, x_n, y_1, \ldots, y_m$ are node variables. The variables x_i are a subset of y_i (i.e., $\{x_1, \ldots, x_n\} \subseteq \{y_1, \ldots, y_m\}$), and they are called distinguished variables. The answer $ans(Q^C, K)$ for a CRPQ is the set of tuples $(v_1, ..., v_n)$ of nodes in K such that there is a total mapping σ to nodes, with $\sigma(x_i) = v_i$ for every distinguished variable, and $(\sigma(y_i), \sigma(y_{i+1})) \in ans(Q^R, K)$ for every RPQ Q^R defined by the term $y_i R_i y_{i+1}$.

6.4.2 CRowdPQ

CRowdPQ is derived from CRPQ by extending the notion of RPQ with DRPQ and RRPQ defined as follows.

Definition 3 (DRPQ). A descriptor regular path query (DRPQ) $Q^{DR} \leftarrow DR$ is a regular path query defined over the extended alphabet $\Sigma \bigcup \Gamma$, where Γ is a human-interpretable infinite alphabet of labels. Recursivity of descriptor regular path queries corresponds to iterative human computation. The *Descriptor* relations are free-text conditions that can be answered by human workers.

Definition 4 (RRPQs). A resolver regular path query (RRPQ) is a regular path query over a predefined alphabet $P = produce \cup consume$, where the labels *produce* and *consume* correspond to dataflow producers and consumers respectively. The left operand of a Resolver relation always has to be a worker node supplied by an integrated crowdsourcing platform. Regular expression over the *produce* relation represent higher-order selection of workers, e.g., workers find workers who find workers who can fetch data. *Resolver* relations are dataflow constructs between the data to be fetched and the workers working on the data. Note, Resolvers are not the only relations that can be specified between a worker and the task at hand, i.e., RPQs can be used to specify worker constraints.

6.4.3 Expressiveness

In this section we demonstrate the expressiveness of CRowdPQ by implementing the three use cases from the motivating scenario. For this purpose we employ a CRowdPQ-enhanced version of SPARQL 1.1: Descriptor and Resolver relations are denoted using triangle and square brackets respectively.

Synchronous collaboration. Implement a web page, and create images for it. These two tasks require distinct skill sets. Also, to improve collaboration, workers assigned to the tasks should be socially related, or at least in close social vicinity (i.e., there exists a path between them of maximum length of 2).

Listing 6.1: Synchronous collaboration

```
SELECT ?webPage, ?pictures
WHERE
{
      ?webPage <"Design a web page">.
      ?pictures <"Draw pictures for the web page"> ?webPage.
      ?webDesigner [produce] ?webPage.
      ?artist [produce] ?pictures.
      ?webDesigner friendOf[1,2] ?artist.
      ?webDesigner [consume] ?pictures.
      ?artist [consume] ?webPage.
}
```

Worker discovery. You have a recorded melody fragment and want to know what song it belongs to. You want only those workers to work on it who like rock music.

Listing 6.2: Worker discovery

```
SELECT ?melodyName
WHERE
{
      ?melodyName <"Is similar to"> @file.
      ?melodyName <"Can you recognize the melody?">.
      ?musicFan [produce] ?melodyName.
      ?musicFan <"Find a person passionate about rock music">.
      ?indexWorker [produce] ?musicFan.
}
```

Note, that the specification of workers with no constraints is optional, i.e., $?indexWorker$ can be omitted.

Workflow and social relations. You need a set of hand-drawn pictures. You ask crowd workers to draw and rate the pictures. To avoid biased ratings, workers rating the pictures should not be socially related to workers drawing the pictures.

Listing 6.3: Workflow and social relations

```
SELECT ?picture, ?ranking
WHERE
{
      ?picture <"Draw a funny sheep.">.
      ?talentedPainter [produce] ?picture.
      ?mercilessCritic [consume] ?picture.
      ?mercilessCritic [produce] ?ranking.
      ?ranking <"How funny is this sheep?"> ?picture.
      FILTER NOT EXISTS
      { ?talentedPainter friendOf ?mercilessCritic }
}
```

Note, the example above can be easily changed to a control group (i.e., one worker creates a picture and another one filters it) by replacing the `?rank` variable with the `?filteredPicture` variable and adjusting descriptor relations appropriately.

6.5 Database Engine

In this section we show how classical database components can be extended to be able to cope with human workers as schemaless, volatile and context-dependent data sources.

6.5.1 Synchronous Collaboration: Social Formations and Caching

In Examples 1 and 3 of Section 6.4.3 we have shown the expressivity of our query language with respect to specifying social formations.

In traditional RDBMS, the purpose of query caching is to speed up query evaluation by reusing results from previous queries. While classical caching mechanisms of preserving query results are also applicable in Crowdstore, here we consider a different kind of caching. Instead of caching results, we cache workers and social formations of workers (teams) in case of synchronous collaboration. If a worker has been answering recently a

similar query to the query at hand, she might be a good fit to the task. For example, if a worker has been searching recently through newspapers for information about charity events, she might be able to quickly answer a query of searching recent newspapers for road incidents.

The key element for efficiency of such caches is the ability to identify similarities between queries, which resorts to finding a similar subgraph in a list of cached query graphs (subgraph isomorphism). Matching descriptors can be achieved by finding similarity between texts (or extracting labels). Answering subgraph isomorphism is a NP-complete problem, so when using exact matching (isomorphism) the query cache will not scale. Moreover, for complex queries expressing dense social formations, like cliques, it might be difficult to find an exact match. To alleviate these two problems we can use approximate graph matching, which, however, might not return the most suitable workers. Depending on the importance of the relations between workers, we can choose either of the two heuristics: relax the input query graph by removing worker nodes or data nodes in order to focus on worker experience (i.e., who worked successfully on what) or maximize social similarity respectively (i.e., what teams were successful).

