DISSERTATION

Adaptation Techniques in large-scale Service-oriented Systems: Models, Metrics, and Algorithms

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaften

unter der Leitung von

Univ.-Prof. Dr. Schahram Dustdar
Institut für Informationssysteme
Abteilung für Verteilte Systeme
Technische Universität Wien

eingereicht an der

Technischen Universität Wien
Fakultät für Informatik

von

Mag. rer. soc. oec. Christoph Dorn

`c.dorn@infosys.tuwien.ac.at`

Matrikelnummer: 9825872
Alserstraße 32/27
A-1090 Wien, Österreich

Wien, September 2009

Kurzfassung

In den letzten Jahren gehen Menschen vermehrt ihren gemeinsamen Interessen online nach. Web-basierte Kollaborationsplattformen wie Facebook, Youtube oder Wikipedia haben enormen Zulauf erhalten. Diese Portale erlauben Zusammenarbeit in bisher ungeahnten Dimensionen. Interessensgemeinschaften entstehen ad-hoc, wachsen auf tausende Teilnehmer an und zerfallen schlussendlich wieder. Die zugrundeliegende Dynamik solcher Kollaborationen ist weitgehend unvorhersehbar und führt zu kontinuierlich wechselnden Systemanforderungen. Während Menschen sich an unterschiedliche Umstände vergleichsweise leicht anpassen können, passt sich Software von selbst, wenn überhaupt, nur eingeschränkt an wechselnde Bedingungen an. Diese Dissertation behandelt das Problem wie sich Software - speziell Web Services - an den Gesamtkontext und die Anforderungen von Massenzusammenarbeit anpassen kann.

Wenn tausende oder mehr technische und menschliche Entitäten zusammenarbeiten, kann kein einzelnes Element die Gesamtbedürfnisse erfassen. Infolgedessen erkennt niemand Situationen, welche die Umgestaltung des Gesamtsystems erfordern würden. Ohne entsprechende Anpassungstechniken läuft die Zusammenarbeit Gefahr ineffizient zu werden oder gar frühzeitig auseinanderzubrechen.

Diese Dissertation präsentiert Techniken auf drei Ebenen. Den meisten Einfluss auf erfolgreiche Zusammenarbeit haben Techniken, welche die Gesamtbedürfnisse feststellen und darauf aufbauend die benötigten Services bereitstellen. Daran anschließend werden Algorithmen beschrieben, welche es ermöglichen, dass die richtigen Services untereinander kommunizieren. Drittens stellen Kontextverteilungsmechanismen sicher, dass die Services die relevanten Kontextinformationen zur Adaption bekommen. Datenmodelle, Algorithmen und Prototypen sind an Hand von Simulationen sowie Experimenten mit Echtdaten eines web-basierten Diskussionsforums evaluiert.

Abstract

Over the past years, people enthusiastically took up web-based services such as Facebook, Youtube, or Wikipedia to pursue joint interests. Large-scale collaborations emerge in an ad-hoc fashion, have participants join in, and eventually dissolve again. Such dynamic collaboration changes result in constantly shifting system requirements. Humans can adapt to some extent to changing conditions, while software remains mostly rigid. Enabling system adaptation to meet these requirements is the main problem addressed in this thesis.

In large-scale socio-technical networks, neither service nor human entities are able to obtain a complete picture of the overall context, constraints, and requirements. Consequently, no single entity perceives the need for reconfiguration. Without proper adaptation techniques, collaborations yield poor performance and are prone to end prematurely.

In this thesis, we present a layered approach to adaptation techniques. Most importantly, infrastructure adaptation ensures provisioning of the required services. Subsequently, service adaptation techniques ensure interaction of the right services. Finally, we present techniques for delivering the relevant context information. We evaluate these contributions with a mixture of collaboration simulations and experiments on real-world data from an online discussion forum.

Acknowledgements

First and foremost, I would like to thank my advisor Prof. Schahram Dustdar for the great opportunity to carry out my thesis at the Distributed Systems Group. His continuous mentoring and supervision taught me the important aspects of conducting research. I greatfully value the freedom I had to explore multiple research directions.

I greatly appreciate the feedback of my second advisor, Prof. Harald Gall. His valuable comments and suggestions helped me to improve this thesis.

I would like to express my thanks to my colleagues, especially Daniel Schall and Hong-Linh Truong for exciting discussions on the various aspects of this work. Special thanks go to Florian Skopik for providing the raw slashdot dataset.

I am in debt to my family and girlfriend for their understanding and limitless support that enabled me to pursue my research interests to such extent.

Finally, I'm thankful for financial support from the EU FP6 project inContext (IST-034718). It provided a unique opportunity to place my research in such an informative project context.

Christoph Dorn
Vienna, Austria, September 9, 2009

For Karin

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past years we have observed a trend towards online collaboration. Web sites for social networking (e.g., Facebook, LinkedIn), collaborative tagging (e.g., Digg, Del.ici.us), content sharing (e.g., Youtube), or knowledge creation (e.g., Wikipedia) have attracted millions of users. People increasingly utilize such tools to pursue joint interests and shared goals.

The scientific community in particular comes to profit from a tight interweaving of social networks and technological networks (Jones, Wuchty, and Uzzi 2008). Barabasi (2005) highlights the tendency for research teams to grow in size. Guimera et al. (2005) describe the impact of social network dynamics on team performance. Scientific teams emerge in an ad-hoc fashion, gather the persons with the required expertise, conduct research, and dissolve again. At the same time as Internet technology is fostering such dynamic collaboration, recent efforts aim to turn research results and research tools equally (re)usable and composable (Foster 2005, Hey and Trefethen 2005, Buetow 2005). Service-oriented computing promises to bring the same flexibility to research collaboration as it does in the domain of enterprise collaboration.

Service-oriented Computing (SOC) is a distributed programming paradigm. A service exhibits a public interface that describes its functionality in a standardized fashion. Service compositions provide the aggregated capabilities of multiple services. SOC supports loose-coupling, thus enabling a service client to discover and rebind to another service exhibiting the same interface. In this thesis, we refer to systems comprising collaborating people and services as *Service Ensembles*.

The scientific community is one example where collaboration emerges in large-scale, heterogeneous systems. Kleinberg (2008) notices the opportunity to observe the dynamics and complexity of such systems that arise from the convergence of social and technical networks in general. Several papers discuss the network topology of large-scale, complex systems (McAuley et al. 2007, Gómez et al. 2008), and devise formalisms that simulate the creation of these systems (Alava and Dorogovtsev 2005, Lieberman et al. 2005). In contrast, system management is receiving notably little attention.

Due to scale, no single ensemble participant has a complete picture of the overall service ensemble. Consequently, the lack of tools for system management causes poor performance and slow reaction to a changing environment: promising collaborations dissolve prematurely, helpful services remain unavailable as nobody becomes aware of the demand. As a result, enabling adaptivity is a prime concern in service ensembles.

Context is a key factor to achieving adaptation in service ensembles. It describes capabilities, properties, and the environment of humans and services. To this end, context also models the interaction between humans, humans and services, and between services. This information gives rise to ensemble metrics. They describe high-level ensemble idiosyncrasies. Ensemble metrics provide important guidance to determine necessary adaptation actions. Subsequent execution of adaptation actions, however, is non-trivial as service ensembles inherently lack centralized control.

1.1 Motivating Scenarios

In service-oriented computing, we distinguish between client-driven or service-driven adaptation. In the first case, the client executes the appropriate adaptation strategies. A person exchanges, for example, a simple document store service for a high-performance cloud storage service. In the second case, a client merely invokes a service. A storage service provider monitors, for example, resource consumption and adapts accordingly by raising its storage capacity.

Both approaches exhibit considerable drawbacks as a result of limited, local information. Clients need to keep track of ensemble-wide requirements, which is hard, if not impossible, to achieve in a multi-organizational environment. Moreover, ensembles have become too complex to be adapted by human administrators (Huebscher and Mccann 2008). Services, on the other hand, need detailed information about their clients' goals. Adaptation actions, however, depend not only on individual clients but have to consider the clients interdependencies with other ensemble participants. The following scenarios highlight this problem in three real-world settings.

Scenario 1 - Providing the right services
Suppose a project report leader delegates the writing of various chapters to individual partners. The leader remains aware of these partners but has no means to observe any further delegations and collaborations these partners trigger within their respective organizations. Each participant in this ensemble perceives only a little part of the overall set of interactions.

Most participants will recognize the need for a document service, but none has the required information about which capabilities such a service ideally should provide. On the service side, a simple document store service remains unaware of the structure, purpose, and involvement of participants, their document artifacts, and applied services. Lacking such knowledge it cannot realize how to adapt, or even recognize that it might be entirely inappropriate for the underlying situation.

Scenario 2 - Utilizing the right services
A storage service receives a new document and has to decide where to send a copy for backup to. The service maintains a list of some available storage services—a subset of the existing storage services in the ensemble. These available storage services differ in their properties, for example location, capabilities, or owning organization.

The service client possesses no information on the policies and interactions that influence the distribution of data amongst the storage services. A single storage service, on the other hand, cannot consult neighboring services as services with different properties exhibit different interaction behavior.

Scenario 3 - Services doing the right thing
An ensemble participant utilizes a document search service to collect relevant documents for his/her underlying activity. The service has little additional information about what documents are relevant other than the keywords provided. It lacks knowledge on the user's interaction structure, the people s/he works with, the documents these collaborators created without the user's involvement, nor the context in which such documents where stored. The user, on the other hand, would have to communicate with his/her peers to obtain information on relevant documents. Additionally, in service ensembles that involve vast amounts of documents, a single participant has difficulties tracking the context of each document to reason about the relevance for the situation at hand.

1.2 Preview of Results

Our main contributions in this thesis are:

Infrastructure Adaptation Techniques include a model, algorithm, and framework to track ensemble-centric requirements and propose suitable services reconfigurations. A capability model describes service features and reconfiguration options. Ensemble metrics describe changes in the ensemble configuration and trigger reevaluation of requirements. We match service capabilities against requirements to identify the most fitting service composition.

Service Adaptation Techniques consist of a metric model and algorithm that evaluate the impact of service properties on service interactions. We introduce a service ranking algorithm that exploits these interaction trends.

Context-awareness Techniques comprise a context model—describing the properties and interactions between ensemble entities—and context distance metrics for establishing the most relevant context for ensemble participants to use in a given situation.

1.3 Structure

This thesis is structured as follows: Chapter 2 provides a review of related work. We discuss the three main research streams: context-awareness, autonomic computing, and service-oriented computing. Subsequently, Chapter 3 presents a concise problem statement, outlines the novelty of this thesis, and discusses the chosen approach.

The subsequent chapters 4 to 6 cover the main contributions of this thesis. Each chapter closes with a self-contained evaluation of the presented research results. Chapter 4 introduces the ensemble context model. We define context-based and interaction-based distance metrics to describe similarity (i.e., relevance) between ensemble entities. We compare the two metrics utilizing real-world data from an online discussion forum. Part of this chapter provides additional concepts for sharing context in mobile ensembles. Chapter 5 outlines the importance of ensemble metrics. We propose a new algorithm that evaluates the impact of service properties on ensemble service interactions. The discovered interaction characteristics are a central input for our novel service ranking algorithm. Simulation demonstrates performance, robustness, and scalability of our approach. Chapter 6 describes ensemble requirements tracking and subsequent adaptation. Our biased requirements clustering algorithm determines suitable service compositions. A tradeoff between optimal requirements fulfillment and minimum composition costs applies the similarity metrics discussed in Chapter 4. Experiments on data from the online discussion forum confirm the benefit of our clustering and service aggregation framework.

Subsequently, Chapter 7 discusses implementation-specific details. We provide service interfaces and technical mechanisms of ensemble management, context provisioning, and ensemble adaptation. Finally, Chapter 8 concludes this thesis. We summarize our results and provide a brief collection of open research ideas and questions.

Chapter 2

Related Work

In this chapter, we present the basic principles and building blocks which we utilize and extend in this thesis. There is no research domain single-handedly addressing the challenges of adaptation in service ensembles. The three most influential research streams are autonomic computing, context-awareness, and service-oriented computing (SOC).

Service ensembles contain both human and software elements. We, therefore, discuss related work from multiple viewpoints. We, additionally, outline the missing links required for realizing a unified adaptation approach, covering the software and human side. We briefly explain the role of the central research streams before we discuss related work in detail.

Context-awareness describes the ability of entities (human or software) to perceive the relevant aspects of their working environment (Morse et al. 2000, Dourish 2004, Baldauf et al. 2007). Instead of having the client (again, human or software) specify all relevant information, context frameworks provide such information to enable the entity to perform its function appropriately. The application domain and entity role determines what the relevant context is. As context is fundamental to adaptation, it becomes also a crucial factor in achieving *autonomic* adaptation (Salehie and Tahvildari 2009).

Service-oriented Computing in the scope of this thesis characterizes the underlying technical infrastructure. In a service ensemble all active entities are modeled and represented as service providers and service clients. Schall et al. (2008) provide models, mechanisms, and frameworks for unifying human and service interactions. In service-oriented environments, adaptation comes primarily in three forms: intra-service adaptation (service-driven actions), service selection (client-driven action), and service replacement (infrastructure driven actions).

Autonomic Computing provides a new paradigm for reducing the complexity of software systems. The central goal is reducing human control by turning software self-aware. Two broad design principles aim for such autonomous behavior. Autonomic systems implementing a feedback loop (Kephart and Chess 2003) require a global view of the system to enforce optimal adaptation actions (Di Nitto et al. 2008). Socially and biology-inspired systems exploit emerging phenomena (Babaoglu et al. 2006). The collective behavior of system elements yields global desirable goals purely based on local information.

Context models describe the structure of relevant information for adaptation. We analyze their comprehensiveness of covering the various ensemble adaptation requirements. Context frameworks capture, reason on, and ultimately provide the actual context to adaptive services. Subsequently, we outline current techniques that aim for service selection and aggregation. Finally, we discuss related work on autonomous adaptation.

2.1 Context Models and Frameworks

The definition of context depends very much on its application area. Bazire and Brézillon (2005) collected 150 definitions from various areas of research. The definition by Dey and Abowd (2000) is widely adopted in the domain of computer science:

> [...] any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

We further extend our adapted definition in Dorn and Dustdar (2007) to highlight the nature of service ensembles:

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, object, or *aggregation* thereof that is considered relevant to the interaction between a user and a service as well as *between services*, including the user and services themselves.

The difference seems trivial, almost negligible, but has fundamental implications on the modeling and provisioning of context. First, relevant context is not simply a set of individual entities, but rather comprises an aggregation of multiple, heterogeneous elements including their interaction characteristics. Second, context extends beyond the basic relationship of human and service (i.e., [user],[has available],[service]). Context needs to describe the dependencies between services and humans as well as in-between service alike.

A context model enabling adaptation in service ensembles requires three different views:

Entity-centric Context captures the situation of individual entities. Traditional models describe human-centric context such as location, devices, presence information, time, and action (de Freitas and da Graca 2005, Belotti, Decurtins, Grossniklaus, Norrie, and Palinginis 2004, Gu, Pung, and Zhang 2005). Some models capture only parts such as the COBRA-ONT (Chen, Finin, and Joshi 2003) ontology describing an agent's location and actions, Amundsen and Eliassen (2008) describing user and devices, or Anagnostopoulos, Mpougiouris, and Hadjiefthymiades (2005) involving only location. Ramparany et al. combine actions, devices, user preferences, and weather conditions (Ramparany, Euzenat, Broens, Bottaro, and Poortinga 2006). Yang et al. focus on user preferences specific to services such as cost, speed, QoS, and mobility (Yang, Mahon, Williams, and Pfeifer 2006). They also consider proximity of services to increase the performance of service compositions.

Other context models focus purely on service aspects. Maamar et al. introduce context to describe available services instances, their execution status, and expected termination of service execution instances (Maamar, Kouadri, and Yahyaoui 2004, Maamar, Benslimane, Thiran, Ghedira, Dustdar, and Sattanathan 2007). Casati et al. model service execution quality in the context of a specific process (Casati, Castellanos, Dayal, and Shan 2004). Mrissa et al. suggest contextual annotation of service interfaces to allow for correct interpretation and mediation (Mrissa, Ghedira, Benslimane, Maamar, Rosenberg, and Dustdar 2007).

Most models contain the concept of an *activity*. However, the general notion of such activities is usually limited to linking a user to an action (e.g., a user is walking, reading, attending class) or a service to an action (Bardram 2005).

Activity-centric Context puts individual actions into a larger perspective. They describe the flow and dependencies of actions, thereby joining people, services, resources, and artifacts in a temporal manner. Dustdar first introduced the concept of activities in the domain of ad-hoc processes in Caramba (Dustdar 2004). Specifically, he focuses on process awareness for enabling users to perceive their role in the context of the overall activity flow.

Other work recognizes the importance of activity context for task-awareness (Moody, Gruen, Muller, Tang, and Moran 2006), self adaptation (Garlan, Poladian, Schmerl, and Sousa 2004, Sousa, Poladian, Garlan, and Schmerl 2005), or resource recommendation (Ning, Gong, Decker, Chen, and O'sullivan 2007). These approaches, however, miss out on the potential of interaction analysis. Relations between activities, resources, and humans are configured during bootstrapping and remain unchanged thereafter.

Ensemble-centric Context describes ensemble characteristics that emerge at a global level. Modeling of such aspects has not received much attention. Related work is spread across multiple niches. Research in the domain of collaborative working environments covers context such as availability and distribution of members, organizational structure, or communication means. Vieira, Tedesco, and Salgado (2005) include interaction and organization aspects in their context ontologies but include activities only as simple tasks without embedding them in an underlying activity flow.

Social network analysis investigates interaction characteristic of online communities. Information that potentially serves as context (e.g., Bird, Gourley, Devanbu, Gertz, and Swaminathan (2006) Valverde and Solé (2006)) is usually not available in near-realtime, nor does it include aspects beyond human-to-human communication.

2.1.1 Context Provisioning in Mobile Environments

da Rocha and Endler (2006) have proposed context granularity as an important part of distributed context-aware systems. Most research efforts on mobile context frameworks, however, tend to focus on architectural aspects. Some of the following frameworks exhibit some notion of context hierarchy, but none of these approaches explicitly enables granular access to context information.

Biegel and Cahill (2004) present a framework for developing mobile, context-aware applications. They introduce the concept of a context hierarchy. However, their hierarchy has the notion of a task tree rather than structuring context information into various levels of detail.