Such quality caches can be pre-built by running pre-labeled queries over gold standard data (e.g., [CB09, DHSC10, LEHB10]) and caching workers and teams that have shown good quality.

6.5.2 Crowd Indexes

In Example 2 of Section 6.4.3 we have shown how the discovery of crowd workers can be crowdsourced itself. We call index workers those workers that select and search for workers for a query at hand. The distinction to regular workers should be driven by different reward mechanisms applied to index workers, i.e., index workers should be rewarded depending on the work quality of the workers they choose. The Crowdstore design incorporates two techniques for worker indexes:

- Routing indexes. In the most trivial case the system can ask an index worker to simply enter a list of workers she thinks satisfy the descriptor relation(s), or a list of index workers who can route further. Routing indexes represent directed graphs, and Crowdstore needs to detect cycles.

- Zonemap indexes. If there are other relations in addition to descriptors that are adjacent to a worker node in a query graph, then Crowdstore can efficiently filter worker candidates. In such case, index workers can be presented with a list of workers they can select from. However, such lists might be immerse, so the system needs to group workers by available tags (e.g., by country, or age), presenting several hierarchical lists to index workers. This approach enables index workers to quickly filter the list of workers.

6.5.3 Query Optimization

One of the central aspects of query optimization is join ordering. Consider the following example: "Where was this picture taken? - this query should be answered by workers living in London". An extremely inefficient case of evaluating such query is sending the question to all the workers, and then filtering responses by workers living in London. When joining descriptor and regular relations the join order is predefined, as automated filtering is always more efficient than filtering done by a crowd. However, join ordering for two crowd-produced relations can be highly error-prone and inefficient as the cost of relations is not known beforehand. Contrarily to existing crowd-powered approaches, in Crowdstore we take a different approach by assuming that a query writer can have better insight into predicate selectivity than a crowd. CRowdPQ, as shown in Example 3 in Section 6.4.3, provides a query writer with the ability to specify join ordering by using *consume* relations. If no *consume* relations are specified, then existing joining techniques can be applied.

Another approach CRowdPQ provides is "denormalized" ("collaborative") joins: instead of asking crowd workers to work independently on two separate relations, Crowdstore can ask workers to collaborate and produce already matched and joined results. The benefit of "collaborative" join is that the worker produced data can be ambiguous and, without direct contact with the data producer, difficult to match. Moreover, creative tasks require collaboration, as shown in Example 1 in Section 6.4.3. If a single worker node in a query is connected with a *produce* relation to multiple nodes, then "denormalization" will result in sending a single HIT to a crowd worker asking to provide data for the whole query graph. When working on "collaborative" joins, crowd workers will need to use synchronous collaboration software.

Joining two crowd-produced relations without predefined join ordering allows two approaches. The first approach consists of two sets of workers producing data for relations independently and in parallel, and then a third set of workers joins the two produced relations. The second approach is inherent to relational DBMS, i.e., data is produced for one relation and then is used to filter in-place data for another relation.

6.5.4 Crowd Pool

In [BBMK11, BKMB12] the authors show that paying workers a small wage to stay on call is enough to draw a crowd together two to three seconds later. The problem here is which workers to keep on payroll based on variable query patterns, e.g., what subset of workers satisfy most queries given the budget constraints. If a worker becomes less active, it is better to replace the worker with another one. Basically, an efficient system needs to maintain a limited set of useful/active workers and efficiently replace ineffective workers with new ones. This scenario resembles problems addressed by the buffer pool manager in traditional RDBMs, i.e., limited working set, replacement of least recently used database pages. Henceforth, we draw here correspondence between crowd workers and database pages: similarly as how a crowd worker can generate/provide data, a database page can provide table records. The central part of the buffer pool manager in RDBMs is the clock

algorithm, which evicts least-recently-used pages (LRU). The Crowd pool component in Crowdstore similarly evicts least-recently-active (LRA) workers. The Crowdstore adaptation of the clock algorithm, however, incorporates the following adjustments:

- Tracking slow-performing workers. The purpose of keeping a page in-memory in RDBMs is the ability to fetch results faster. Similarly, keeping a worker on payroll leads to the expectation of fast results. If a worker responds slower than other payroll (and non-payroll) workers, then Crowd pool can evict such worker.
- Delayed enrollment on payroll. In RDBMs when reading records from a database page it is necessary to fetch the page in memory (page-in). In Crowdstore, however, there is no such restriction, i.e., even if some query required workers with a skill set disjoint with skill sets in Crowd pool, such skill set might not be needed again. So, apart of counting how useful is a payroll worker, Crowd pool needs to count how useful a non-payroll worker is.

6.6 Conclusions

In this chapter we present the hybrid human-machine database Crowdstore, powered by the graph query extension CRowdPQ. Contrarily to existing crowdsourcing query languages, CRowdPQ can express social collaborations between crowd workers, sophisticated worker discovery and complex crowdsourcing workflow patterns. Incorporation of dataflow constructs makes CRPQs slightly less declarative, since a query writer can directly influence execution plans. However, mispredictions in query evaluation performed by the crowd possess considerable cost overhead, rendering explicit join ordering critical. Crowdstore serves as a holistic design concept of a new generation crowdsourcing database, featuring extended indexes, caches and buffer pool manager.

CHAPTER 7

Conclusions and Future Research

In this chapter we summarize the main results of this thesis. Section 7.1 describes the main outcomes of the conducted research with respect to advancing the state of the art in human computation. In Section 7.2 we revisit and critically analyze the research questions posed in Section 1.2. Finally, Section 7.3 provides future outlook on open topics.

7.1 Summary of Contributions

In this thesis we have introduced a novel technique, called *collaboration-assisted computation*, which focuses on empowering human computation with social traits and collaborative capabilities. The thesis defined what constitutes collaboration-assisted computation, and postulated what programming models are needed to support and promote collaboration-based computation at various phases of software engineering process. To express complex social formations, a graph query language extension was introduced. To intuitively capture social collaboration, a modeling approach along with a visual notation were devised. To simplify implementation and coordination of collaborative processes, the Statelets interpreter was implemented. To scale collaborative processes, a crowdsourcing database Crowdstore was designed.

Evaluation of modeling and programming languages is difficult as language intuitiveness, expressivity and simplicity are subjective metrics. Therefore, each of the chapters presented and discussed implementations of elaborate and comprehensive use cases. The use cases were reflecting real-world scenarios and explicitly demonstrated the importance of the research issues.

Figure 7.1 revisits Figure 1.2 in Section 1.2 and highlights languages and frameworks introduced in the thesis. For the sake of simplicity, Statelets and Crowdstore were presented in the respective chapters in isolation from other components. However, holistic integration of all the framework components is easily achievable. For example, Statelets

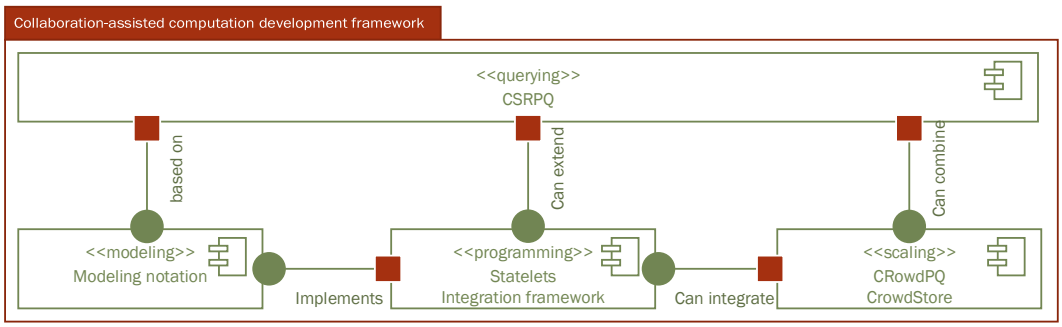


Figure 7.1: Collaboration-based computation development framework overview

can be easily extended to parse CSRPQs as part of Statelet `queries`, while CRowdPQ can be easily combined with CSRPQ as both extensions rely on Conjunctive Regular Path Queries. Similarly, Crowdstore can be integrated into Statelets framework via an additional `connector`.

Synergy of CSRPQ, the modeling approach, Statelets and Crowdstore is, what we believe, a comprehensive framework that simplifies modeling, coordination, and scaling of social collaboration processes.

7.2 Research Questions Revised

In this section we revisit research questions posed in Section 1.2 with respect to: (i) how they have been answered in this thesis; (ii) limitations of our contributions.

Question 1: *How to express complex social formations with relaxed structure in a formal and structured way?*

In Chapter 3 we devised a CRPQ extension, called CSRPQ, that can express complex social formations in a succinct and intuitive way. The expressivity and intuitiveness have been demonstrated via implementation of various patterns and characteristics widely used in social network analysis, such as: closeness centrality, connectedness, k-plex, liaison, structural equivalence, and components (i.e., independent teams). CSRPQ defines a notion of a group as a first-class citizen, which allows specification of not only social structure within a team, but also inter-team social traits. The flexibility in a group structure is achieved by connectedness constraints on a group. This approach is more suitable than approximate graph matching for the domain of social network analysis, as definitions for common social network patters are exact and approximating them can produce completely irrelevant results.

While CSRPQ expressivity was studied over the most interesting patterns in social network analysis, there exist many more patterns, such as n-clans, LS sets, lambda

sets [BES90], which are yet to study in the context of CSRPQ expressiveness and their applicability to collaboration-assisted computing.

Question 2: *How to model collaboration-assisted computation?*

Chapter 4 introduced a novel approach for modeling social collaboration as evolution of overlay of social and document networks. The bottom-up nature of the modeling approach provides the collaborators with flexibility on how to drive collaboration, i.e. the approach focuses on specifying what has to be achieved rather than how to achieve it. Integration of CSRPQ in the modeling approach allows to express complex social context of collaborative teams.

Even though the modeling approach has the formal specification, it does not have any serialization scheme defined to enable programming of tools that can analyze process definitions and generate source code in execution languages, such as Statelets.

Question 3: *How to simplify implementation of collaboration-assisted computation?*

As per se, collaboration-assisted computation relies on collaborative and social software. In Chapter 5 we introduced a framework that enables easy integration and coordination of various groupware and social networking platforms. The framework features Statelets, a coordination language capable of expressing complex synchronization between collaborating teams.

Unlike the other languages introduced in this thesis, Statelets was designed from scratch rather than as an extension to an existing language. For example, Statelets might have been designed as a language integrated query (LINQ[1]) extension in C#, making it easier to integrate with plethora of existing programming frameworks and libraries.

Question 4: *How to scale collaboration-assisted computation?*

Chapter 6 proposed a hybrid crowdsourcing database model Crowdstore that enables synergy of crowdsourcing platforms and collaboration-assisted computation. Crowdstore focuses on addressing requirements posed by collaboration-assisted computation, such as worker selection based on social context, and worker coordination. Moreover, Crowdstore proposes solutions to enable real-time collaborative work that potentially can provide instant results.

Crowdstore relies on unconventional "push" model for task distribution among workers, while many existing crowdsourcing platforms rely on "pull" model. This discrepancy can complicate integration with the existing crowdsourcing platforms.

7.3 Future Work

Premise of human computation to solve problems difficult for machines is bounded by human-machine interface. Machine components require structured input with a predefined schema that not always can accommodate human outputs, such as free text, or drawings. Deep neural networks is a promising approach that can bridge this gap, as it can be used to classify human output into machine inputs with predefined structure.