Web Service Context (WS-Context) (Little, Newcomer, and Pavlik 2004) is a specification proposed by OASIS to describe the context of an activity—composed of several Web services. WS-Context defines methods to pass context by value or just by reference. In the latter case, the receiving service obtains the actual information from the *context manager service*. Context information itself can be structured hierarchically as WS Context includes an optional element, which refers to the parent context.

The service-oriented context-aware middleware (SOCAM) by Gu, Pung, and Zhang (2004) provides push- and pull-mechanisms for retrieving context information. However, such information is only gathered but not forwarded to other services and solely provided to the applications build on top of SOCAM.

Costa, Pires, van Sinderen, and Filho (2004) designed a platform for mobile context-aware applications. Context information is shared by subscribing to this platform using the WASP Subscription Language (WSL).

The Solar middleware by Chen and Kotz (2002) provides a platform for context-aware mobile applications consisting of one star and several planet nodes. Client applications need not collect, aggregate or process context themselves but subscribe to context changes at the central star.

Other subscription enabled context frameworks include work by Sørensen, Wu, Sivaharan, Blair, Okanda, Friday, and Duran-Limon (2004) and Hinze, Malik, and Malik (2005).

A comprehensive survey on context-aware systems by Baldauf, Dustdar, and Rosenberg (2007) provides additional in-depth details on architecture, context model, and context life-cycle.

2.2 Context Selection and Ranking

In the scope of this thesis, we treat selection as a problem of choosing the best service (or resource) given a set of metadata (i.e., any type of information about a service other than the service interface description). In this process, ranking constitutes the penultimate step, right before the final selection amongst the top rated elements. Particular to service selection, we do not consider interface matching or mediation as part of this problem.

Extensive research efforts focus on service selection based on Quality-of-Service (QoS) attributes (Yu and Lin 2005, Wang, Vitvar, Kerrigan, and Toma 2006, Rosenberg, Leitner, Michlmayr, Celikovic, and Dustdar 2009). In pure SOA-environments, Vu, Hauswirth, and Aberer (2005), Maximilien and Singh (2004), and Maximilien and Singh (2005) extend this approach and include trust metrics. Skopik, Schall, and Dustdar (2009) introduce trust to mixed service-oriented systems for selection of both humans and services.

In contrast to automatically derived metrics, tagging-based frameworks —e.g., Tai, Desai, and Mazzoleni (2006), Desai, Mazzoleni, and Tai (2007)—and recommendation-based frameworks—e.g., Manikrao and Prabhakar (2005), Silva-Lepe, Subramanian, Rouvellou, Mikalsen, Diament, and Iyengar (2008)—collect meta-data directly from service users.

Approaches in the middle between these two extremes concentrate on past invocations. Birukou, Blanzieri, D'Andrea, Giorgini, and Kokash (2007) analyze similar requests, while Casati, Castellanos, Dayal, and Shan (2004) observe the context of previous successful processes to recommend suitable services.

Ning, Gong, Decker, Chen, and O'sullivan (2007) suggest a goal driven approach to resource recommendation. Based on the person's use of resources (i.e., the context), the systems infers his/her current goal and suggests additional suitable resources.

The dynamic ranking approach by (Bottaro and Hall 2007) comprises contextual scopes, filters, and scoring functions. Context itself is limited to information on services (e.g., service state, capabilities, QoS) and traditional context such as location.

2.2.1 Ranking Functions

Ranking criteria based on QoS or trust metrics describe aggregations of raw data associated with individual elements (i.e., service, humans, resources). In contrast, service ensemble context includes interaction data between humans and services. Ranking functions on interaction data make heavy use of graph metrics.

A prominent example of a graph-based global importance metric is Google's page rank (Brin and Page 1998). A context-aware version (Haveliwala 2003) yields total ranks by aggregating search-topic-specific ranks. Inspired by the page rank algorithm, Schall (2009) applies interaction intensities and skills to rank humans in mixed service-oriented environments.

Our approach differs in two important aspects. First, we do not apply global ranking. Ranking of elements in our k-partite action graph happens from a particular perspective (i.e., we rank elements as seen from one chosen element within the graph). Second, similarity of two same-type elements (e.g., a person) is not merely restricted to direct interaction between two elements. Similarity derives from their involvement in joint activities, use of common resources, or modification of the same artifacts.

2.3 Service Composition

Service composition describes the process of combining multiple services to provide a particular functionality which none of the individual services can offer by itself. Service composition relies on service selection and ranking to determine the most suitable candidates for aggregation. In a survey on Web service composition, Dustdar and Schreiner (2005) highlight a number of composition concerns such as message coordination between composed services, transaction properties, context-awareness, and execution monitoring.

The fundamental process underlying most composition approaches consists of mapping abstract requirements (i.e., capabilities) onto concrete service instances. These abstract requirements reside at various levels of granularity and need to be broken down into sub-requirements before the ultimate mapping occurs. Most approaches exhibit the implicit assumption that each requirement identifies one service type at the end of the mapping process. Subsequently, services of each particular type get ranked according to QoS metrics, policies, and context. The top scored services yield the composition.

Maamar et al. identify Web services, policies, and context as the key components to Web service composition (Maamar, Benslimane, Thiran, Ghedira, Dustdar, and Sattanathan 2007). They introduce a multi-level approach comprising component level (service capabilities and interfaces), composite level (service discovery and aggregation), semantic level (service interface heterogeneities), and resource level (service runtime environment). Each level is associated with a corresponding context type (Maamar, Kouadri, and Yahyaoui 2004). Such context defines which and how policies control the transition between levels. At the composite level, service chart diagrams and state chart diagrams (Maamar, Benatallah, and Mansoor 2003) control how individual services are combined. These state charts in combination with location and time context are also applied by Sheng, Benatallah, Maamar, Dumas, and Ngu (2004) to achieve personalized service composition.

Mrissa et al. propose context-sensitive semantic description of service interfaces to allow for mediation of data heterogeneities in BPEL processes (Mrissa, Ghedira, Benslimane, Maamar, Rosenberg, and Dustdar 2007). Hull and Su (2005) provide an overview of tools for composite Web services.

Baresi, Bianchini, Antonellis, Fugini, Pernici, and Plebani (2003) describe the context-aware composition of communication services. They consider user location and QoS metrics to assemble the best combination and configuration of services on fixed and mobile stations. Compositions are modeled as generic micro flows, which are adapted to the execution context during runtime.

In the project Daidalos, Yang, Mahon, Williams, and Pfeifer (2006) apply an ontology to identify required services fulfilling a user's task. User context and preferences are key for adapting the composition as needed.

Quitadamo, Zambonelli, and Cabri (2007) demonstrate a knowledge-network-driven approach to service selection and aggregation. They link semantic models to input and output of services within the scope of an *enzyme*. Such enzymes represent data transition between ontology concepts, rather than workflows. They aggregate when a single enzyme cannot provide the necessary functionality.

Our work differs from traditional composition approaches in that we specifically focus on providing suitable service agglomerations. Service requirements describe what capabilities are required but not how the respective services should be composed.

The approaches introduced above consider only a subset of an ensemble context during composition. Adaptation and selection criteria usually build upon QoS metrics or context about the service execution environment. Involved user context comprises mostly location, devices, and preferences. None of these efforts evaluate the complete setting of humans and services in an ensemble.

2.4 Autonomic Service Adaptation

In recent years software management turned increasingly difficult as IT systems become ever more complex. Systems do not only grow bigger in terms of lines of code. Their interconnection and dependency on other systems, often subject to different authorities, adds to the overall dilemma. In dynamic environments that yield changing requirements and conditions, it becomes impossible to manually execute management tasks such as (re)configuration, maintenance, optimization, protection, or recovery. In 2001, IBM introduced the concept of *Autonomic Computing* (Horn 2001). The central idea to autonomic computing is self-management of software components.

The initial four self-* properties are self-configuration, self-healing, self-optimization, and self-protection (Kephart and Chess 2003). Self-configuration envisions components to install, setup, and integrate themselves solely based on some high-level policies. Self-healing seeks automatic discovery of internal, undesirable situations, and devises plans to recover from them. Self-optimization monitors the system status and adjusts parameters to increase performance when possible. Finally, self-protection aims for detection and mitigation of external threats (White, Hanson, Whalley, Chess, and Kephart 2004).

The conceptual architecture for autonomic computing envisions an autonomic manager observing and controlling a managed element (see Figure 2.1), thereby creating an autonomic element (Kephart and Chess 2003, IBM 2005). The autonomic manager consists of five key components: *Monitoring*, *Analysis*, *Planning*, and *Execution*, all of which apply a common set of *Knowledge* (i.e., the MAPE-K cycle).

Monitoring obtains information about the managed element and its environment. Monitoring forwards aggregated and cleaned data to the analysis component.

Analysis evaluates the current situation and determines if counteractions are required. If adaptation is required, planning becomes involved.

Planning determines how to react to a given situation. It devises the concrete adaptation measures.

Execution enforces the required adaptation steps. Actions apply to the managed elements but also include notification of or escalation to supervising autonomous entities.

Knowledge maintains information on the autonomous element's embedding in its greater environment. It provides guidance for the other four elements in form of requirements, rules, domain knowledge, and policies.

These five aspects are fundamental to any autonomous system—albeit some works assign different names to these steps (Parashar and Hariri 2004, Dobson et al. 2006). They form a feedback loop together with the managed element.

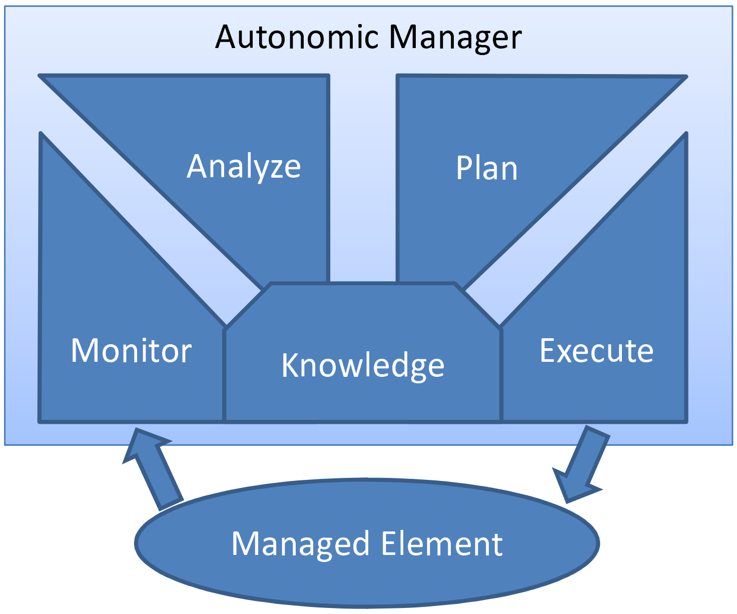


Figure 2.1: Autonomic element: an autonomic manager observing and controlling the managed element.

The initial concept of autonomic elements interacting with each other to appropriately self-adapt works well in environments that yield little interaction between autonomic elements and rather simple compositions of autonomic elements. In such environments the main focus is adaptation of the managed element.

The *Autonomic communications* research domain addresses the challenges arising from managing communication networks (Dobson, Denazis, Fernández, Gaïti, Gelenbe, Massacci, Nixon, Saffre, Schmidt, and Zambonelli 2006, Schmid, Sifalakis, and Hutchison 2006). Network infrastructures lack a single point of control, yield highly dynamic topology changes, and address conflicting client requirements. We cannot directly apply the autonomic feedback loop to achieve self-* properties.

Emergence-based adaptation describes a completely decentralized approach based on collective interaction phenomena (Figure 2.2). Desirable behavior emerges from a group of interacting elements. Each element follows a set of rules, none of which directly accounts for the overall behavior. The relevant fundamental characteristics of emergence are according to Wolf and Holvoet (2004):

Micro-Macro Effect: actions carried out by individual elements (micro-level) result in a specific behavior at the macro-level of the system.

Radical Novelty: from a top-down view, the macro-level behavior cannot be explained by decomposing the system into the individual elements. From a bottom-up view, the rules determining the micro-level actions do not describe the macro-level result. The overall behavior is only implicitly described at the micro-level. The individual elements remain unaware of their global goal.

Interaction: behavior of individual elements must include interaction with other elements. Such interaction can be direct or indirect (i.e., an element observes and reacts to the actions of another element).

Local View: in large-scale systems, individual elements cannot keep track of all other elements. The view of the 'world' is reduced to a subset of neighboring elements. Subsequently, any behavioral rules must not require complete awareness.

Decentralized Control: emerging behavior arises without any form of central control. Control mechanisms perceive only local information and enforce local actions.

Self-organizing, emergent systems are often based on multi-agent technology (Serugendo et al. 2003, Babaoglu et al. 2004, Wolf and Holvoet 2005). As this thesis concentrates on service ensembles, we analyze SOA-related autonomic research efforts with respect to supported context scope, adaptation architecture, and dynamic context relevance. In service ensembles, autonomous adaptation must not restrict supported context to software system elements. Context needs to describe the tight interdependencies between humans and services. Furthermore, adaptation needs to consider the overall ensemble configuration—not only the individual user context or service context. However, tracking of detailed context information for large-scale ensembles is not feasible. Any adaptation architecture, consequently, needs to remain decentralized to some degree. Finally, dynamic context relevance describes the ability to continuously identify (and subsequently adapt to) the most significant impact factors in a system. To the best of our knowledge, current approaches fail in at least one of these three concerns.

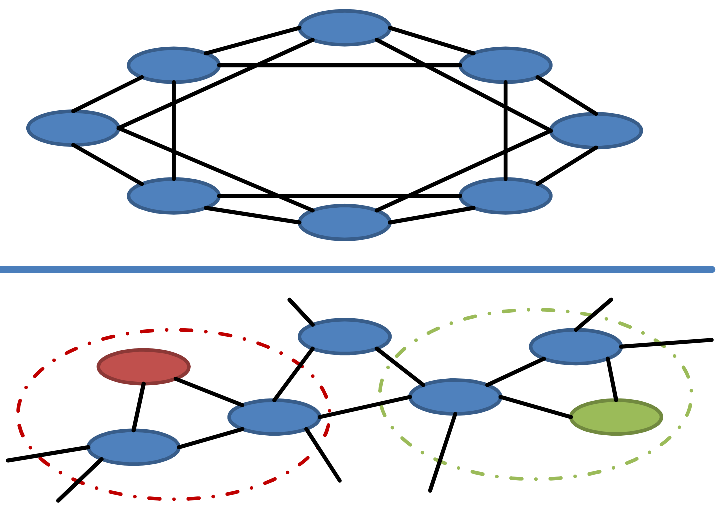


Figure 2.2: Emergence: individual elements interact (black lines) with their peers purely based on local information (dashed circles). These actions at the micro-level result in desirable outcome on the macro-level.

Current general-purpose autonomic techniques and toolkits such as Sterritt, Smyth, and Bradley (2005), Bigus, Schlosnagle, Pilgrim, Mills, and Diao (2002), or IBM (2004) primarily apply context about the software environment. These frameworks adhere to the basic MAPE-K feedback loop, limiting the application of user context to properties such as location or device. In Self-Configuring Socio-Technical Systems Bryl and Giorgini (2006) describe a multi-agent system reacting to dynamic reconfiguration needs. They claim to apply both local and global information. Unfortunately, they lack details on the extent of global information or specific aggregation mechanisms to support scalability in large-scale systems. Goal-driven adaptation of service compositions such as Greenwood and Rimassa (2007) or Yu and Lin (2005) consider exclusively service context and require complete control over the aggregation.

Following research efforts yield some form of decentralized control, but completely lack user-centric context. Andreolini, Casolari, and Colajanni (2008) exploit load trends for autonomic request forwarding between geographically distributed systems. Colman (2007) proposes a hybrid approach to self-organization services through hierarchical structuring of autonomic managers and services. The autonomic manager monitors and controls all composed services, thereby severely limiting the size of manageable service compositions. Jennings, van der Meer, Balasubramaniam, Botvich, Foghlu, Donnelly, and Strassner (2007) discuss an architecture for autonomic management of communication networks. They suggest applying the MAPE-K cycle to a complete set of entities, thus limiting the architecture's applicability to domains exhibiting a central set of goals.

Although autonomic computing is a well established paradigm for self-adaptiveness (Hariri, Khargharia, Chen, Yang, Zhang, Parashar, and Liu 2006), most systems (Huebscher and Mccann 2008) still apply a stable set of impact properties.

Saffre, Tateson, Halloy, Shackleton, and Deneubourg (2008) present an algorithm that results in self-organizing behavior of services. Membership properties enable the algorithm to achieve the desirable behavior using again only local context information. However, the type and impact of context information is defined a-priori.

Dynamically identifying the most relevant factors for self-adaptation includes research by Zhang and Figueiredo (2006). Their Bayesian network-based autonomic feature selection, however, focuses exclusively on service-internal measurements and thus neglects any form of interaction metrics. Sterritt, Mulvenna, and Lawrynowicz (2004) make the case for behavioral knowledge from which to compute metrics, but they remain at a general activity-focused level, not considering other ensemble aspects. Marinescu, Morrison, Yu, Norvik, and Siegel (2008) measure the importance of properties for system self-organization, but focus on the impact of simulated gene diversity.

Chapter 3

Problem Statement

The principal research question is how to enable services to autonomously adapt to the dynamic changes in service ensembles. A key problem is achieving adaptation based on the overall ensemble requirements—not just adaptation based on the needs of individual elements.

A *Service Ensemble* is composed of humans and services—the active ensemble entities. Interaction in service ensembles includes people communicating with other people, people utilizing services, service invoking other services.

Service ensembles are amalgamations of social and technical systems. An ensemble cannot be regarded as a pure social system, as services have great impact on how people interact. Services determine how people are able to coordinate, communicate, and carry out their joint work. Neither can a service ensemble be regarded as a pure technical system. The social structure yields great influence on required service capabilities. Groups that exhibit great trust amongst members want to collaborate more freely and unstructured than groups that follow a rigid organizational structure.

In an ensemble, each entity maintains connections to a neighboring subset of all entities due to scale. It observes changes only in its vicinity. Emerging phenomena that arise from the complete set of interactions cannot be observed by an individual at all. Thus, an entity applies only limited, local information when deciding what actions to execute next.

Ensembles grow from entities belonging to multiple organizations. There exists no central authority that controls growth and evolution of an ensemble. It emerges from the common goals its participants share. Changes occur as people shift their interests, as people leave the ensemble and new ones join in. Technical entities cause changes to equal extent: new services arise, existing service evolve, and some services disappear. These dynamic changes require constant adaptation to keep the ensemble working.

Interaction in service ensembles adheres to environmental constraints. At one point the organizational structure determines best interaction partners. At other times, location is most influential on communication patterns. The impact of the various aspects governing entity behavior shifts continuously as ensembles dynamically evolve. The impact scope of such aspects is often global. Individual entities find it non-trivial to recognize such trends based on local information only.

In the scope of this thesis, adaptation refers to the reconfiguration of an entity in order to adjust to changing environmental conditions. To this end, adaptable entities include services, humans, and aggregations thereof. Adaptation actions include remodeling a composition, selecting a different communication partner, or exhibiting different internal behavior. The adaptation techniques in this thesis target primarily services. However, they apply in a generic form to humans as well.

The main achievements necessary for enabling adaptation in service ensembles are:

- the ability to describe and detect overall ensemble requirements. Effective adaptation relies on comparison of actual and desirable ensemble configuration on a global level.
- an infrastructure supporting adaptive services as services cannot trigger reconfiguration, replacement, or deployment themselves.
- services exhibiting adaptive behavior to continuously react to changes in their local environment.
- mechanisms providing the relevant context required for adaptation.

3.1 Analysis of Related Work

Adaptation techniques for service ensembles remain largely unexplored. In Figure 3.1, we place related research domains in perspective to this thesis. On the *Focus* axis, we distinguish between research focusing on either human and social aspects or on technical aspects. On the *Scope* axis, we separate approaches that focus on local context information only—thereby achieving local optimization only—and approaches that require complete control and context to achieve global optimization.

Context models describe either social aspects such as personal preferences or focus on service-centric aspects. Activity-centric context takes a central role in this thesis, but is insufficient by itself for adaptation. Almost no model or metrics describe large-scale ensemble context. Analysis of social networks provides insight into the interaction structure of humans but addresses no technical concerns.

QoS-based and goal-driven selection techniques focus entirely on technical aspects. Goal-driven approaches consider user preferences and context, but remain ignorant of human interactions. Trust-based selection considers merely the interaction between individual elements. In contrast, global importance ranking requires complete interaction information to identify important ensemble entities.

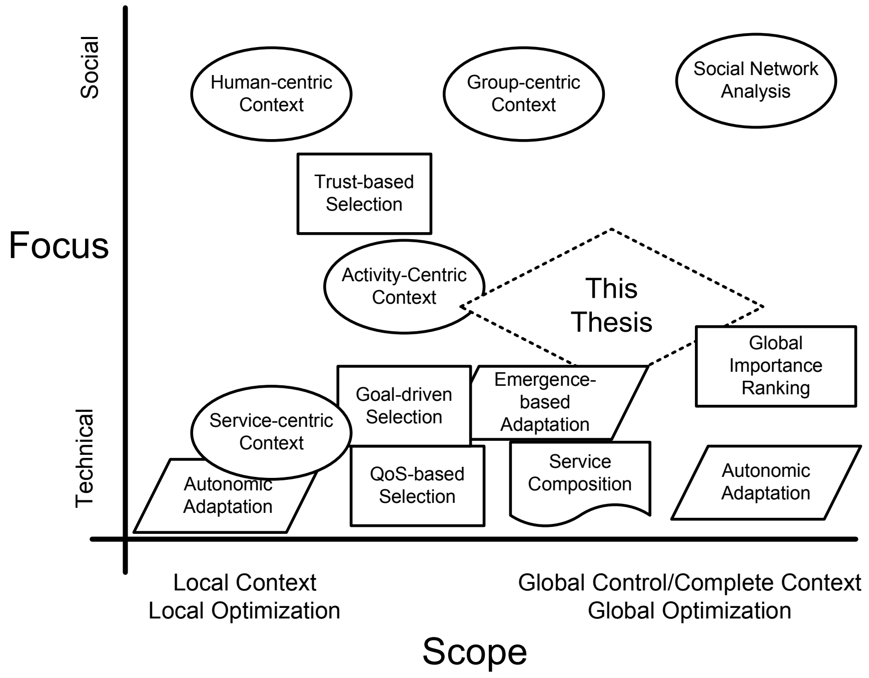


Figure 3.1: Related Work: Ellipses depict context models; rectangles depict (service) selection, respectively ranking techniques; documents represent composition mechanisms; and trapeziums represent adaptation techniques. The central diamond defines the research area of this thesis.

Research in service composition almost exclusively concentrates on the aggregation of technical services. Although some user context is applied during service selection, social interaction aspects yield no impact on the final composition.

Autonomic adaptation reside on both ends of the scope axis. Frameworks implementing the MAPE-K cycle achieve local optimization when utilizing local context, but require complete control (and context) to achieve global optimization. Emergence-based techniques are well suited for large-scale systems. They exploit interaction between elements to achieve overall desirable behavior based on local information only. Emergence based frameworks—mostly agent based—require a central authority to a-priori configure the interaction rules. This approach is not feasible in service ensembles without centralized control.

3.2 Relevance to Real-World Problems

Current large-scale collaborative environments remain mainly simple and unstructured. Our mechanisms allow for more complex collaborations on a larger scale. Requirements tracking on a global level promises to improve efficiency of large-scale ensembles. Individual workers become increasingly aware of their overall ensemble requirements. Additional services for coordination, communication, and execution are deployed just-in-time when needed. Adaptation techniques ensure the configuration of role-based resource access strategies for ensembles exhibiting a growing number of involved organizations, while they focus on asynchronous communication and work monitoring services for ensembles that spread over multiple time-zones.

The adaptation techniques in this thesis apply to collaborative working environments (CWE) in general. The introduction listed some motivating scenarios from the domain of scientific collaboration. Potential application domains, however, also include Enterprise Interoperability, where efficient interaction between small- and medium-sized companies becomes ever more important.

3.3 Approach

3.3.1 Assumptions

The following assumptions are crucial in putting our approach into perspective. Humans and services participate in multiple ensembles simultaneously. As we introduce models, algorithms, and our framework, we focus only on one ensemble instance throughout this thesis for sake of clarity. This includes entity interactions, context information, and entity properties. Amongst these, context information plays a fundamental role for adaptation. While we present corresponding models and raw data extraction techniques in this thesis, we refer the reader interested in the actual context sensor logic to the inContext project report D2.2 (Dorn, Polleres, and Yi 2008).

Replacability of services is a fundamental problem in service-oriented systems. Existing research work—e.g., Mrissa, Ghedira, Benslimane, Maamar, Rosenberg, and Dustdar (2007)—provides viable approaches upon which we build without going into detail. Specifically, we assume existing data mappings between incompatible services interfaces.

Other general aspects include security, reliability, integrity, and performance of services. These cross-cutting concerns would easily fill a thesis on their own. Here they remain out of scope. Finally, a graphical user interface and respective integration with services remains unaddressed. The inContext collaboration web portal (inContext Consortium 2008) demonstrates a possible approach.

3.3.2 Adaptation Methodology

Service ensemble characteristics and respective challenges require addressing adaptation on multiple levels. We achieve the highest impact by providing the most suitable services. These services have to continuously adapt to provide their capabilities effectively. Subsequently, they need the relevant context information. Figure 3.2 outlines this approach.

Adaptive Infrastructure derives and analyzes ensemble requirements. Comparison of current requirements and deployed service capabilities highlights potential adaptation

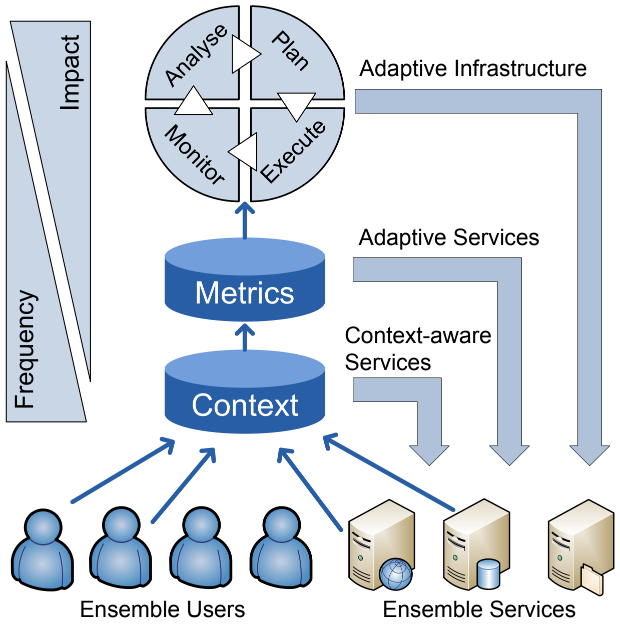


Figure 3.2: Approach

actions. Along these lines, the infrastructure recommends service deployment, undeployment, replacement, and reconfiguration. Reconfiguration explicitly addresses switching to a different adaptive behavior. At the infrastructure level, algorithms only decide on the best adaptation strategy. The actual adaptive behavior is internal to the implementing service.

Infrastructure-based adaptation targets long-term effects. Analysis of the overall ensemble requires aggregation of entity interaction information. Scale and complexity of service ensembles limit this process' execution frequency.

We discuss architecture, components and implementation of an adaptive infrastructure framework in Chapter 6 and Chapter 7, respectively. Tracking of requirements includes ensemble-specific metrics, which we introduce in Chapter 5.

Adaptive Services are implicitly aware about the ensemble's requirements through their capabilities and configuration. However, they lack an overall picture of all relevant aspects. Neither can they trace these aspects. To execute their adaptation strategies, they need to be context aware.

In Chapter 5, we provide a self-stabilizing algorithm to guide newcomers when joining a service ensemble.

Context-aware Services know about the common context in which they are used and apply the correct context information. Context use is very frequent, but a service's view remains limited to a neighboring set of ensemble entities. Subsequently, services adapt for short-term effects with limited scope.

Chapter 4 presents a context model describing both human and service aspects in ensembles. Our relevance-based context sharing algorithm ensures services are working with the right set of context information.

3.4 Publications

Parts of this thesis are published as journals, conference papers, workshop papers, and technical reports. Specifically, we disseminated the following main contributions.

Context Model: We discussed core context aspects in service ensembles at the DMC workshop (Dorn, Schall, Gombotz, and Dustdar 2007). These first design considerations guided the implementation of the context model provided in the technical report (Dorn, Polleres, and Yi 2007) (revised in (Dorn, Polleres, and Yi 2008)). At the 37th EUROMICRO conference, we demonstrated the role of the core activity model for self-adaptive collaboration services (Schall, Dorn, Dustdar, and Dadduzio 2008). Section 4.1 presents a detailed description and discussion of the individual context model elements.

Addressing the needs of mobile ensembles, we explored the potential of hierarchical context structures (Dorn, Schall, and Dustdar 2006). Section 4.5.1 specializes on results of context modeling efforts for mobile environments.

Context Provisioning: The basic context ranking algorithm (introduced in Schall, Dorn, Dustdar, and Dadduzio (2008)) utilized two ranking criteria. Detailed implementation details are given in the technical report (Casella, Dorn, Polleres, and Yi 2008). In Section 4.3 we revisit the ranking algorithm and extend on criteria and distance functions. For mobile environments, we additionally enable context sharing based on hierarchies. Section 4.5.2 presents the mechanisms previously outlined in a special edition of the Distributed and Parallel Databases journal (Dorn and Dustdar 2007).

Adaptation Mechanisms: We explored emergent metrics describing service ensembles in (Dorn, Truong, and Dustdar 2008). This paper presented an early version of the Property Distribution Entropy (PDE). More Activity-specific metrics in Dorn, Schall, and Dustdar (2008) highlight the potential of dynamically choosing the relevant aspects for service adaptation. Self-stabilizing algorithms based on a revised PDE definition are discussed and evaluated in detail in Chapter 5. Our results will be presented at ECOWS 2009 (Dorn, Schall, and Dustdar 2009b).

Adaptive Infrastructure: We described the infrastructure adaptation process including capability model and gracefully degrading matching algorithm in a paper (Dorn, Schall, and Dustdar 2009a) submitted to an international conference and are awaiting notification.

Other publications related to this thesis but addressing different challenges are:
- Schall, Dorn, Truong, and Dustdar (2008) outlining general techniques for facilitating integration of humans and services in ensembles.
- Schall, Gombotz, Dorn, and Dustdar (2007) presenting specific mechanisms to integrate human and services in mobile environments.
- Reiff-Marganiec, Truong, Casella, Dorn, Dustdar, and Moretzki (2008) and Dorn, Dustdar, Giuliani, Gombotz, Ning, Perray, Schall, and Tilly (2007) describing the inContext platform.
- Gombotz, Schall, Dorn, and Dustdar (2006) investigating interaction patterns for sharing context.

Chapter 4

Ensemble Context Provisioning

Context describes the underlying conditions and circumstances of a service ensemble. To support adaptation, we need to model the significant ensemble aspects, capture the corresponding context information, and provide the subset of the overall context relevant for the particular ensembles adaptation mechanisms. To this end we investigate in this chapter:

Ensemble Context Model detailing the context of humans, services, artifacts, resources, their coordination aspects, and fine-grained interactions.

Raw Context Capturing Mechanism extracting the fundamental configuration changes and interaction events that represent current constraints and conditions in a service ensemble.

Context Ranking Techniques recommending the most significant context elements based on interaction-centric distance metrics. Such metrics describe the context-sensitive similarity between context entities (i.e., people, services, artifacts . . .).

Mobile Context Provisioning proposing a hierarchical context modeling and sharing framework tailored to the device and network constraints in mobile service environments.

4.1 Context Model

In (Dorn, Schall, Gombotz, and Dustdar 2007), we outline the relevant aspects for modeling the context of service ensembles. The five main aspects comprise *Location*, *Organization*, *Activity*, *Human (Interactions)*, and *Resources*. We discuss location related details in Section 4.5.1 in the scope of context provisioning for mobile service ensembles.

The context model consists of a set of submodels describing semi-static configurations and relationships (e.g., organizational structure, humans, services, activity hierarchies). In addition, we model the highly dynamic actions captured in the service ensemble. Specifically, the *Action* model captures actions exactly as they are carried out, providing an event history. In contrast, the extended *FOAF* model, the *Activity* model, and the *Resource* model specify the semi-rigid ensemble configuration.

4.1.1 Entity Model

Services and humans are the active elements in service ensembles. Organizational and informal structures affect the interaction between and composition of these elements. Thus, we need to model both the background from which entities join an ensemble and the ensemble structure itself.

We extend the FOAF (Friend-of-a-Friend) model which describes relations between people on the web. The original concepts fit roughly into five categories listed on `http://xmlns.com/foaf/spec`, namely *FOAF Basics*, *Personal Info*, *Online Accounts and Instant Messaging*, *Project and Groups*, as well as *Documents and Images*. We reuse the following core concepts from *FOAF Basics*:

Agent represents any active entity in an ensemble such as a *Person*, an *Organization*, or a *Group*. An *Agent* can be member of a *Group* and exhibits a *mbox* (respectively *mbox_sha1sum* for privacy reasons) for identification purposes.

Person subclass of *Agent* represents any human being. In this thesis we describe a *Person* by *firstName*, and *family_name* only. These properties do not serve as unique identifier; instead we apply the agent's *mbox* property.

Organization subclass of *Agent* represents semi-stable aggregations of agents such as organizations or societies.

Group subclass of *Agent* applicable for ad-hoc or informal collection of agents. In contrast to an *Organization* a *Group* can contain multiple *Agent*, thereby creating a tree hierarchy.

The original FOAF concepts insufficiently describe all active ensemble entities. We also need to address the organizational aspects of services.

Service provides capabilities to a set of agents, belongs to an agent (the provider), and refers to its representation as a resource in the Resource model.

Fundamentally, a service ensemble consists of a set of agents combining both humans and services. Figure 4.1 visualizes the Ensemble Entity Model, including only the reused FOAF concepts.

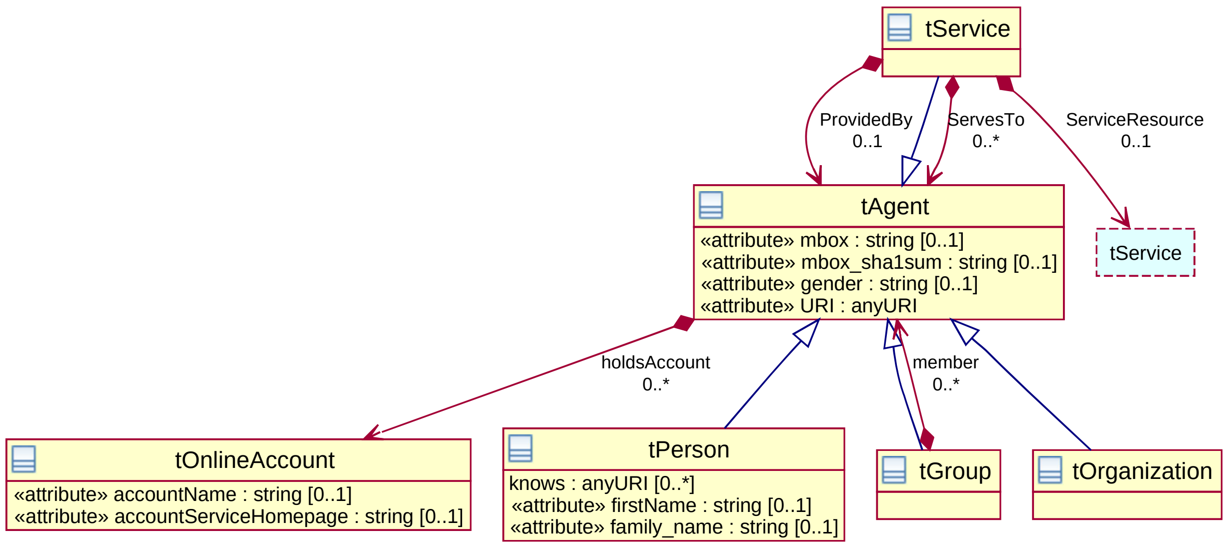

tService
ProvidedBy
0..1
ServesTo
0..*
ServiceResource
0..1
tAgent
«attribute» mbox : string [0..1]
«attribute» mbox_sha1sum : string [0..1]
«attribute» gender : string [0..1]
«attribute» URI : anyURI
tService
holdsAccount
0..*
member
0..*
tOnlineAccount
«attribute» accountName : string [0..1]
«attribute» accountServiceHomepage : string [0..1]
tPerson
knows : anyURI [0..*]
«attribute» firstName : string [0..1]
«attribute» family_name : string [0..1]
tGroup
tOrganization

Figure 4.1: Ensemble Entity model UML class diagram

4.1.2 Activity Model

Ensembles combine services and humans to fulfill a specific purpose. Humans plan and execute their actions depending on their current context. In doing so, they create and maintain an implicit, mental process—flexibly linking individual work steps. Loosely coupled services, in contrast, lack overall awareness of the ensemble context. They remain ignorant of past service invocations, interdependent work steps, and relevant human actions. As humans—rather than predefined processes—drive the interaction with other humans and services, we need to provide a decentralized, decoupled means for tunneling context between services.