[1] https://msdn.microsoft.com/en-us/library/bb397926.aspx

Another interesting topic is group selection using machine learning techniques to offload detection of successful collaboration patterns to deep neural networks, rather than asking designers or programmers to specify the target group social traits or context.

Bibliography

[AAB+05] A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C. K. Liu, S. Thatte, P. Yendluri, and A. editors. Yiu. Web services business process execution language version 2.0., May 2005.

[AAD+07] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, et al. Web services human task (ws-humantask), version 1.0. *available at http://incubator.apache.org/hise/WS-HumanTask_v1.pdf*, 2007.

[ABMK11] Salman Ahmad, Alexis Battle, Zahan Malkani, and Sepander Kamvar. The jabberwocky programming environment for structured social computing. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 53–64, New York, NY, USA, 2011. ACM.

[AFG+12] Liliana Ardissono, Roberto Furnari, Anna Goy, Giovanna Petrone, and Marino Segnan. The context aware workflow execution framework. *International Journal of Autonomous and Adaptive Communications Systems*, 5(1):58–76, 2012.

[AG08] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, February 2008.

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[ASFN12] Maribel Acosta, Elena Simperl, Fabian Flöck, and Barry Norton. A sparql engine for crowdsourcing query processing using microtasks. *Institute AIFB, KIT, Karlsruhe*, 2012.

[AYLY09] Sihem Amer-Yahia, Laks V. S. Lakshmanan, and Cong Yu. Socialscope: Enabling information discovery on social content sites. *CoRR*, abs/0909.2058, 2009.

[BBC12] Alessandro Bozzon, Marco Brambilla, and Stefano Ceri. Answering search queries with crowdsearcher. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 1009–1018, New York, NY, USA, 2012. ACM.

[BBMK11] Michael S. Bernstein, Joel Brandt, Robert C. Miller, and David R. Karger. Crowds in two seconds: Enabling realtime crowd-powered interfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 33–42, New York, NY, USA, 2011. ACM.

[BDR07] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[BE93] Stephen P. Borgatti and Martin G. Everett. Two algorithms for computing regular equivalence. *Social Networks*, 15(4):361 – 376, 1993.

[BEF02] S.P. Borgatti, M.G. Everett, and L.C. Freeman. UCINET 6 For Windows: Software for Social Network Analysis, 2002.

[BES90] Stephen P Borgatti, Martin G Everett, and Paul R Shirey. Ls sets, lambda sets and other cohesive subsets. *Social Networks*, 12(4):337–357, 1990.

[BFLM01] Jean-Pierre BanÄtre, Pascal Fradet, and Daniel Le MÃ©tayer. Gamma and the chemical reaction model: Fifteen years after. In Cristian Calude, Gheorghe PAun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Multiset Processing*, volume 2235 of *Lecture Notes in Computer Science*, pages 17–44. Springer Berlin / Heidelberg, 2001.

[BFS00] Peter Buneman, Mary Fernandez, and Dan Suciu. Unql: a query language and algebra for semistructured data based on structural recursion. *The VLDB Journal*, 9(1):76–110, March 2000.

[BGH+07] Kamal Bhattacharya, Cagdas Evren Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards Formal Analysis of Artifact-Centric Business Process Models. In *5th International Conference on Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 2007.

[BHLW10] Pablo Barcelo, Carlos Hurtado, Leonid Libkin, and Peter Wood. Expressive languages for path queries over graph-structured data. In *29th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '10, pages 3–14, New York, NY, USA, 2010. ACM.

[BHS09] Kamal Bhattacharya, Richard Hull, and Jianwen Su. A data-centric design methodology for business processes. In *Handbook of Research on Business Process Modeling, chapter 23*, pages 503–531, 2009.

[BIJ02] H. Blau, N. Immerman, and D. Jensen. A visual language for querying and updating graphs. Technical Report 2002-037, Department of Computer Science, University of Massachusetts, 2002.

[BKMB12] Michael S. Bernstein, David R. Karger, Robert C. Miller, and Joel Brandt. Analytic methods for optimizing realtime crowdsourcing. *Computing Research Repository*, 2012.

[BM02] Vladimir Batagelj and Andrej Mrvar. Pajek - analysis and visualization of large networks. In *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 8–11. Springer Berlin / Heidelberg, 2002.

[Bra08] Daren C Brabham. Crowdsourcing as a model for problem solving an introduction and cases. *Convergence: the international journal of research into new media technologies*, 14(1):75–90, 2008.

[BW95] Paulo Barthelmess and Jacques Wainer. Workflow systems: a few definitions and a few suggestions. In *Proceedings of conference on Organizational computing systems*, COCS '95, pages 138–147, New York, NY, USA, 1995. ACM.

[BW03] Pavel Balabko and Alain Wegmann. Context Based Reasoning in Business Process Models. In *2003 IEEE International Conference on Information Reuse and Integration (IRI - 2003)*, pages 120–128, Washington, DC, USA, 2003. IEEE Computer Society.

[BZ05] Nadia Busi and Gianluigi Zavattaro. Prioritized and parallel reactions in shared data space coordination languages. In Jean-Marie Jacquet and Gian Picco, editors, *Coordination Models and Languages*, volume 3454 of *Lecture Notes in Computer Science*, pages 207–219. Springer Berlin / Heidelberg, 2005.

[CB06] Cosmin Carabelea and Olivier Boissier. Coordinating agents in organizations using social commitments. *Electron. Notes Theor. Comput. Sci.*, 150(3):73–91, May 2006.

[CB09] Chris Callison-Burch. Fast, cheap, and creative: Evaluating translation quality using amazon's mechanical turk. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, pages 286–295, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[CCPP95] F. Casati, S. Ceri, B Pernici, and G. Pozzi. Conceptual modeling of workflows. In Michael Papazoglou, editor, *OOER '95: Object-Oriented and Entity-Relationship Modeling*, volume 1021 of *Lecture Notes in Computer Science*, pages 341–354. Springer Berlin / Heidelberg, 1995.