The concept of activities goes beyond simply reflecting a mental work process. Context tunneling based on activities facilitates coordination in ensembles where humans lack direct communication. We present the underlying activity model here, giving details on the tunneling process in Section 4.2.

An activity (Figure 4.2) specifies coordination properties as well as relationship of humans, services, artifacts, and resources. The activity context contains the overall structure of activities, dependencies between activities, the temporal flow of (future) activities, and history of activity changes. The central elements are the following: (Focusing on coordinative aspects here, we define details of humans, services, and resources in the remaining models.)

Activity describes everything an agent has done, is doing, or will be doing in order to fulfil a goal. An Activity has a name and description, is identified by a URI, and provides details inspired by iCal RFC 2445[1]. We predominately structure activities in

[1]`http://www.ietf.org/rfc/rfc2445.txt`

a hierarchical fashion. Subactivities describe refinements of their parent activity. The relatesTo link enables arbitrary graph structures for managing views which deviate from the main hierarchical structure.

InvolvementRole specifies the engagement of an agent in an activity. Basic role types are Creator, Observer, Contributor, Responsible, and Supervisor. We can assign multiple agents to a single activity, and a single agent to multiple activities.

Artifact is the subject of work being created or modified in an activity. An Artifact wraps a Resource to highlight its special role in an activity. The same Resource can serve as input in one activity and be manipulated as an Artifact in another activity.

Resource is any preexisting form of capability enabling an activity's execution. The activity models only links to a resource. The actual resource is defined in the separate resource model (see below).

Location specifies the place for carrying out the work.

Requirement informs about required and optional skills and roles the agents need to provide in order to successfully carry out the activity.

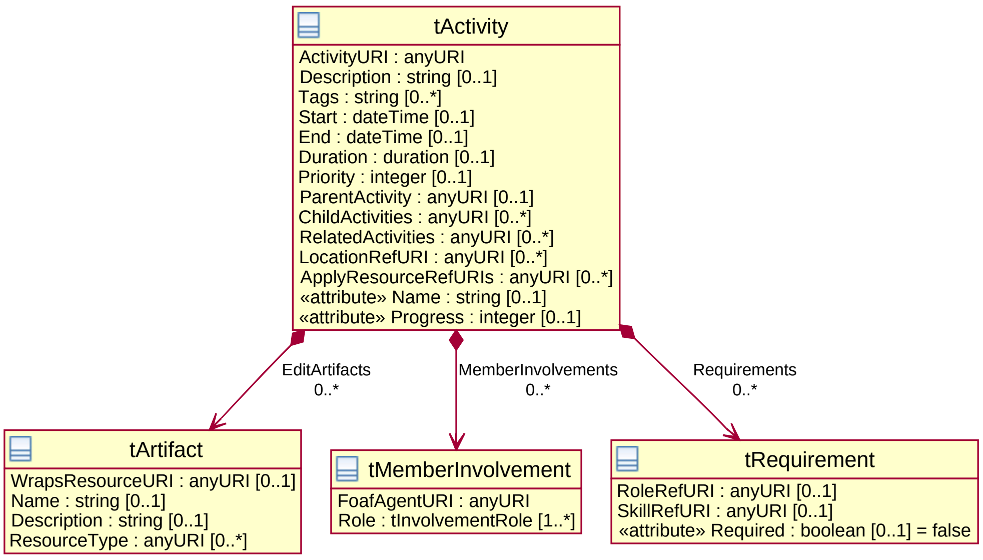

tActivity
ActivityURI : anyURI
Description : string [0..1]
Tags : string [0..*]
Start : dateTime [0..1]
End : dateTime [0..1]
Duration : duration [0..1]
Priority : integer [0..1]
ParentActivity : anyURI [0..1]
ChildActivities : anyURI [0..*]
RelatedActivities : anyURI [0..*]
LocationRefURI : anyURI [0..*]
ApplyResourceRefURIs : anyURI [0..*]
«attribute» Name : string [0..1]
«attribute» Progress : integer [0..1]
EditArtifacts
0..*
MemberInvolvements
0..*
Requirements
0..*
tArtifact
WrapsResourceURI : anyURI [0..1]
Name : string [0..1]
Description : string [0..1]
ResourceType : anyURI [0..*]
tMemberInvolvement
FoafAgentURI : anyURI
Role : tInvolvementRole [1..*]
tRequirement
RoleRefURI : anyURI [0..1]
SkillRefURI : anyURI [0..1]
«attribute» Required : boolean [0..1] = false

Figure 4.2: Activity model UML class diagram

4.1.3 Resource Model

In service ensembles, a resources is anything services and human apply or consume for completing an activity. Some resources types create artifacts (i.e., resources themselves), others manage artifacts, some manipulate artifacts. Resources represent tangible objects such as personal devices, other are purely virtual and serve as container for other resources.

We broadly distinguish between spatial resources that represent a real world, physically locatable entity, and virtual resources. Services connect these two types. Specifically we model:

Resource must be identified by a URI. Human readable name, description, and tags support additional, optional details. A reference to a WS Resource Catalog entry provides details on the different means to access the resource. Sub classes include *Spatial Resource* and *Virtual resource*. The location of a spatial resource exhibits a textual, semantic description in addition to linking to the actual location details. Virtual resources are provided by services.

Service is a virtual resource. It exhibits URLs to service endpoint(s) and WSDL document(s). A service is deployed on one or multiple hosts. We distinguish between a service in the Entity model as an active entity in the ensemble and a service in the resource model as a passive element. Services in the entity model exhibit a reference to their resource counterpart.

Document Resource is any form of message as experienced by a human. Multiple MIME types describe the distinct ways of interpreting the underlying virtual resource.

Host is a subclass of spatial resource and has at least one IP address. It can maintain multiple domains. A *Mobile Device* is a subclass of host and represents personal devices such as smartphones.

CommunicationChannel describes supported communication means to contact an agent. In distributed settings, humans need to interact via communication channels. Services can directly interact. A communication channel, however, does not represent the actual online account, neither a person's availability. The *CommProtocol* specifies further details on how to connect to the communication channel.

4.1.4 Action Model

Understanding the true, fine-grained interaction flows in service ensembles requires detailed data. The managed information—defined in the models above—cannot accurately reflect a service ensemble. With time, the documented configuration will no longer match the actual situation. Additionally, activities are to coarse-grained to capture dynamic properties such as workload on humans and services. Thus, we need to describe the actual work being carried out. Along these lines, we introduce Actions (Figure 4.4) to describe the atomic events caused by interacting ensemble entities. Coordination types and communication types are inspired by the activity primitives introduced in (Dustdar 2004).

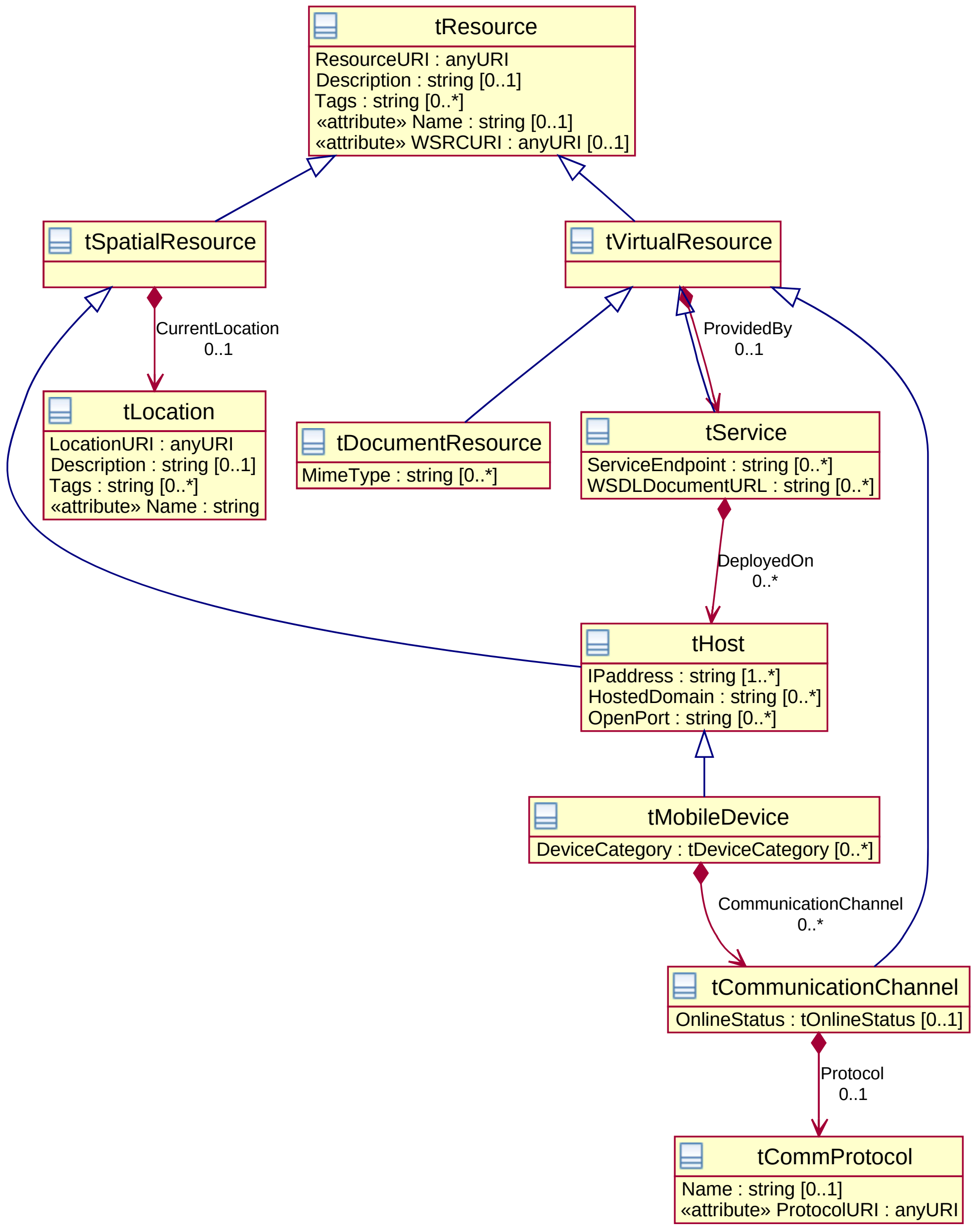

tResource
ResourceURI : anyURI
Description : string [0..1]
Tags : string [0..*]
«attribute» Name : string [0..1]
«attribute» WSRCURI : anyURI [0..1]
tSpatialResource
tVirtualResource
CurrentLocation
0..1
ProvidedBy
0..1
tLocation
LocationURI : anyURI
Description : string [0..1]
Tags : string [0..*]
«attribute» Name : string
tDocumentResource
MimeType : string [0..*]
tService
ServiceEndpoint : string [0..*]
WSDLDocumentURL : string [0..*]
DeployedOn
0..*
tHost
IPaddress : string [1..*]
HostedDomain : string [0..*]
OpenPort : string [0..*]
tMobileDevice
DeviceCategory : tDeviceCategory [0..*]
CommunicationChannel
0..*
tCommunicationChannel
OnlineStatus : tOnlineStatus [0..1]
Protocol
0..1
tCommProtocol
Name : string [0..1]
«attribute» ProtocolURI : anyURI

Figure 4.3: Resource model UML class diagram

Action refers to exactly one Activity and lists the involved Agents, Artifacts, and Resources participating in that Action. Specifically, we distinguish between the immediate source (i.e., InvokedByServiceClient) and the actual originator (i.e., ExecutedOnBehalfOfFoafAgent). Applying the timestamps for temporal ordering, we can establish invocation traces across humans and services. Three sub classes specify further details on coordination, communication, and execution.

CoordinationAction describes activity changes, delegations, and work notifications.

Changes include new, updated, and deleted activities. Delegation types consist of Regular (single responsibility), Joint (combined responsibility), and Split (multiple activity copies, each exhibiting a single responsibility). Delegation replies inform about accepted or denied delegation requests. Work notifications announce an entity starting or stopping work on an activity.

CommunicationAction identifies communicating entities, the applied service, and optionally the type (RequestTodo, RequestConfirmation, RequestDiscussion, RequestComment, RequestInformation, or Unknown). The involved communication service is responsible for capturing interaction duration and intensity.

ExecutionAction specifies the exact invoked service operation.

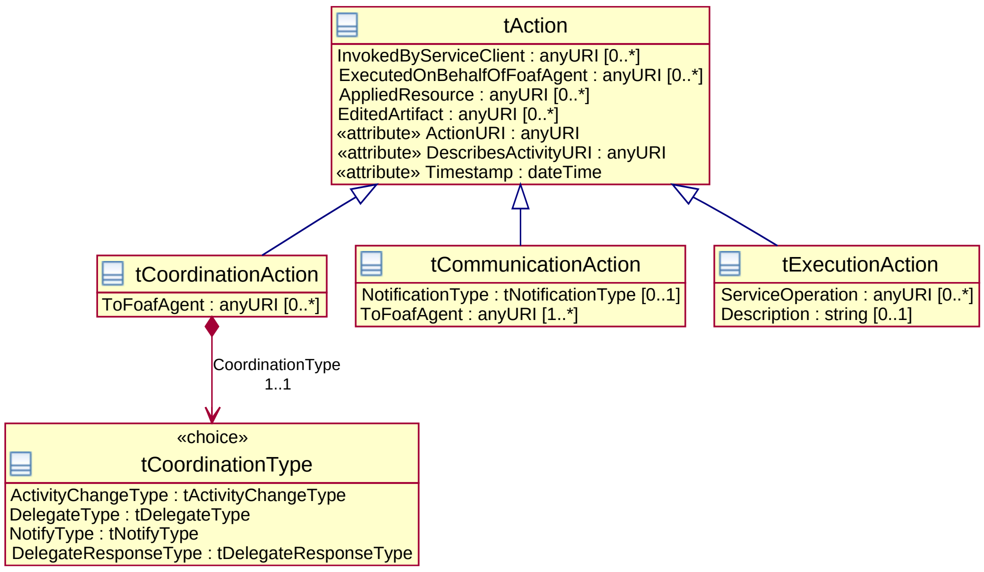

tAction
InvokedByServiceClient : anyURI [0..*]
ExecutedOnBehalfOfFoafAgent : anyURI [0..*]
AppliedResource : anyURI [0..*]
EditedArtifact : anyURI [0..*]
«attribute» ActionURI : anyURI
«attribute» DescribesActivityURI : anyURI
«attribute» Timestamp : dateTime
tCoordinationAction
ToFoafAgent : anyURI [0..*]
tCommunicationAction
NotificationType : tNotificationType [0..1]
ToFoafAgent : anyURI [1..*]
tExecutionAction
ServiceOperation : anyURI [0..*]
Description : string [0..1]
CoordinationType
1..1
«choice»
tCoordinationType
ActivityChangeType : tActivityChangeType
DelegateType : tDelegateType
NotifyType : tNotifyType
DelegateResponseType : tDelegateResponseType

Figure 4.4: Action model UML class diagram

We introduce these three action types to distinguish between different concerns of active and passive entities. Coordination action focus on who is (supposed to) carry out the work, changes of responsibility, and tracking of work execution. Communication actions enable synchronization of human entities. This is either unidirectional (a service or human notifying another human about a certain fact) or bidirectional (information exchange between humans). Eventually communication actions result in coordination actions or execution actions. Execution actions capture the particular services used for advancing the activity progress. Here, we focus on what happens; which service/resource enables creation and modification of which artifact. Thus, separation into action types enables distinguishing between different activity types. Domain specific action intensities are another reason to distinguish between the three types. In one application we might experience frequent coordination actions, in another application we might encounter extensive execution actions. Hence, we are able to factor in such differences during action analysis.

4.2 Context Capturing

The MAPE-K cycle for autonomic computing highlights the fundamental importance of monitoring. Without continuous feedback, we cannot adapt to changing environmental conditions. When we cannot directly measure these dynamic constraints, monitoring amounts to observing the actions of active entities.

In ensembles, service interaction mining becomes the main source of context information. The inContext platform (PCSA) (Reiff-Marganiec, Truong, Casella, Dorn, Dustdar, and Moretzki 2008) provides services for managing resources (especially services and files), tracking activities, providing human communication, establishing organizational structures, and storing context. Humans and services need to manage the semi-static information on activities, organization, and resources themselves. Actions, however, require automatic sensing.

The PSCA exhibits a logging interface that provides detailed information on all service interactions. Specialized sensors further inspect request and response SOAP messages to generate the appropriate action subclass. Activity services yield coordination actions, communication services yield communication actions, and all other services result in execution actions. Listing 4.1 displays an example coordination action. Correlation of SOAP messages, entities, and activities relies on additional SOAP header information. We provide more technical details on logging and the context tunneling SOAP header extension in Chapter 7.

Other forms of context sensing provide details on location, face-to-face communication, and non-service based actions. They remain, however, outside the scope of this thesis. Focus of such research lies in the domains of context-aware computing (Schilit, Adams, and Want 1994,Baldauf, Dustdar, and Rosenberg 2007), pervasive computing (Satyanarayanan 2001,Henricksen, Indulska, and Rakotonirainy 2001), and ubiquitous computing (Endres, Butz, and MacWilliams 2005).

4.3 Context Ranking

Monitoring of service ensembles generates a considerable amount of raw context information. Determining the set of relevant information for the situation at hand becomes a fundamental problem in service adaptation. Consider an entity having to continuously choose the most suitable storage service. It needs to known which service properties to consider, and how to evaluate their utility.