[CGK+03] F. Curbera, Y. Goland, J. Klein, F. Leymann, S. Weerawarana, et al. *Business Process Execution Language for Web Services (BPEL4WS), Version 1.1.* 2003.

[CGLV00a] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, pages 176–185, 2000.

[CGLV00b] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proceedings of the 2000 International Conference on Knowledge Representation and Reasoning*, KR'2000, pages 176–185, Breckenridge, Colorado, USA, 2000.

[Chw00] Michael Suk-Young Chwe. Communication and coordination in social networks. *Review of Economic Studies*, 67(1):1–16, 2000.

[CM89] M. P. Consens and A. O. Mendelzon. Expressing structural hypertext queries in graphlog. In *Proceedings of the second annual ACM conference on Hypertext*, HYPERTEXT '89, pages 269–292, New York, NY, USA, 1989. ACM.

[CM90] Mariano P. Consens and Alberto O. Mendelzon. Graphlog: a visual formalism for real life recursion. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '90, pages 404–416, New York, NY, USA, 1990. ACM.

[CMW87] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. *SIGMOD Rec.*, 16(3):323–330, December 1987.

[CN85] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, February 1985.

[Con68] M.E. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[DB11] Schahram Dustdar and Kamal Bhattacharya. The social compute unit. *Internet Computing, IEEE*, 15(3):64 –69, may-june 2011.

[DD00] Gary W. Dickson and Gerardine DeSanctis. *Information Technology and the Future Enterprise: New Models for Managers.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[Dey01] Anind K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[DG11] S. Dustdar and M. Gaedke. The social routing principle. *Internet Computing, IEEE*, 15(4):80 –83, july-aug. 2011.

[DH87] David and Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[DHSC10] Julie S. Downs, Mandy B. Holbrook, Steve Sheng, and Lorrie Faith Cranor. Are your participants gaming the system?: Screening mechanical turk workers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2399–2402, New York, NY, USA, 2010. ACM.

[Die01] Jan L.G. Dietz. Demo: Towards a discipline of organisation engineering. *European Journal of Operational Research*, 128(2):351–363, 2001.

[DNDR09] Anton Dries, Siegfried Nijssen, and Luc De Raedt. A query language for analyzing networks. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 485–494, New York, NY, USA, 2009. ACM.

[DRH11] Anhai Doan, Raghu Ramakrishnan, and Alon Y Halevy. Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4):86–96, 2011.

[Dus04] Schahram Dustdar. Caramba â a process-aware collaboration system supporting ad hoc and collaborative processes in virtual teams. *Distributed and Parallel Databases*, 15:45–66, 2004.

[EBGC09] Guillaume Erétéo, Michel Buffa, Fabien Gandon, and Olivier Corby. Analysis of a real online social network using semantic web frameworks. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 180–195, Berlin, Heidelberg, 2009. Springer-Verlag.

[EG02] Gregor Engels and Luuk Groenewegen. Towards team-automata-driven object-oriented collaborative work. In Wilfried Brauer, Hartmut Ehrig, Juhani Karhumäki, and Arto Salomaa, editors, *Formal and Natural Computing*, volume 2300 of *Lecture Notes in Computer Science*, pages 247–255. Springer Berlin / Heidelberg, 2002.

[EGR91] Clarence A Ellis, Simon J Gibbs, and Gail Rein. Groupware: some issues and experiences. *Communications of the ACM*, 34(1):39–58, 1991.

[Ell97] Clarence Ellis. Team automata for groupware systems. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work: the integration challenge*, GROUP '97, pages 415–424, New York, NY, USA, 1997. ACM.

[FBG96] Gert Florijn, Timo Besamusca, and Danny Greefhorst. Ariadne and hopla: Flexible coordination of collaborative processes. In Paolo Ciancarini and Chris Hankin, editors, *Coordination Languages and Models*, volume 1061

of *Lecture Notes in Computer Science*, pages 197–214. Springer Berlin / Heidelberg, 1996.

[FFLS00] Mary Fernández, Daniela Florescu, Alon Levy, and Dan Suciu. Declarative specification of web sites with strudel. *The VLDB Journal*, 9(1):38–55, March 2000.

[FHS09] Christian Fritz, Richard Hull, and Jianwen Su. Automatic construction of simple artifact-based business processes. In *12th International Conference on Database Theory*, volume 361 of *ACM International Conference Proceeding Series*, pages 225–238. ACM, 2009.

[FKK+11] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: Answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 61–72, New York, NY, USA, 2011. ACM.

[FLM+11] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Yinghui Wu. Adding regular expressions to graph reachability and pattern queries. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 39 –50, apr 2011.

[FLP+06] Dieter Fensel, Holger Lausen, Axel Polleres, Joe de Bruijn, Michael Stollberg, Dumitru Roman, and John Domingue. *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[FMR+09] Dirk Fahland, Jan Mendling, Hajo A. Reijers, Barbara Weber, Matthias Weidlich, and Stefan Zugal. Declarative versus Imperative Process Modeling Languages: The Issue of Maintainability. In *Business Process Management Workshops 2009*, volume 43 of *Lecture Notes in Business Information Processing*, pages 477–488. Springer, 2009.

[FMS09] John Field, Maria-Cristina Marinescu, and Christian Stefansen. Reactors: A data-oriented synchronous/asynchronous programming model for distributed applications. *Theoretical Computer Science*, 410(2â3):168 – 201, 2009.

[GC92] David Gelernter and Nicholas Carriero. Coordination languages and their significance. *Commun. ACM*, 35:97–107, February 1992.