In general, ranking of context candidates (e.g., locations, activities, services, resources) consists of four steps. First, we need to identify suitable metrics for measuring the distance between two candidates. Second, we have to determine the impact each of those distance functions should have on the overall ranking result. Third, we need to determine for each criteria the appropriate utility evaluation function for comparing multiple candidates.

```
<?xml version="1.0" encoding="UTF-8"?>
<act:CoordinationAction
 xmlns:act="http://www.in-context.eu/ns/action"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 ActionURI="http://www.in-context.eu/action/CoordAction32547"
 DescribesActivityURI="http://www.in-context.eu/activity/ReportReview#2"
 Timestamp="2008-11-28T12:00:00">
 <act:InvokedByServiceClient>
    http://www.in-context.eu/resource/mobileclient_3
 </act:InvokedByServiceClient>
 <act:ExecutedOnBehalfOfFoafAgent>
    http://www.in-context.eu/user/Bob
 </act:ExecutedOnBehalfOfFoafAgent>
 <act:AppliedResource>
    http://www.in-context.eu/service/CoreActivityService
 </act:AppliedResource>
 <act:EditedArtifact>
    http://www.in-context.eu/files/InternalReport_v2.pdf
 </act:EditedArtifact>
 <act:CoordinationType>
    <act:DelegateType>DelegateJoint</act:DelegateType>
 </act:CoordinationType>
 <act:ToFoafAgent>
    http://www.in-context.eu/user/Alice
 </act:ToFoafAgent>
 <act:ToFoafAgent>
    http://www.in-context.eu/user/Carol
 </act:ToFoafAgent>
</act:CoordinationAction>
```

Listing 4.1: Example CoordinationAction: Bob delegating a joint document review to Alice and Carol.

Finally, we compute utility values for each criteria and aggregate all individual scores according to the criteria weights.

4.3.1 Distance Metrics

There are three basic alternatives for measuring the distance between two elements of the same type. We briefly outline these categories before discussing them in detail.

Natural distance measurement applies explicitly given information. We utilize Euclidean, chess board, or Manhattan distance for comparing locations. Activity and organization hierarchies rely on edge weights to derive distance to parent, child, and sibling elements. Temporal distance calculates the absolute difference of two timestamps.

Context based metrics consider the situation of two elements. For two activities we compare the overlap of involved persons, invoked services, and utilized resources. For two services we analyze the set overlap of activities, persons, artifacts, and resources. Context-based distance measurement works on managed data (i.e., explicitly configured activities and organizational structure) or captured action data.

Interaction based distance evaluation focuses on action volume and entity involvement. We apply only action data for determining the distance. Specifically, we consider the action distribution of individual elements. Services that are utilized in every activity, for example, contribute no information for deriving the similarity of two activities.

Interaction based metrics yield the most expressive distance measurements. A newly establish ensemble, however, lacks the required actions. In the early stages, individual actions extensively distort the interaction based computation; here context based metrics suit better. Subsequently, we apply natural distance functions at the beginning, later switch to context based functions, and finally utilize interaction based metrics.

4.3.1.1 Natural Distance Functions

Distance measurements for location and time are well understood. We will not discuss these here. For Activity, Organization, and Role, we introduce a graph based distance measurement. The Activity model and Entity model provide primitives to establish a tree structure. For Roles, we define a domain-independent 3-level tree. The leaves represent individual entities (services and persons). The second level aggregates entities exhibiting the same roles, with the top most level (i.e., the root node) combining the various roles. In this tree, comparable elements reside only in the leave nodes.

Let us define the distance between two elements in a hierarchy based on the relation e_{ij}:

$$dist(i,j) = \begin{cases} d_c & \text{if} \quad e_{ij} \in \texttt{child}(i,j) \\ d_p & \text{if} \quad e_{ij} \in \texttt{parent}(i,j) \\ d_s & \text{if} \quad e_{ij} \in \texttt{sibling}(i,j) \end{cases} \tag{4.1}$$

Hierarchies lack explicit links between siblings. We, therefore, apply a graph transformation and labeling algorithm. We interpret a hierarchy's tree structure as an undirected, unweighted graph $\mathcal{H}$ which we subsequently map onto a weighted, directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. We iterate through all child elements starting a the root vertex of the hierarchy. For each vertex, we add an edge from the vertex to its parent with distance d_p, and an inverse edge carrying distance d_c. We avoid creating links between all nodes of a set of siblings by introducing an anonymous node *anon*. Edges from every sibling to this node carry distance d_s. The inverse edges from *anon* to each sibling exhibit distance 0. Figure 4.5 visualizes an example transformation and labeling for an example activity graph with distance $d_p = 2$, $d_c = 1$, and $d_s = 2$.

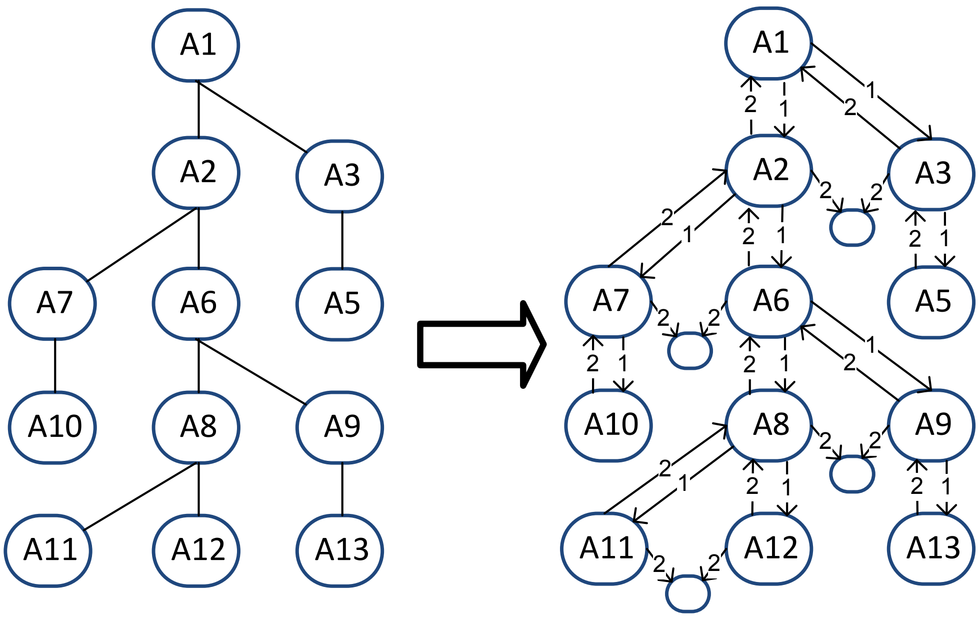


Figure 4.5: Hierarchy transformation and labeling process for distance $d_p = 2$, $d_c = 1$, and $d_s = 2$. Edges beginning at anonymous nodes with edge label 0 are omitted.

The distance between two elements which are not directly connected, is the aggregated distance when traversing the graph on the shortest path between the two elements. We compute the shortest path using Dijkstra's algorithm (see for example Cormen, Leiserson, Rivest, and Stein (2001)).

4.3.1.2 Context-based Distance Functions

We define two types of context-based distance functions distinguished by the underlying data structure. The first type applies to activity structures yet lacking a corresponding set of actions. We focus on involved persons and services, applicable resources, and manipulated artifacts as defined in the activity model. Two activities are considered identical when they exhibit the same set of referenced persons, services, resources, and artifacts.

Algorithm 1 Transformation and Labeling algorithm $\Lambda(\mathcal{H}, d_c, d_p, d_s)$.

```
function BuildGraph($\mathcal{H}, d_c, d_p, d_s$)
    /* Initialize set of vertices and edges. */
    $V \leftarrow \emptyset$
    $E \leftarrow \emptyset$
    $r \leftarrow rootVertex(T)$
    $V \leftarrow r$
    for all $v \in$ child($r$) do
        call $AddNode(v, \mathcal{H}, V, E, d_c, d_p, d_s$
    end for
    call $ConnectSiblings($child$(r), V, E, d_s$
    $\mathcal{G} \leftarrow createDirectedGraph(V, E)$
     return $\mathcal{G}$
end function

function AddNode($n, \mathcal{H}, V, E, d_c, d_p, d_s$)
    $V \leftarrow n$
    /* Add child to parent link. */
    $E \leftarrow edge(n,$ parent$(\mathcal{H}, n), d_p)$
    /* Add parent to child link. */
    $E \leftarrow edge($parent$(\mathcal{H}, n), n, d_c)$
    for all $v \in$ child($n$) do
        call $AddNode(v, \mathcal{H}, V, E, d_c, d_p, d_s$
    end for
    call $ConnectSiblings($child$(n), V, E, d_s$
end function

function ConnectSiblings($N, V, E, d_s$)
    /* Only if there are siblings. */
    if $|N| > 1$ then
        /* Create anonymous sibling connector vertex. */
        $anon \leftarrow vertex()$
        $V \leftarrow anon$
        for all $n \in N$ do
            $E \leftarrow edge(n, anon, d_c)$
            $E \leftarrow edge(anon, n, 0)$
        end for
    end if
end function
```

	Action 4-tuple set $\mathcal{T}$	Context 2-tuple set
p_1		
	$< a_1, \{p_1, p_2\}, \{r_3\}, \{\} >$	$< \{a_1, \cdot\}, \{a_1, r_3\}, \{\cdot, r_3\} >$
	$< a_1, \{p_1\}, \{r_2, r_3\}, \{o_4\} >$	$< \{a_1, \cdot\}, \{a_1, r_2\}, \{a_1, r_3\}, \{a_1, o_4\}, \{\cdot, r_2\}, \{\cdot, r_3\}, \{\cdot, o_4\},$ $\{r_2, o_4\}, \{r_3, o_4\} >$
	$< a_2, \{p_1\}, \{\}, \{o_4\} >$	$< \{a_2, \cdot\}, \{a_2, o_4\}, \{\cdot, o_4\} >$
	Joined	$< \{a_1, \cdot\}, \{a_1, r_3\}, \{\cdot, r_3\}, \{a_1, r_2\}, \{a_1, o_4\}, \{\cdot, r_2\}, \{\cdot, o_4\},$ $\{r_2, o_4\}, \{r_3, o_4\}, \{a_2, \cdot\}, \{a_2, o_4\} >$
p_2		
	$< a_1, \{p_1, p_2\}, \{r_3\}, \{\} >$	$< \{a_1, \cdot\}, \{a_1, r_3\}, \{\cdot, r_3\} >$
	$< a_3, \{p_2, p_3\}, \{r_2\}, \{\} >$	$< \{a_3, \cdot\}, \{a_3, p_3\}, \{a_3, r_2\}, \{\cdot, r_2\}, \{p_3, r_2\} >$
	Joined	$< \{a_1, \cdot\}, \{a_1, r_3\}, \{\cdot, r_3\}, \{a_3, \cdot\}, \{a_3, p_3\}, \{a_3, r_2\}, \{\cdot, r_2\},$ $\{p_3, r_2\} >$
		$\lvert p_1 \cup p_2 \rvert = 15; \lvert p_1 \cap p_2 \rvert = 4$
	Jaccard Distance	$0.7\dot{3}$

Table 4.1: Distance calculation for two action sets of p_1 and p_2 applying Jaccard's distance function.

We limit the set overlap to activity references when comparing resources, service, persons, and artifacts as they lack direct relations in the activity model. The Jaccard distance is a commonly used set overlap function, defined as the difference between set union and set intersection, divided by the set union:

$$J_\delta = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \tag{4.2}$$

where sets A and B contain the references to persons, services, references, and artifacts when comparing two activities a_A and a_B. J_δ becomes 0 when the two sets contain the same elements and yields 1 when the two sets are completely disjoint.

The second type applies action data. We interpret actions as a collection of 4-tuples $\mathcal{T}_{action} < \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{O} >$ comprising activity, persons (including services), resources, and artifacts. For each element v, there exist a corresponding set $\mathcal{T}_{action}(v)$ that contains all actions involving v. Subsequently, we map each action 4-tuple into a set of 2-tuples through permutation of each element in the categories $\mathcal{A}, \mathcal{P}, \mathcal{R}$, and $\mathcal{O}$ with each other element of different category. In both sets, we replace the compared elements i and j with $< \cdot >$. Subsequent joining of the 2-tuple sets ignores multiple instances of identical 2-tuples. The Jaccard formula on the aggregated 2-tuple sets provides the corresponding distance measurement. Table 4.1 gives an example of this process. In the example, the first tuples $\mathcal{T}_{action}$ of the persons p_i, p_j are identical as this was a joint action.

4.3.1.3 Interaction-based Distance Functions

Natural and context-based distance functions ignore the focus of individual elements. There is no difference between the same action tuple occurring once or 100 times. Resources applied in every activity have the same impact as services applied in specific activities only. The goal of inter-action based distance functions is weighting an elements local impact according to its global significance.

To this end, we introduce a 4-partite, labeled action graph $\mathcal{AG}_{\triangle}(\mathcal{V}, \mathcal{E})$. A 4-partite graph (a specialization of a k-partite graph) comprises four vertex categories (K). Each vertex v maps to exactly one category $k \in K$. Edges are undirected and exist only between nodes of different categories. An edge $e_{k,l}$ links vertices of category k and l, with $k \neq l$.

The 4-partite action graph represents the four categories found in an action tuple $\mathcal{T}_{action}$: activity, person, resource, and artifact. Edge labels provide the number of actions containing the particular 2-tuple. The graph in Figure 4.6 is equivalent to the tuples in Table 4.1. In contrast to the context-aware distance metric on actions, edges deriving from joint actions (i.e., involving multiple elements of the same category) are counted once only. We establish four 2-tuples from the action $< a_1, \{p_1, p_2\}, \{\}, \{\} >$ when comparing p_1 and p_2. The 4-partite graph for the same action yields only two edges, each labeled 1. Repeated actions increase the edge label. As p_1 engages in two actions within scope of activity a_1, edge $< a_1, p_2 >$ exhibits label 2.

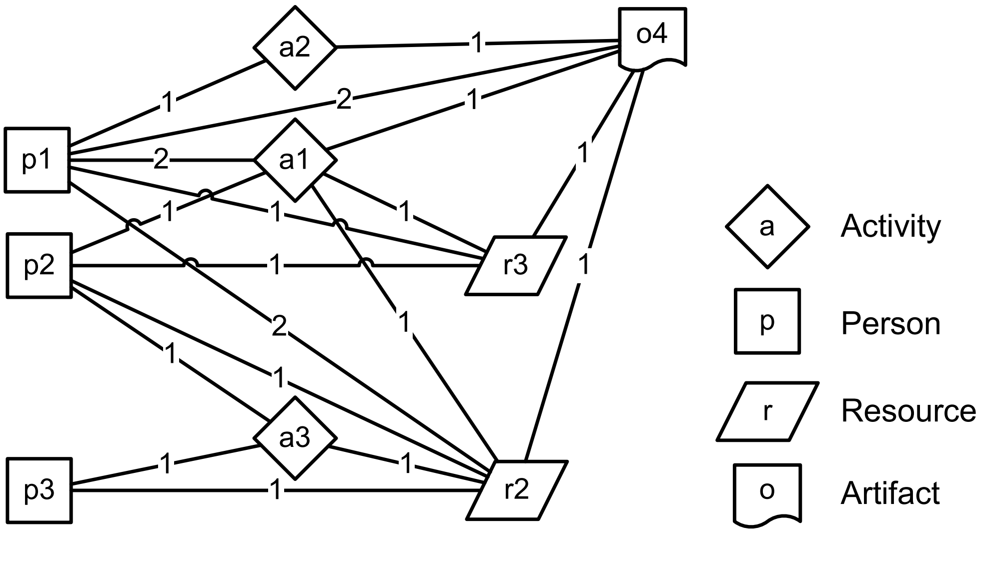


Figure 4.6: 4-partite labeled action graph for the action tuples $\mathcal{T}$ in Table 4.1.

We measure the distance between two elements of the same category by analyzing shared elements within neighboring categories. The distance, for example, between two persons is based on actions involving joint activities, joint resources, and joint artifacts. In general, the tighter two elements of category l are linked to identical elements of category k (with $k \neq l$), the smaller the distance. We define distance on a minimal subgraph

(Figure 4.7) comprising the two elements for comparison $(v1_l, v2_l)$, the connecting third element $(v3_k)$, and two edges $(e1_{k,l}(v1, v3), e2_{k,l}(v2, v3))$ with corresponding edge labels s_1, s_2.

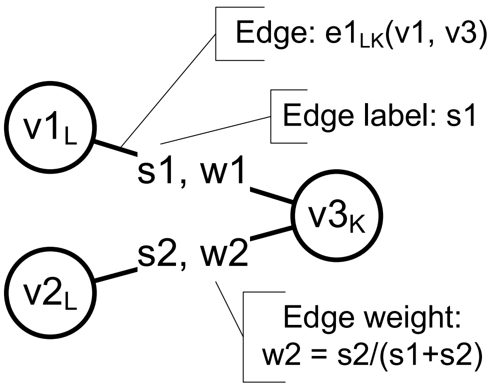


Figure 4.7: Minimal subgraph for calculating distance between elements $v1_l$ and $v2_l$ via element $v3_k$.

We consider the amount of shared elements, the magnitude of involvement, and involvement distribution. Two elements each linked with label 1 to the same three neighbors, yield higher distance than two elements each linked with label 10 to only one shared neighbor. Also, two elements linked with equally distributed labels (e.g., $s_1 = s_2 = 6$) are more similar than elements with unequally distributed labels (e.g., $s_1 = 2, s_2 = 10$). For comparing elements $v1, v2$, we compute the edge weight for each link to neighbor element $v3$. Edge weights w_1, w_2 equal the corresponding, normalized edge labels such that $w_1 = s_1/(s_1 + s_2)$ and $w_2 = s_2/(s_1 + s_2)$. The distinct difference between edge labels and edge weights is the application scope. Edge weights exist only in the corresponding minimal subgraph. They describe the action distribution from element $v3$ towards elements $v1, v2$. For any other minimal subgraph containing one of the edges $e1, e2$, we need to recalculate the weights w_1 and w_2.