[GGJ+10] Andrea Galeotti, Sanjeev Goyal, Matthew O. Jackson, Fernando Vega-Redondo, and Leeat Yariv. Network games. *Review of Economic Studies*, 77(1):218–244, 2010.

[GP07] Steve Gregory and Martha Paschali. A prolog-based language for workflow programming. In Amy Murphy and Jan Vitek, editors, *Coordination Models and Languages*, volume 4467 of *Lecture Notes in Computer Science*, pages 56–75. Springer Berlin / Heidelberg, 2007.

[GT06] Gösta Grahne and Alex Thomo. Regular path queries under approximate semantics. *Annals of Mathematics and Artificial Intelligence*, 46(1-2):165–190, February 2006.

[GV06] Stijn Goedertier and Jan Vanthienen. Designing Compliant Business Processes with Obligations and Permissions. In *Business Process Management Workshops 2006*, volume 4103 of *Lecture Notes in Computer Science*, pages 5–14. Springer, 2006.

[HA99] C. Hagen and G. Alonso. Beyond the black box: event-based inter-process communication in process support systems. In *Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference on*, pages 450 –457, 1999.

[HDF+10] Richard Hull, Elio Damaggio, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath, Stacy Hobson, Mark H. Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Sukaviriya, and Roman Vaculín. Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles. In *7th International Workshop on Web Services and Formal Methods*, volume 6551 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2010.

[How08] Jeff Howe. *Crowdsourcing: How the power of the crowd is driving the future of business.* Random House, 2008.

[HPW09] Carlos A. Hurtado, Alexandra Poulovassilis, and Peter T. Wood. Ranking approximate answers to semantic web queries. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications*, ESWC 2009 Heraklion, pages 263–277, Berlin, Heidelberg, 2009. Springer-Verlag.

[HS08] Huahai He and Ambuj K. Singh. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 405–418, 2008.

[Hul08] Richard Hull. Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In *On the Move to Meaningful Internet Systems (OTM 2008)*, volume 5332 of *Lecture Notes in Computer Science*, pages 1152–1163. Springer, 2008.

[IK92] Hiroshi Ishii and Minoru Kobayashi. Clearboard: A seamless medium for shared drawing and conversation with eye contact. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 525–532. ACM, 1992.

[JN03] David Jensen and Jennifer Neville. Data mining in social networks. In *Dynamic Social Network Modeling and Analysis: Workshop Summary and Papers*, pages 287–302, 2003.

[KC04] Graham Klyne and Jeremy J. Carroll, editors. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. World Wide Web Consortium, February 2004.

[KCH12] Anand Kulkarni, Matthew Can, and Björn Hartmann. Collaboratively crowdsourcing workflows with turkomatic. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 1003–1012, New York, NY, USA, 2012. ACM.

[KCM06] David Kitchin, William Cook, and Jayadev Misra. A language for task orchestration and its semantic properties. In Christel Baier and Holger Hermanns, editors, *CONCUR 2006 â Concurrency Theory*, volume 4137 of *Lecture Notes in Computer Science*, pages 477–491. Springer Berlin / Heidelberg, 2006.

[KCSS10] Moo Nam Ko, G.P. Cheek, M. Shehab, and R. Sandhu. Social-networks connect services. *Computer*, 43(8):37 –43, aug. 2010.

[Kib06] Rodger Kibble. Speech acts, commitment and multi-agent communication. *Comput. Math. Organ. Theory*, 12(2-3):127–145, October 2006.

[KKL+05] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic. Ws-bpel extension for people–bpel4people. *Joint white paper, IBM and SAP*, 2005.

[KNB+13] Aniket Kittur, Jeffrey V. Nickerson, Michael Bernstein, Elizabeth Gerber, Aaron Shaw, John Zimmerman, Matt Lease, and John Horton. The future of crowd work. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, pages 1301–1318, New York, NY, USA, 2013. ACM.

[Kos01] Alex Kosorukoff. Human based genetic algorithm. In *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, volume 5, pages 3464–3469. IEEE, 2001.

[KS01] Yaron Kanza and Yehoshua Sagiv. Flexible queries over semistructured data. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, pages 40–51, New York, NY, USA, 2001. ACM.

[KSI+08] Gjergji Kasneci, Fabian M. Suchanek, Georgiana Ifrim, Shady Elbassuoni, Maya Ramanath, and Gerhard Weikum. Naga: harvesting, searching and ranking knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1285–1288, 2008.

[KSKK11] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E. Kraut. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 43–52, New York, NY, USA, 2011. ACM.

[LBW07] Rong Liu, Kamal Bhattacharya, and Frederick Y. Wu. Modeling business contexture and behavior using business artifacts. In *Proceedings of the 19th international conference on Advanced information systems engineering*, CAiSE'07, pages 324–339, Berlin, Heidelberg, 2007. Springer-Verlag.

[LEHB10] John Le, Andy Edmonds, Vaughn Hester, and Lukas Biewald. Ensuring quality in crowdsourced search relevance evaluation: The effects of training question distribution. In *SIGIR 2010 workshop on crowdsourcing for search evaluation*, pages 21–26, 2010.

[Les] Ulf Leser. A query language for biological networks. *Bioinformatics*, 21(suppl 2):ii33–ii39.

[LKS+14] Vitaliy Liptchinsky, Roman Khazankin, Stefan Schulte, Benjamin Satzger, Hong-Linh Truong, and Schahram Dustdar. On modeling context-aware social collaboration processes. *Information Systems*, 43:66–82, 2014.

[LKTD12a] Vitaliy Liptchinsky, Roman Khazankin, Hong Linh Truong, and Schahram Dustdar. A novel approach to modeling context-aware and social collaboration processes. In *CAiSE*, volume 7328 of *Lecture Notes in Computer Science*, pages 565–580. Springer, 2012.

[LKTD12b] Vitaliy Liptchinsky, Roman Khazankin, Hong-Linh Truong, and Schahram Dustdar. Statelets: Coordination of social collaboration processes. In Marjan Sirjani, editor, *Coordination Models and Languages*, volume 7274 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2012.

[LMM99] Diego Latella, Istvan Majzik, and Mieke Massink. Towards a formal operational semantics of uml statechart diagrams. In *Proceedings of the IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, pages 465–. Kluwer, B.V., 1999.

[LSSD15] Vitaliy Liptchinsky, Benjamin Satzger, Stefan Schulte, and Schahram Dustdar. Crowdstore: A crowdsourcing graph database. In *International*

Conference on Collaborative Computing: Networking, Applications and Worksharing, pages 72–81. Springer, 2015.

[LSZD13] Vitaliy Liptchinsky, Benjamin Satzger, Rostyslav Zabolotnyi, and Schahram Dustdar. Expressive languages for selecting groups from graph-structured data. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 761–770, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[MA98] David W. McDonald and Mark S. Ackerman. Just talk to me: a field study of expertise location. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, CSCW '98, pages 315–324, New York, NY, USA, 1998. ACM.

[MBM+07] David Martin, Mark Burstein, Drew McDermott, Sheila A. McIlraith, Massimo Paolucci, Katia P. Sycara, Deborah L. McGuinness, Evren Sirin, and Naveen Srinivasan. Bringing Semantics to Web Services with OWL-S. *World Wide Web*, 10(3):243–277, 2007.

[MGW11] Mauro San Martín, Claudio Gutierrez, and Peter T. Wood. Snql: A social networks query and transformation language. In *AMW*, 2011.

[MKM+13] Adam Marcus, David Karger, Samuel Madden, Robert Miller, and Sewoong Oh. Counting with the crowd. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 109–120. VLDB Endowment, 2013.

[MM06] Ignacio J. Martinez-Moyano. *Exploring the Dynamics of Collaboration in Interorganizational Settings*, chapter 4. Jossey-Bass, San Francisco, 2006.

[Mos57] A. Mostowski. On a generalization of quantifiers. *Fundamenta Mathematicae*, 44:12–36, 1957.

[MRH07] Dominic Müller, Manfred Reichert, and Joachim Herbst. Data-driven modeling and coordination of large process structures. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4803 of *Lecture Notes in Computer Science*, pages 131–149. Springer Berlin / Heidelberg, 2007.

[MWK+11a] Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. Human-powered sorts and joins. *Proceedings of the VLDB Endowment*, 5(1):13–24, September 2011.

[MWK+11b] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Crowdsourced databases: Query processing with people. In *5th Biennial Conference on Innovative Data Systems Research*, 2011.

[NB03] Selmin Nurcan and Judith Barrios. Enterprise knowledge and information system modelling in an evolving environment. In *First International Workshop on Engineering Methods to Support Information Systems Evolution*, September 2003.

[NC03] Anil Nigam and Nathan S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.

[Neu66] John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[NKM+10] P. Nandi, D. Koenig, S. Moser, R. Hull, V. Klicnik, S. Claussen, M. Kloppmann, and J. Vergo. Data4bpm, part 1: Introducing business entities and the business entity definition language (bedl), April 2010.

[NN08] Angel Nunez and Jacques Noye. An event-based coordination model for context-aware applications. In Doug Lea and Gianluigi Zavattaro, editors, *Coordination Models and Languages*, volume 5052 of *Lecture Notes in Computer Science*, pages 232–248. Springer Berlin / Heidelberg, 2008.

[Nur08] Selmin Nurcan. A survey on the flexibility requirements related to business processes and modeling artifacts. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, HICSS '08, pages 378–388, Washington, DC, USA, 2008. IEEE Computer Society.

[Ols08] Michael Olson. The amateur search. *ACM SIGMOD Record*, 37(2):21–24, 2008.

[Oye74] Paul D. Oyer. Peopleware: key to success of information systems. *SIGCPR Comput. Pers.*, 5(2):2–6, December 1974.

[PAG06] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2006.

[PE10] Hubert Plociniczak and Susan Eisenbach. Jerlang: Erlang with joins. In Dave Clarke and Gul Agha, editors, *Coordination Models and Languages*, volume 6116 of *Lecture Notes in Computer Science*, pages 61–75. Springer Berlin / Heidelberg, 2010.

[PP11] Aditya Parameswaran and Neoklis Polyzotis. Answering queries using humans, algorithms and databases. In *Conference on Inovative Data Systems Research (CIDR 2011)*. Stanford InfoLab, January 2011.

[PPGM+12] Aditya Ganesh Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: Declarative crowdsourcing. In *Proceedings of the 21st ACM International Conference on Information*

and Knowledge Management, CIKM '12, pages 1203–1212, New York, NY, USA, 2012. ACM.

[PPI04] Anne Powell, Gabriele Piccoli, and Blake Ives. Virtual teams: a review of current literature and directions for future research. *SIGMIS Database*, 35:6–36, February 2004.

[PvdA06] Maja Pesic and Wil M. P. van der Aalst. A Declarative Approach for Flexible Business Processes Management. In *Business Process Management Workshops 2006*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2006.

[QB11] Alexander J Quinn and Benjamin B Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1403–1412. ACM, 2011.

[Rei87] R. Reiter. *On closed world data bases*, pages 300–310. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[RGW11] Mauro San Martin Ramas, Claudio Gutierrez, and Peter T. Wood. Snql: Social networks query language. Technical Report TR/DCC-2011-05, Departamento de Ciencias de la Computacion, Universidad de Chile, 2011.

[RR06] Michael Rosemann and Jan Recker. Context-aware Process Design: Exploring the Extrinsic Drivers for Process Flexibility. In *Seventh Workshop on Business Process Modeling, Development, and Support (BPMDS'06) at the 18th International Conference on Advanced Information Systems Engineering (CAiSE'06)*, volume 749 of *CEUR Workshop Proceedings*, pages 149–158. CEUR-WS.org, 2006.