We apply Shannon's entropy definition (Shannon 1948) $H(w) = -\sum(w * log(w))$ on edge weights w with $\sum w = 1$ to describe the local focus of an element. We normalize the entropy $H(w)$ to the interval $[0, 1]$ by dividing by $log(2)$ [2]. The focus of an element will be minimal (i.e., maximal entropy $H(w)/log(2) = 1$) if actions are equally distributed. The focus will be maximal (i.e., $H(c) = 0$) when one of the edge labels is 0. The local distance between two elements $v1, v2$ via neighbor $v3$ is defined as:

$$dist_{v3,k}(v1, v2) = \frac{s_1 + s_2}{2} * \frac{H(w)}{log(2)} \tag{4.3}$$

Local edge weights and edge weight distribution are insufficient to sufficiently establish the distance between two elements. We also consider the global significance of a neighbor-

[2]We divide by $log(2)$ as there are two values aggregated by the entropy; i.e., the edge weights w_1 and w_2.

sig	a_1	a_2	a_3	p_1	p_2	p_3	r_2	r_3	o_4
a	n/a	n/a	n/a	0,42	0,37	0	0,37	0	0,37
p	0,42	0	0,37	n/a	n/a	n/a	0	0,42	0
r	0,08	0	0	0,08	0	0	n/a	n/a	0
o	-	-	-	-	-	-	-	-	n/a

Table 4.2: Global context significance for elements in Figure 4.6.

ing element in contributing to the distance measurement. For example, resource r_2 exhibits edges with identical labels to all persons. It should not be considered for comparing any two persons as it does not add any information about the distance between elements in the person category. To this end, the global context significance metric applies Shannon's entropy to describe the information content of an element's edge labels for a specific category. The significance of element v_k for edges $e_{k,l}$ is defined as:

$$sig_k(v_l) = 1 - \frac{-\sum_i (w_{v,i,k} * log(w_{v,i,k}))}{log(|k|)} \qquad \forall \qquad k \neq l, |k| > 1 \tag{4.4}$$

where $w_{v,i,k}$ is the normalized weight of the ith outgoing edge towards category k of element v such that $\sum_i w_{v,i,k} = 1$; $|k|$ denotes the number of elements in category k. The normalization yields a significance value in the interval $[0, 1]$. Significance is 0 when an element exhibits no focus (i.e., high entropy) and equally links to all elements of a particular category. Each element provides $|K| - 1 = 3$ global significance values; one for each neighboring category. Table 4.2 lists the significance values for elements in Figure 4.6. There are no results available for the artifact category as there occurs only one artifact. Resource r_2 is suitable for comparing only activities, while resource r_3 applies only to persons. Person p_3 and activity a_2 yield no information for comparison at all.

The overall distance metric is the inverse sum of all weighted local distance measurements for all neighboring categories:

$$dist(v1_l, v2_l) = \left[\sum_K \left(\sum_{N(k,v1,v2)} sig_k(n) * dist_{n,k}(v1, v2) \right) \right]^{-1} \qquad \forall \qquad k \neq l \tag{4.5}$$

where n is a vertex from the neighborhood set N containing joint neighbors of $v1, v2$ for category k. The distance between two elements becomes infinity when they share no neighbor or only neighbors with significance 0.

To compare non-connected elements (e.g., a_2 and a_3), we calculate multi-hop distance on the intra-category distance measurements. Direct distance measurement within a category yields an undirected, weighted, mono-partite graph. The distance in such a graph for two elements $(v1_l, v2_l)$ h hops apart aggregates the distance measures on the shortest path including a penalty distance for every additional required hop. Suppose element A collaborates closely with element B, and B collaborates closely with C, but A and C have few actions in common. To establish the distance between A and C, we cannot just add the distance AB and BC. Instead we increase the distance BC as defined by a penalty function. The penalty function mimics the transitivity principle in trust propagation (Josang, Ismail, and Boyd 2007,Artz and Gil 2007). For our purpose, a simple exponential penalty function is sufficient. Consequently the penalty increases for growing hops. The penalized distance for two consecutive nodes v_a and v_b on the shortest path between source node ($v1$) and target node ($v2$) is defined as:

$$dist_p(h, v_a, v_b) = dist(v_a, v_b) * 2^{2h-1} \qquad \forall\ h > 0 \tag{4.6}$$

where h is the hop count to v_a from the source node. Thus, the distance for the second hop doubles, and quadruples for the third hop. The distance remains infinity for two elements where each resides in one segment of a partitioned graph (i.e., the there exists no connecting path of any hop count).

For distance between activity a_2 and a_3, we aggregate distance of a_2 to a_1 ($dist = 1.0$) and the penalized distance of a_1 to a_3 ($dist_p(1) = 1.355$). The final distance yields 3.71. For the remaining elements in Figure 4.6, we derive following distance measurements: $p_1, p_2 = 0.792; p_2, p_3 = 2.71; p_1, p_3 = 6.222$ and $r_2, r_3 = 4.08$.

Metrics merely describe how to measure the distance between two values. Any preferences of smaller over larger values (or vice versa) are expressed in the utility functions introduced further below.

4.3.2 Relevance Functions

Relevance function determine the impact individual metrics should exercise upon final aggregation. Each metric i comes with a corresponding weight ω_{r_i}. We assign equal importance to every metric with $\omega_{r_i} = 1/max(i)$. As configuration of weights is very case specific, we cannot give generally valid recommendations for the various metrics. For the remainder of this chapter, we consider all metrics of equal importance.

4.3.3 Utility Functions

Utility functions express the decline of observed fitness as candidate values deviate from the optimum value. In many cases, a continuous function across the full value space is insufficient. The utility function templates in Figure 4.8 display exemplary linear restrictions on value ranges. The horizontal axis shows the relevance metric values and the vertical axis returns the corresponding utility values. Limits a to d list the required configuration parameters. Services need to provide the configuration parameters, which can depend on context themselves. We define following function templates:

HardLowerDecliningOver 4.8 (a), specifies the optimum value as a hard lower limit, with higher values steadily decreasing.

HardUpperDecliningLower 4.8 (b), specifies benefit steadily rising until reaching an optimum hard upper limit.

SoftLowerStableLimitedOver 4.8 (c) extends *HardUpperDecliningLower* with a range of equally optimal metric values.

SoftUpperStableLimitedLower 4.8(d) extends *HardLowerDecliningUpper* with a range of equally optimal metric values.

LimitedSoftLowerLimitedSoftUpper 4.8(e) defines an optimal value with (not necessary equally) decreasing utility on both sides.

LimitedSoftLowerPlateauLimitedSoftUpper 4.8(f) extends *LimitedSoftLowerLimitedSoftUpper* with a range of equally optimal metric values. This function can emulate all previous functions.

Setting a negative impact weight ω_{r_i} reverses the respective utility function. For example, *HardLowerDecliningOver* will treat values outside the limits a and b as rewarding, with utility declining from limit b towards limit a.

4.3.4 Ranking Algorithm

Let us define the set of relevance metric functions $R = \{r_1(ce), r_2(ce), \dots, r_n(ce)\}$ that are associated with each candidate element and ω_r as the weight assigned to metric r such that $\sum_{i=1,\dots,n} |\omega_r| = 1$. Utility functions $U = \{u_1(r_1(ce)), u_2(r_2(ce)), \dots, u_n(r_n(ce))\}$ express the fitness of candidate element ce given the respective relevance metric r. The score for a particular utility function, relevance metric, and candidate element is defined as follows:

$$score(u, r, ce) = \frac{\texttt{max}(u) - u(r(ce))}{\texttt{max}(u) - \texttt{min}(u)} \tag{4.7}$$

where $\texttt{max}(u)$ returns the utility function maximum over all candidate elements; $\texttt{min}(u)$ returns the utility function minimum over all candidate elements; and $u(r(ce))$ provides the utility function result for the candidate element ce by applying the corresponding relevance metric r. We avoid negative metric weights ω because negative weights are either implicitly considered by selecting the appropriate utility function or explicitly considered by inverting the utility function.

The scoring function $score(ce, u, r)$ scales all scores for a particular metric to the interval $[0, 1]$ such that the best candidate yields score 1 and the worst candidate yields score 0.

In the second step, we rank individual context elements according to the overall selected set of metrics and weights. Our approach applies a simplified LSP method (see Dujmovic

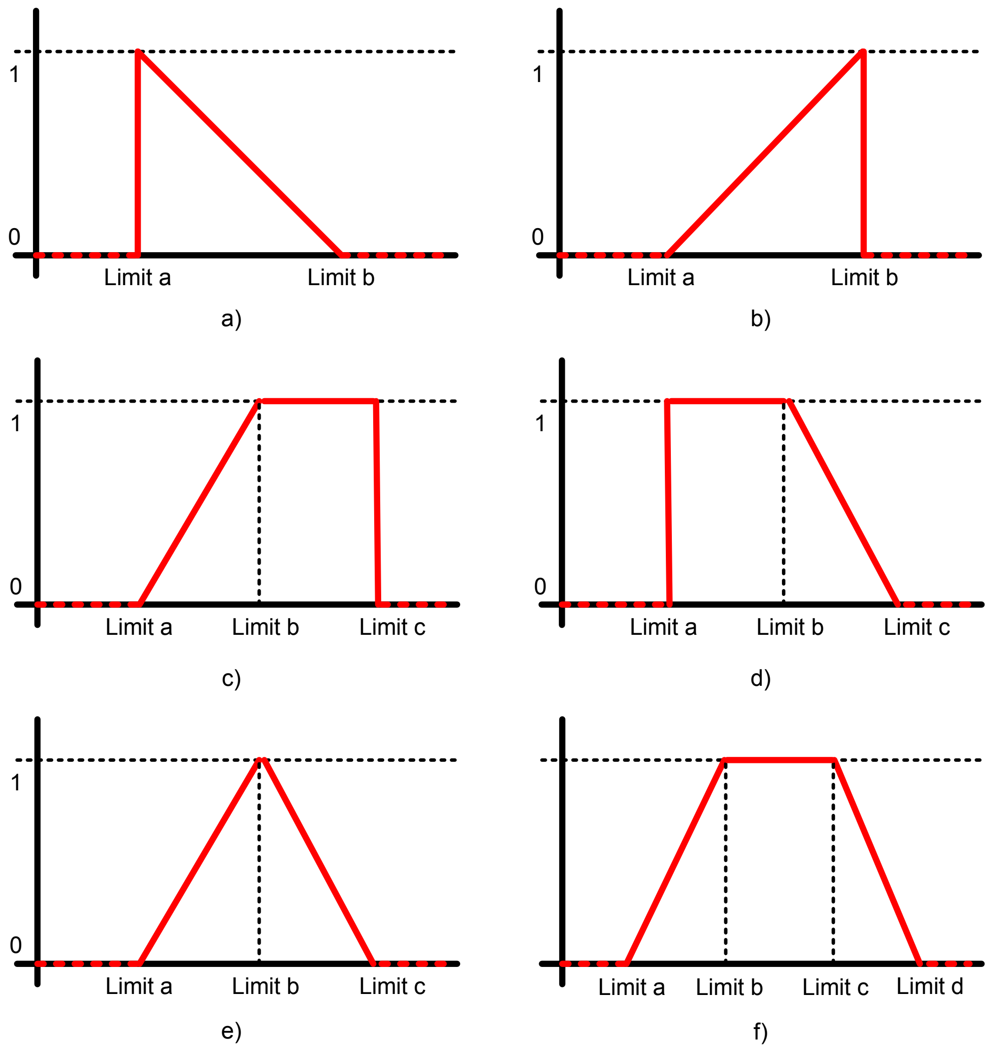


Figure 4.8: Context ranking utility functions

(2007) for a detailed overview). The global score of a candidate element is given as:

$$E(ce) = \sum_{i=1}^{n} \omega_{u_i} * score(ce, u_i, r_i) * 100 \tag{4.8}$$

The separate application of utility functions constitutes the distinct advantage over the simplified LSP method. The LSP applies scoring directly on the candidate elements, thus tightly coupling metric computation and utility evaluation.

4.3.5 Example Application of Context Ranking

In the example in Figure 4.9, we select the natural, hierarchy-based activity distance $w(d)$ and temporal distance $w(t)$ for context elements. We apply the *HardLowerDecliningOver*

Algorithm 2 Ranking Algorithm $\mathcal{RA}(CE, R, \{\omega\})$.

```
  function ComputeGlobalRank($CE, R, \{\omega\}$)
Require: $\sum_{\omega} = 1$
      /* Set or ranked context elements */
      $E \leftarrow \emptyset$
      /* Aggregate individual weights */
      for all $ce \in CE$ do
          $rank_c \leftarrow 0$
          for all $r \in R$ do
              $rank_c \leftarrow rank_c + Util(ce, r) * \omega_r$
          end for
          $E[ce] \leftarrow rank_c$
      end for
      return $sort(E)$
  end function
```

utility function for both metrics. Context elements yield maximum utility when linked to the activity at hand, applicable shortly in the past or future. The parameters for the utility functions are 0 and maximum tree distance, respectively, maximum temporal distance. Thus, for both metrics we prefer lower values over higher values, i.e., closer activities and shorter time difference indicate more relevant context elements. Adjusting the weights determines the importance of one metric over the others. In this scenario, we apply the same ranking weights $w(d) = w(t) = 0.5$ for every activity and context element respectively. For the activity distance utility function, we set the edge weights as: parent $d_p = 2$, child $d_c = 1$, and siblings $d_s = 2$.

We have developed a Context Ranking Web service, which assists in searching for relevant documents. In dynamic ensembles, manual a-priori configuration of relevant services for every activity is infeasible. Instead, the relevance ranking algorithm determines at run-time the most promising services — in the given example *Storage Web Services*.

We trigger the ranking service in scope of activity A8 on the 28th January. The activity graph in Figure 4.9 displays context elements of type Storage Web Service (SWS). Activity distance is given in the upper right corner. Past dates indicate last access of SWS, whereas future dates indicate planned documents available as templates in a SWS. The hierarchy structure and timestamp values are manually generated for this scenario to highlight the effect of the context ranking process.

The context elements used in our ranking example are *Resources* of type SWS. Candidate SWS are those who are referenced within *Action* elements. In addition, we include SWS that are scheduled for managing already existing document templates in future. As both actions and activities contain a timestamp, the ranking algorithm (Alg. 2) is able to rank these context elements. Table 4.3 presents the resulting activity-distance rank $R(d)$, temporal-distance rank $R(t)$, and the aggregated rank $R(M, AG)$.

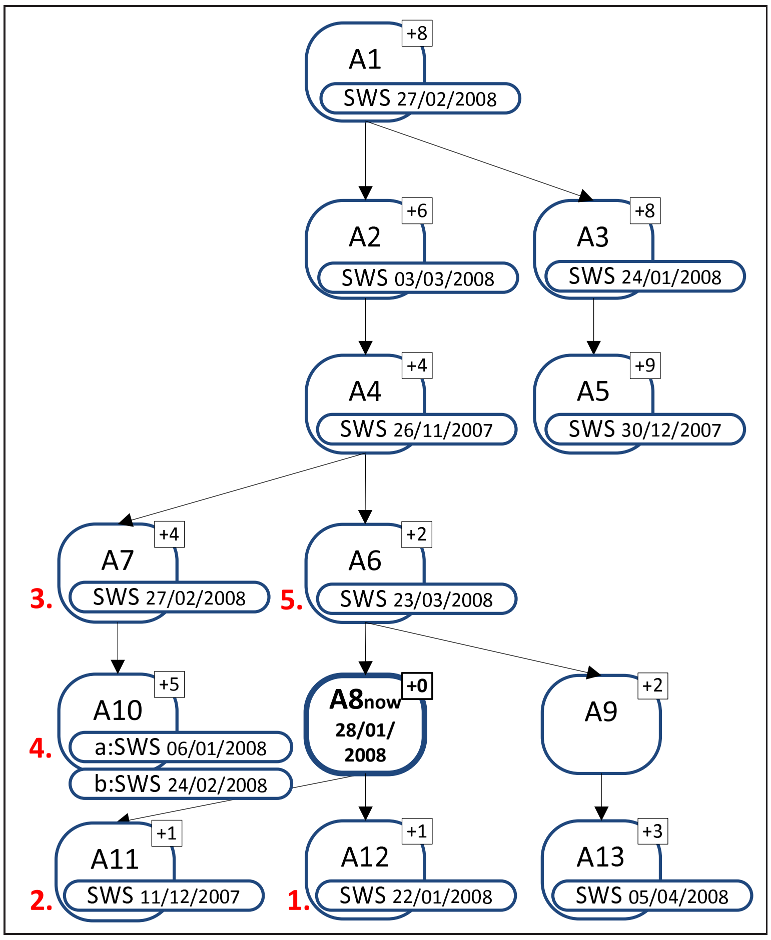


Figure 4.9: Activity Graph excerpt.

Context elements receiving high ranks in both time-distance metric and activity-distance metric will also rank high the overall results (e.g., SWS in A12). A good position in a single ranking, however, does not guarantee final relevance, as this depends on the distribution of the other context elements. For example, the SWS in A11 is ranked 9th place in the time-distance metric, respectively ex aequo 1st in the activity ranking, ending up 2nd place in the overall ranking. In contrast, the SWS in A03 ranked 1st in the time-distance metric, but ranked 10th in the activity-distance metric, winds up in the middle on 7th place. The reason lies in the distribution of metric values. For a number of values (i.e., timestamps) in close proximity (compared to the overall range) their intra metric ranking scores become less significant and the other metrics (i.e., activity distance) become dominant. When two context elements have almost equidistant timestamps from the current activity (A8) we cannot decide which one is more relevant. Thus, the respective activity-distance measurements primarily influence the final rank. In the example, the activity metric dominates over the timestamp metric for following context elements: A10-b, A05, A07, A01 — all within a 0.04 $R(t)$ range.

Act.	d	Date	$R(d)$	$R(t)$	$R(M, AG)$
A12	1	22.01.08	(1) 1.00	(2) 0.96	(1) 98.24
A11	1	11.12.07	(1) 1.00	(9) 0.47	(2) 73.53
A07	4	27.02.08	(5) 0.63	(6) 0.68	(3) 65.37
A10-a	5	06.01.08	(7) 0.50	(3) 0.78	(4) 63.82
A06	2	23.03.08	(3) 0.88	(10) 0.39	(5) 63.16
A10-b	5	24.02.08	(7) 0.50	(4) 0.72	(6) 60.88
A03	8	24.01.08	(10) 0.13	(1) 0.99	(7) 55.66
A02	6	03.03.08	(9) 0.38	(8) 0.62	(8) 49.93
A13	3	05.04.08	(4) 0.75	(12) 0.24	(9) 49.26
A04	4	26.11.07	(5) 0.63	(11) 0.29	(10) 45.96
A01	8	27.02.08	(10) 0.13	(6) 0.68	(11) 40.37
A05	9	30.12.07	(12) 0.00	(5) 0.69	(12) 34.71

Table 4.3: Intermediary and final ranking results: ranking values derive from the structure and elements of the activity in Figure 4.9.

4.4 Evaluation of Context-based and Interaction-based Distance metrics

Context-based and interaction-based distance metrics consider different aspects of an action network. Thus, we first discuss the fundamental difference based on a simple example graph. We then analyze a simulated action network to describe the conditions for which each of the two metrics yield most informative results. Ultimately, we apply the distance metrics to a real world dataset and describe our findings.

4.4.1 Fundamental Differences

When comparing two entities, the context-based metric considers shared and individual action links. Independent of the number of shared links, the distance between two entities increases as the number of individual links grows. In contrast, the interaction-based metric purely analyzes the distribution of actions across the set of shared links.

We limit our analysis to a bipartite graph comprising only persons and activities for sake of clarity. The distance metric principles, however, apply to any k-partite graph. Suppose the action network in Figure 4.10 (a). The example graph consists of three persons involved in six activities. All the links carry equal weight 1 representing an initial ensemble configuration. For this network configuration, Figure 4.10 provide the context-based distance measurements for activities (c) and persons (f). Subfigures (d) and (g) provide the respective interaction-based distance values.

The context-based distance metric yields measurements in the range $[0, 1]$ while the interaction-based metric yields values in the range $[0, \infty]$, Thus, we do not compare absolute distance measurements but focus on the differences within each distance graph (c)-(h).

Observing the context-based activity distance values, we detect the shortest distance $d = 0$ between nodes $a1 <> a2$, $a5 <> a6$, and $a7 <> a8$. These activity pairs feature a complete overlap of adjacent person nodes. The next closest links occur between activities that have one adjacent person in common, with exactly one of the nodes linking to a second person (e.g., $a1$ and $a5$ connect via $p1$, with $a1$ also linking to $p2$.) The distance grows when both nodes exhibit additional non-shared links (e.g., $d(a1, a3) = 0.66$ as besides the shared person $p1$, $a1$ links to $p2$ and $a3$ links to $p3$.)

For the initial graph, the interaction-based distance metric yields less distinguished differences. Distance is shortest between activity pairs $d(a3, a4) = 1.5$ and $d(a1, a2) = 2.2$, with all other links yielding distance $d = 4.4$. The underlying reason is the interaction-based metric's ignorance of non-shared links when comparing two elements. Differences nevertheless arise from the significance values of connecting elements. Table 4.4 lists the global significance and entropy values for the elements of graph (a) and (b). Person $p3$ yields higher global significance than $p1$ and $p2$ as she links to merely two out of eight activities, while $p1$ and $p2$ exhibit involvement in five activities each. Consequently, activity $a3$ and $a4$ are considered closer than $a1$ and $a2$, even the latter ones yield two common person neighbors.

The person distance measurements in Figure 4.10 (f) and (g) provide similar results, e.g., the distance between $p1$ and $p2$ is smaller than both edges connecting $p3$. In more detail, we observe that the interaction-based metric better highlights the difference in involvement (e.g., the distance between $p1$ and $p2$ is half that of $p2$ and $p3$). The context-based metric considers $p2$'s involvement in other activities, thereby reducing the effect of $p1$ and $p2$ having more common activities than $p2$ and $p3$.

	P1	P2	P3	A1	A2	A3	A4	A5	A6	A7	A8
(a)											
Significance	0.226	0.226	0.667	0.369	0.369	0.369	0.369	1	1	1	1
Abs. Entropy	1.609	1.609	0.693	0.693	0.693	0.693	0.693	0	0	0	0
Rel. Entropy	1	1	1	1	1	1	1	0	0	0	0
(b)											
Significance	0.279	0.279	0.667	0.369	0.369	0.369	0.369	1	1	1	1
Abs. Entropy	1.499	1.499	0.693	0.693	0.693	0.693	0.693	0	0	0	0
Rel. Entropy	0.931	0.931	1	1	1	1	1	0	0	0	0

Table 4.4: Significance, absolute entropy, and relative entropy derived for the interaction-based distance metric for graphs in Figure 4.10 (a) and (b).

We add additional actions to highlight further fundamental differences between context-based and interaction-based distance metrics. The graph in Figure 4.10 (b) comprises the same nodes as graph (a) but contains additional actions. Specifically, the three persons tripled their involvement in activities $a1, a2, a3$, and $a4$, i.e., they exhibit a focus on certain activities.

For the evolved graph, we need only recalculate interaction-based distance measurements. The context-based values remain the same as long as we keep the number of nodes and the set of edges the same. We utilize Figure 4.10 graph (c) and (e) to compare activity distance, and (f) and (h) to compare person distance in the evolved graph. Observing activity distance, we notice that the interaction-based metric now provides more details. The distance in-between $a1, a2$ and $a3, a4$ decreases as well as the distance across these four nodes.

With the introduction of additional actions, the global significance values change. Subsequently, nodes that haven't experienced additional actions yield different distance measurements. Activities $a5$ and $a6$, for example, become closer as the global significance of $p1$ increases from 0.226 to 0.279 (see Table 4.4 lower part).

With respect to person distance, we observe the same relative differences as found in the original graph. However, the interaction-based metric allows to distinguish between two set of nodes (e.g., the set $[p1, p2, p3]$ in graph (a) and (b), but not necessarily limited to identical nodes) exhibiting the same distribution of actions, but different involvement magnitude. Thus, we establish that the three persons reside in greater proximity in graph (b) than in graph (a).

To summarize the fundamental differences:

- As the name of the metrics already imply, context-based distance considers predominately the number of non-shared elements, while interaction-based distance focuses entirely on the involvement of common elements.
- Distances rapidly change when elements start to exhibit an unequal distribution of actions. Context-based distance cannot detect elements focusing their actions on a subset of their overall action involvement.
- Context-based distance assigns the same distance values to two sets of elements with identical link structure. In such a configuration, interaction-based distance provides lower distance for the set yielding the greater number of actions.

4.4.2 Simulation-based evaluation

We construct a simulated interaction network to analyze the distance differences for various action distributions. The underlying graph comprises 5000 persons involved in 5000 activities. Similar to real-world complex networks (Albert, Jeong, and Barabási 1999,Albert and Barabasi 2002), the graph exhibits node degrees following a power-law distribution. We generate the link structure by modifying the original algorithm for monopartite graphs by Barabasi and Albert (1999) to produce a bipartite graph. Figure 4.11 displays the degree distribution for activities (a) and persons (b).

We apply the following general procedure to measure the relative differences in distance values in two k-partite graphs $\mathcal{AG}_1$ and $\mathcal{AG}_2$ containing the same nodes but exhibiting different edge weight characteristics.

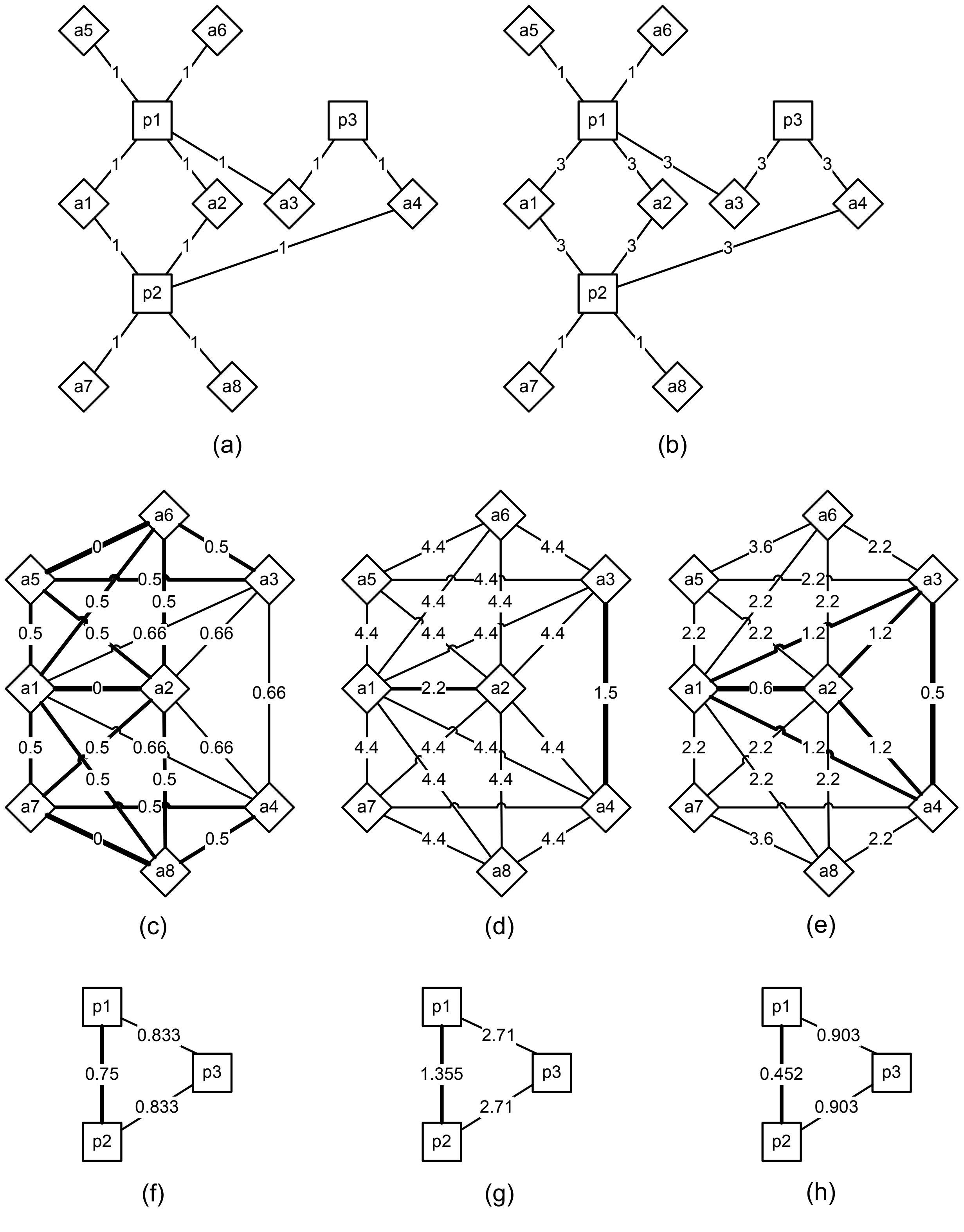
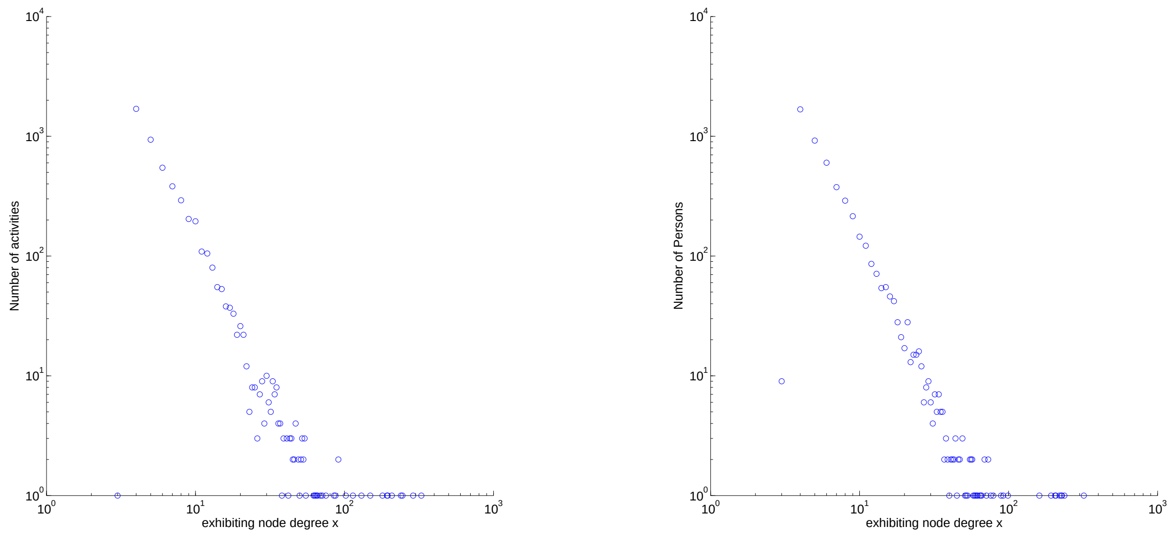


Figure 4.10: Interaction-based and context-based monopartite distance graph for evolving bipartite action graph. Line thickness in subfigures (c) to (h) represents node similarity.

- For the two graphs, we calculate distance measurement for each node type (i.e., Activity, Person ...) creating the respective monopartite distance graphs ($\mathcal{D}_{k1}, \mathcal{D}_{k2} \; \forall \; k = 1 \; \rightarrow \; K$).
- For each node type k, we select a set of candidate nodes from one of the respective distance graphs. For each of these candidates, we select a set of random nodes within the distance graph, the candidate's buddy set.
- For each candidate, we calculate the distance to each buddy and derive the ranking in descending order. For every candidate, we generate the distance rank in each of the two distance graphs (r_{k1}, r_{k2}).
- We apply Pearson's Coefficient for every pair of rankings to measure the difference in element positions. We average over all coefficients for each node type to determine the type specific difference between the two graphs $\mathcal{AG}_1$ and $\mathcal{AG}_2$.

This procedure also applies to graphs with non overlapping node sets when following two conditions hold. First, the set of ranked elements needs to be a node subset in both graphs. Second, the set of ranked elements must be connected in both graphs (i.e., there must exist a path between any two elements of the candidate set in each of the graphs).



(a) (b)

Figure 4.11: Degree distribution for 5000 activities (a) and 5000 persons (b) in a bipartite graph.

4.4.2.1 Pearson's Correlation Coefficient

Pearson's correlation coefficient ρ describes the similarity of two equal-length data sets with $(-1 \leq \rho \leq 1)$. Identical data sets yield 1 and inverse ordered data sets yield -1. 0 indicates no correlation. We apply the coefficient to describe the ranking difference between the two distance metrics on the same graph, and also the ranking difference within one particular distance metric as the underlying graph evolves.

Pearson's correlation coefficient is defined as:

$$\rho_k = \frac{m * (\sum r_{k1}(i), r_{k2}(i)) - (\sum r_{k1}(i))(\sum r_{k2}(i))}{\sqrt{m * (\sum r_{k1}(i)^2) - (\sum r_{k1}(i))^2}\sqrt{m * (\sum r_{k2}(i)^2) - (\sum r_{k2}(i))^2}} \tag{4.9}$$

with m the number of elements (here the buddy set's size), $r_{k1}(i)$ the rank for node i of type k in graph $\mathcal{AG}_1$, and $r_{k2}(i)$ is the rank for node i of type k in graph $\mathcal{AG}_2$.

For our simulated interaction network, we observe the ranking difference for interaction-based distance metric caused by various action distributions. We further compare how these distributions differ from applying the context-based distance metric. Keeping the number of elements and underlying link structure the same, we need not compare context-based distance metrics for the various action distributions. There will be no difference ($\rho = 1$). The three applied action distributions are:

Even - every person engages in exactly one action for any of its neighboring activities.

Linear decreasing - every person p engages in $x * degree(p)$ actions. Each person exhibits a focus on the neighboring activity with the highest involvement of other persons, and linear decreasing focus on the remaining neighboring activities. Thus a particular person engages in $[(x * degree(p) * 0.5) - 1, \ \ldots \ , 1]$ actions with its neighbors such that the average action per neighbor becomes x.

Logarithmic decreasing - similar to linear decreasing, every person engages on average in x actions per neighboring activity. The focus, however, decreases logarithmically. The activity with the most involvement receives most actions. Any subsequent activity receives half the actions of the previous activity until the last neighboring activity receives a single action.

We derive three bipartite graphs $\mathcal{AG}_{ev}$, $\mathcal{AG}_{lin}$, and $\mathcal{AG}_{log}$, that comprise the same activity and person nodes connected via the same set of edges. The graphs merely differ in the edge labels according to the three action distributions. In each of these graphs we derive distance rankings for the same 40 random candidate elements, each exhibiting a set of 40 random buddy elements. The upper part of Table 4.5 contains the Pearson's coefficients for rankings derived from interaction-based distance measurements. In the lower part, we compare ranking differences of interaction-based, and context-based distance measurements.

We notice a significant difference within interaction-based distance ranks for the various action distributions. For persons we establish slightly more distinct differences than for activities. There is hardly any correlation between context-based distances and the three interaction-based distance calculations. This fact highlights the importance of selecting the appropriate distance metric. Even in the case of a single action per link, when interaction-based and context-based metric apply exactly the same set of information, there resulting distance ranks yield hardly any correlation. This supports our suggestion to apply the context-based distance metric in situations when no detailed action data is available, for example, at the beginning of an ensemble.

Interaction vs Interaction	even-linear	linear-logarithmic	logarithmic-even
Activity	0.51 (σ 0.20)	0.47 (σ 0.17)	0.35 (σ 0.17)
Person	0.40 (σ 0.24)	0.34 (σ 0.16)	0.26 (σ 0.18)
Context vs Interaction	even	linear	logarithmic
Activity	0.13 (σ 0.19)	0.06 (σ 0.20)	0.00 (σ 0.20)
Person	0.07 (σ 0.19)	0.12 (σ 0.18)	0.27 (σ 0.18)

Table 4.5: Pearson's coefficient (and standard deviation σ) for node rank differences derived from interaction-based and context-based distance metrics.

4.4.3 Distance metrics applied to real-world data

We analyze the interaction characteristics of slashdot[3] discussion threads. Specifically, interaction-based and context-based distance metrics provide different ranking results when comparing person and activity entities. Ultimately, our analysis outlines the distance changes in dynamically growing ensembles.

First, we give a short introduction to Slashdot and motivate our choice for selecting this data set as a representation of an ensemble. Subsequently, we present our mechanisms to map the Slashdot data into our ensemble context model before providing the core analysis part.

4.4.3.1 Introduction to Slashdot

Slashdot is a user driven news portal focusing on various aspects of information technology. News fall into multiple categories (i.e., subdomains). For our purposes, we concentrate on the subdomain *linux*. Users submit news pieces which editors decide to publish or not. A published piece of news becomes a *story* which all users—anonymous or logged in—can comment on. These comments create a posting hierarchy.