[RRFA06] Michael Rosemann, Jan Recker, Christian Flender, and Peter Ansell. Understanding Context-Awareness in Business Process Design. In *17th Australasian Conference in Information Systems (ACIS 2006)*. Association for Information Systems, Atlanta, GA, USA, 2006.

[RS09] Royi Ronen and Oded Shmueli. Soql: A language for querying and creating data in social networks. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 1595–1602, Washington, DC, USA, 2009. IEEE Computer Society.

[SA07] Dafna Shahaf and Eyal Amir. Towards a theory of ai completeness. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, pages 150–155, 2007.

[San11a] J.L.C. Sanz. Entity-centric operations modeling for business process management - a multidisciplinary review of the state-of-the-art. In *Service*

Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on, pages 152 –163, dec. 2011.

[San11b] Jorge L. C. Sanz. Entity-Centric Operations Modeling for Business Process Management – A Multidisciplinary Review of the State-of-the-Art. In *IEEE 6th International Symposium on Service Oriented System Engineering (SOSE 2011)*, pages 152–163, 2011.

[SC05] Chetan Shankar and Roy Campbell. A policy-based management framework for pervasive systems using axiomatized rule-actions. In *Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications*, pages 255–258, Washington, DC, USA, 2005. IEEE Computer Society.

[Sch94] Doug Schuler. Social computing. *Communications of the ACM*, 37(1):28–29, 1994.

[Sco00] John P. Scott. *Social Network Analysis: A Handbook*. SAGE Publications, 2 edition, January 2000.

[Sin99] Munindar P. Singh. An ontology for commitments in multiagent systems. *Artif. Intell. Law*, 7(1):97–113, 1999.

[SN07] Oumaima Saidani and Selmin Nurcan. Towards Context Aware Business Process Modelling. In *8th Workshop on Business Process Modeling, Development, and Support (BPMDS'07) at the 19th International Conference on Advanced Information Systems Engineering (CAiSE'07)*. Springer Berlin Heidelberg, 2007.

[SN09] Oumaima Saidani and Selmin Nurcan. Context-Awareness for Adequate Business Process Modelling. In *Third IEEE International Conference on Research Challenges in Information Science (RCIS 2009)*, pages 177–186, Washington, DC, USA, 2009. IEEE Computer Society.

[spa12] Sparql 1.1 query language – w3c working draft 05 january 2012. 2012.

[SSP12] Daniel Schall, Benjamin Satzger, and Harald Psaier. Crowdsourcing tasks to social networks in BPEL4People. *World Wide Web*, 2012.

[SSSA12] Stefan Schulte, Dieter Schuller, Ralf Steinmetz, and Sven Abels. Plug-and-play virtual factories. *IEEE Internet Computing*, 16(5):78–82, 2012.

[STD11] Daniel Schall, Hong-Linh Truong, and Schahram Dustdar. The human-provided services framework. In *Socially Enhanced Services Computing*, pages 1–15. Springer, 2011.

[TD09] Hong Linh Truong and Schahram Dustdar. A survey on context-aware web service systems. *International Journal of Web Information Systems*, 5(1):5–31, 2009.

[Tea12] Neo4J Team. The neo4j manual. 2012.

[VA09] Luis Von Ahn. Human computation. In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, pages 418–419. IEEE, 2009.

[vdABEW00] W. van der Aalst, P. Barthelmess, C. Ellis, and J. Wainer. Workflow modeling using proclets. In Peter Scheuermann and Opher Etzion, editors, *Cooperative Information Systems*, Lecture Notes in Computer Science, pages 198–209. Springer Berlin / Heidelberg, 2000.

[vdASW03] Wil M.P. van der Aalst, Moniek Stoffele, and J.W.F. Wamelink. Case handling in construction. *Automation in Construction*, 12(3):303–320, 2003.

[vdAWG05] Wil M.P. van der Aalst, Mathias Weske, and Dolf Grünbauer. Case handling: a new paradigm for business process support. *Data & Knowledge Engineering*, 53(2):129–162, 2005.

[VRH04] Peter Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. The MIT Press, February 2004.

[WCZM07] 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, 2007.

[Win86] Terry Winograd. A language/action perspective on the design of cooperative work. In *Proceedings of the 1986 ACM conference on Computer-supported cooperative work*, CSCW '86, pages 203–220, New York, NY, USA, 1986. ACM.

[WKFF12] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Crowder: Crowdsourcing entity resolution. *Proc. VLDB Endow.*, 5(11):1483–1494, July 2012.

[WKNL07] M. Wieland, O. Kopp, D. Nicklas, and F. Leymann. Towards Context-aware Workflows. In *CAISE'07 Proceedings of the Workshops and Doctoral Consortium Vol. 2*, 2007.

[WLK+13] Jiannan Wang, Guoliang Li, Tim Kraska, Michael J. Franklin, and Jianhua Feng. Leveraging transitive relations for crowdsourced joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 229–240, New York, NY, USA, 2013. ACM.

[Woo12a] Peter T. Wood. Query languages for graph databases. *SIGMOD Rec.*, 41(1):50–60, April 2012.

[Woo12b] Peter T. Wood. Query languages for graph databases. *ACM SIGMOD Record*, 41(1):50–60, 2012.

[YM94] Eric S. K. Yu and John Mylopoulos. From e-r to "a-r" - modelling strategic actor relationships for business process reengineering. In Pericles Loucopoulos, editor, *ER*, volume 881 of *Lecture Notes in Computer Science*, pages 548–565. Springer, 1994.

[ZB11] Yang Zhang and Gary E. Bolton. Social Network Effects on Coordination: A Laboratory Investigation. *SSRN eLibrary*, 2011.