Slashdot's moderation system is a distinguished feature. Each *posting* receives a score between -1 and 5, where 5 denotes an outstanding contribution. Postings by anonymous

[3]`http://slashdot.org/`

users are automatically scored 0. Postings by authenticated users are scored 1 by default. The editors and a changing set of selected users possess a limited number of moderation points to raise or lower these initial posting scores. In addition, they can tag a posting with a predicate such as *Interesting*, *Insightful*, *Informative*, *Funny*, etc. Users that receive higher scores in their postings are more likely to become moderators than users with lower scores. Predicates will not be used here but in the evaluation of Service Infrastructure Adaptation mechanisms in Chapter 6. We include them here for sake of completeness.

Slashdot exhibits similar characteristics as large-scale service ensembles. Some entities remain consistently active throughout all subdomains. Other entities join in an ad-hoc manner, participate for a limited period, and then vanish again. Postings in Slashdot resemble user actions in activities. Users are interested in providing their knowledge to improve the quality and information content of a story (i.e., they fulfil a task.) They rarely engaging in direct communication with other users (Gómez, Kaltenbrunner, and López 2008, Skopik, Truong, and Dustdar 2009). We thus map user postings to actions, and users to person elements in our context model. Extraction of an appropriate activity structure requires additional steps outlined in the following subsection.

4.4.3.2 Slashdot Posting Aggregation

A Slashdot posting provides details on user, time of posting, a unique id, and a reference to its parent posting. A set of postings from the same story establish an action hierarchy, but no explicit mapping to an activity hierarchy. Treating each action as an individual activity is straight forward, but will provide little information in the ultimate bi-partite graph as every activity will then always link only two persons: the posting's creator and the posting parent's creator.

We devised Algorithm 3 to aggregate a posting hierarchy. In the course of the algorithm multiple actions are assigned to the scope of an activity. The story always becomes the root activity (function *Init*). We also define an energy threshold e that defines when a posting should trigger the creation of a new activity. The posting then becomes an action both in the scope of the new activity and the parent activity.

Each posting exhibits energy based on its child postings and grand child posting (function *CalculateEnergy*). The basic energy level corresponds directly to the number of child postings. This value is increased dependent on the distribution of grandchildren. To this end, we count the number of grandchildren each child exhibits and derive the corresponding entropy value. This entropy is then normalized to the interval $[0, 1]$. Posting structures that feature equally distributed grandchildren yield maximum entropy and thus double the basic energy level. Posting structures that exhibit no grand children, or only one child with grandchildren yield minimum entropy and leave the basic energy level unchanged. Examples for the former configuration include three children with each two grandchildren, while an example for the latter configuration comprises three children with only one having two children.

The current posting triggers a new activity when the posting specific energy combined with the amount received from the parent posting exceeds the energy threshold. Otherwise, the posting splits its energy equally across all its child postings. We have to propagate energy from parents to children to avoid all postings being assigned to the root activity in case no single posting reaches the energy threshold by itself.

The idea behind this energy-based aggregation mechanism is following. Multiple child postings indicate different views (i.e., different activity aspects) on the posting at hand. When these child postings exhibit themselves a set of (grand)children, we assume this views to be of substantial significance. Hence, when there are sufficiently many children, respectively sufficiently distributed grandchildren, we create a new activity. When a posting exhibits children that create a narrow (and potentially deep) hierarchy tree, we consider these postings belonging together and thus the observed posting remains in the scope of the parent posting's activity.

The energy threshold parameter controls the amount of activities created within a story. Setting $e = 0$ results in every posting becoming an activity. Even this yields more structural information than simply turning postings into activities (as outlined above). Each posting would result in two actions: one belonging to the current activity, and one associated with the parent activity. For the other extreme, $e \to \infty$, all postings within a story create actions belonging to a single activity.

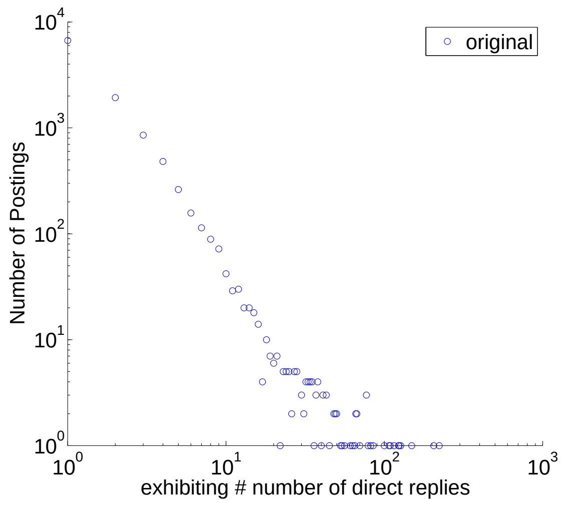
We need to ensure that the aggregation algorithm preserved the posting characteristics before we can continue to create and analyze the bipartite graph comprising persons and activities. The posting-to-activity aggregation is not the only transformation of posting hierarchies. A considerable amount of postings rise from anonymous users. We need to filter these postings without breaking the overall hierarchy. Starting at the root posting, we bridge every posting of an anonymous user, by rewiring the parent reference of all child postings.

The underlying dataset comprises 3477 stories from 19 subdomains in the period of January 1st 2008 until July 1st 2008. In these six months, the linux subdomain exhibits 96 stories. Figure 4.12 (a) prints the number of postings in the linux subdomain against their child count (i.e., direct replies) before any filtering. In subfigure (b), anonymous posts are removed. This step completely preserves the node degree characteristics.

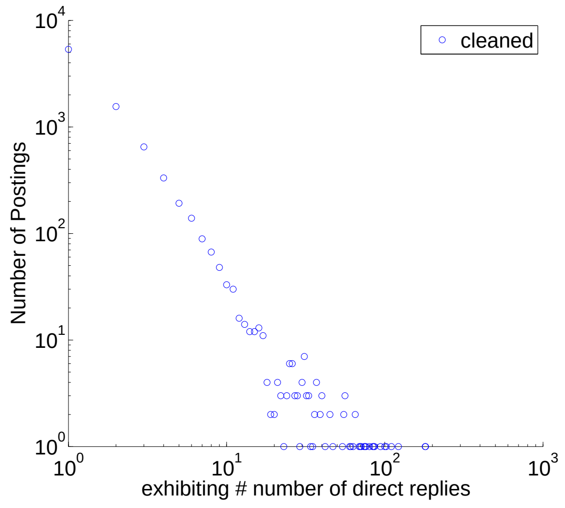
Subfigures (c) and (d) describe the activity hierarchy based on aggregated cleaned postings. We apply an aggregation energy value of $e = 3$ throughout our experiments in this chapter. Observing subfigure (c), we note a shift to the left caused by aggregation. There are significantly less activities than postings. The steepness of data distribution, however, remains the same, thus preserving the degree characteristic. Finally, in subfigure (d), aggregation caused a dampening. The number of activities exhibiting only few actions (i.e., postings) is greatly reduced. The energy threshold eliminates the possibility of activities having no or only a few associated actions. The degree distribution for activities with 10 or more actions yields the same steepness as for the original and cleaned postings. Ultimately, we derive the bipartite graph from the set of actions as outlined in Section 4.3.

Algorithm 3 Aggregate Postings to Activities Algorithm $\mathcal{AGG}(story, e)$.

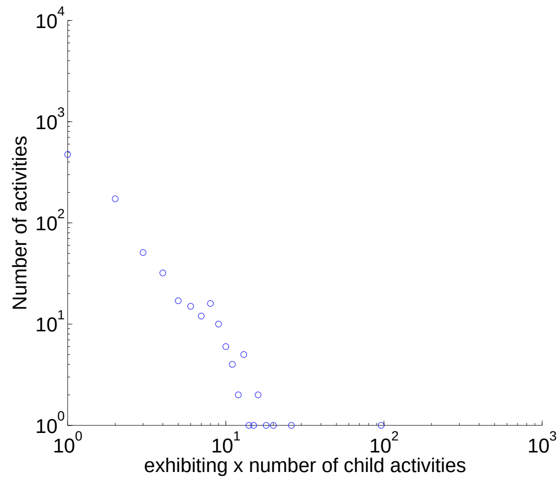
```
function INIT($story, e$)
    /* Create a root activity for the story. */
    $a \leftarrow newActivity(story)$
    /* Add the new activity to the set of activities. */
    $A \leftarrow a$
    for all $childPostings \in story$ do
        call $Aggregate(childPosting, 0, e, A, a)$
    end for
    return $A$
end function

function AGGREGATE($posting, topEnergy, e, A, currentActivity$)
    $currentEnergy \leftarrow CalculateEngery(posting, topEnergy)$
    if $currentEnergy > e$ then
        $aNew \leftarrow newActivity(posting)$
        $A \leftarrow aNew$
        $addAction(currentActivity, posting)$
        $addAction(aNew, posting)$
        $currentActivity \leftarrow aNew$
        $currentEnergy \leftarrow 0$
    else
        $addAction(currentActivity, posting)$
        $currentEngery \leftarrow currentEnergy/childCount(posting)$
    end if
    for $childPostings \in posting$ do
        call $Aggregate(childPosting, currentEnergy, e, A, posting)$
    end for
end function

function CALCULATEENERGY($posting, parentEnergy$)
    $GC \leftarrow$
    for $childPosting \in posting$ do
        $GC \leftarrow childCount(childPosting)$
    end for
    $entropy \leftarrow 0$
    if $|GC| > 1$ then
        $ent \leftarrow calcEntropy(GC)$
        $entropy \leftarrow ent/|GC|$
    end if
    $energy \leftarrow childCount(posting) * (1 + entropy)$
    return $energy \leftarrow energy + parentEnergy$
end function
```



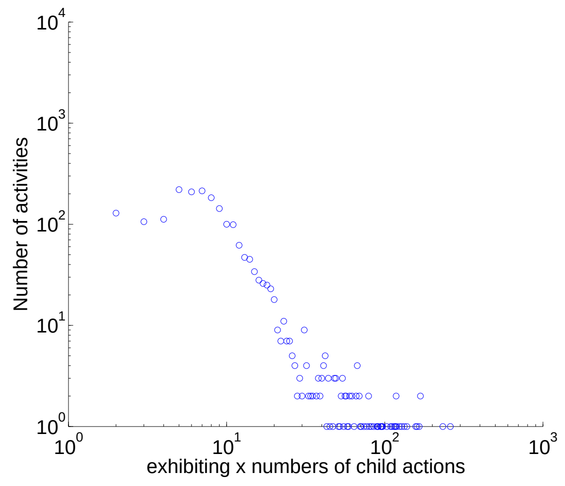
(a)



(b)



(c)



(d)

Figure 4.12: Degree Distribution for complete posting set (a) and cleaned of anonymous postings (b). Degree distribution for child activities from aggregated posting hierarchy (c) and action distribution (d). All postings from stories in the linux subdomain between Jan 1st, 2008 and July 1st, 2008.

4.4.3.3 Analysis of Evolving Ranking Differences

We analyze the rank differences for interaction-based and context-based distance metrics for a growing bipartite graph. Thereafter, we introduce an aging mechanism that removes old actions from the bipartite graph. Our goal is to describe the effect of additional actions entering the graph on the ranking order of activities and persons.

Slashdot data displays different characteristics of how persons and activities emerge for the first time. Subfigures 4.13 (a) and (b) print the amount of unique persons, respectively activities, against the number of actions (21390) in temporal order. For 7172 persons, we note a slightly concave increase in new entities across the whole duration. The 1992 activities increase in a more linear manner. The difference becomes more obvious when we limit the analysis to elements with degree 14 and higher. The emergence of the most active users (subfigure (c)) happens rapidly: 75% of all regular users (267) submit a posting (i.e., action) within the first 20% of postings. For the top connected activities (269) (subfigure (d)), we observe only a slight difference to the complete activity set. Activities, in contrast to persons, remain connected just within a story. Once a story has received its last posting, no more activities are added. Users, on the other hand, are free to submit in any other upcoming story.

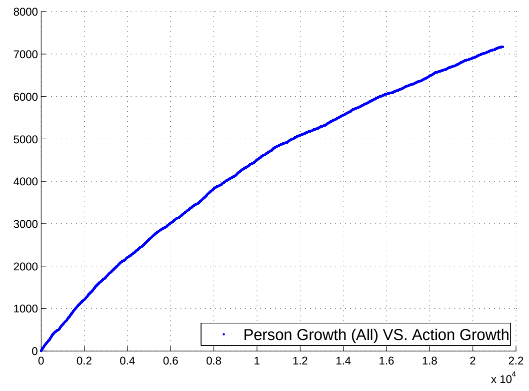
Similar to experiments on our simulated graph, we compare ranking differences of a set of entities for two different action configurations (i.e., original and evolved graph). The experiment runs identically for persons and activities, thus we outline the procedure only for persons.

We select a subset of the overall persons from the bipartite graph within the linux subdomain. This subset comprises of 267 persons with degree equal or higher than 14 and is refered to as the *top persons*.

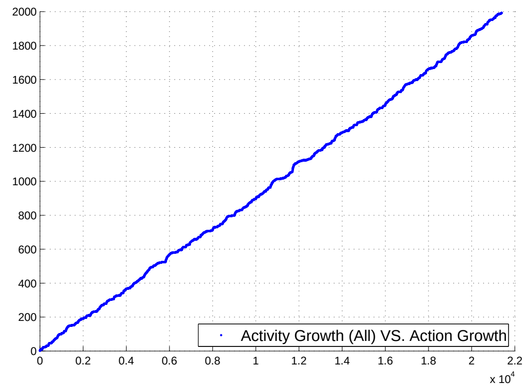
The initial bipartite graph consists of actions from the first 11 stories. The starter top person set (147) denotes the persons that occur both in the top persons set and in the initial graph. Thereof, we select random 20 persons—the candidates—from these set. For each candidate, we assign additional 20 random persons—the buddy set—from the starter top person set. For each candidate, the various distance measurements to its buddies provides the first ranking.

For the remaining duration, we select batches of 10 stories, add the actions to the bipartite graph and recalculate the distances. For two consecutive distance rankings, we apply again the Jaccard coefficient to provide the changes in distance values caused by the additional actions. We continue to add actions and recalculate ranks for all stories within the linux subdomain.

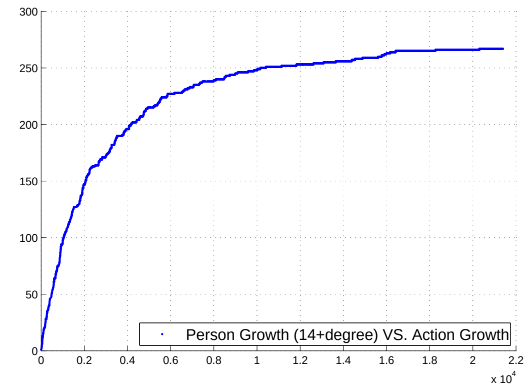
Figure 4.14 (a) displays the average Jaccard coefficient for the 20 candidate persons for interaction-based and context-based distance metrics. To compare the top persons to the average person, we additionally selected 20 random members from the initial graph, assigned 20 random members again from the overall initial graph, and likewise derived distance rankings and their differences. Figure 4.14 (b) visualizes the same procedure for activities.



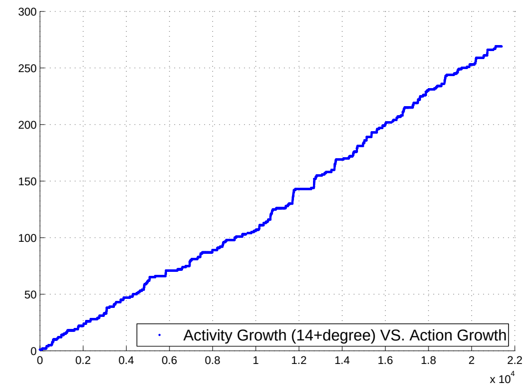
(a)



(b)

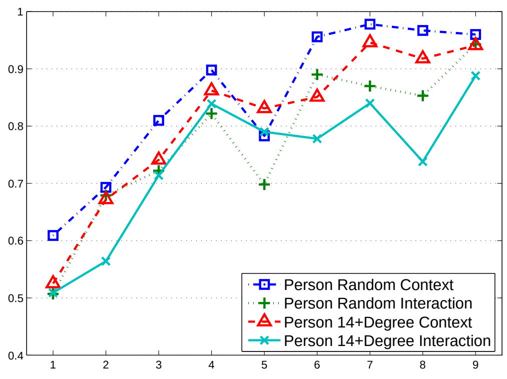


(c)

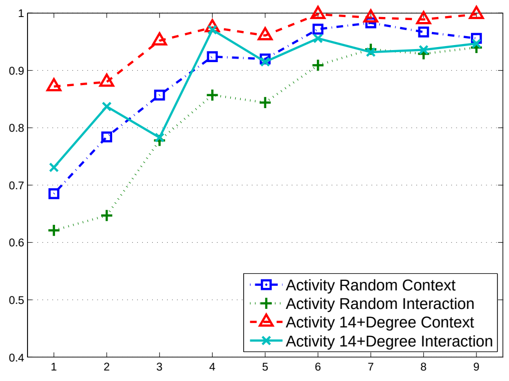


(d)

Figure 4.13: Emergence of unique elements versus growth of actions: (a) all persons, (b) all activities, (c) persons with $degree > 14$ in the overall graph, (d) activities with $degree > 14$ in the overall graph. Cleaned 21390 postings from 96 stories in the linux subdomain between Jan 1st, 2008 and July 1st, 2008.



(a)



(b)

Figure 4.14: Distance ranking differences for every 10 additional stories in the linux subdomain for (a) persons and (b) activities.

In Figure 4.14 (a), we observe the interaction-based rankings providing larger differences than context-based rankings for both top and random candidates. With exception to interval 5, interaction-based metric for the top persons ($\times$) outperforms the same metric ($+$) for random candidates. Similarly, for context-based distance ranking differences, the top candidates ($\triangle$) yield larger differences than the random candidates ($\square$). The top candidates are more likely to engage in additional actions and, in addition, are linked to other persons of high activity level. They are, thus, more prone to distance ranking changes than a random person.

The final interval exhibits a rapid decrease for interaction-based differences (visible as a sharp incline of the corresponding curves). This interval comprises only 5 additional stories compared to the usual 10 and thus introduces fewer additional activities and persons. Interaction-based differences are already close the the difference minimum (1), thus do not decrease further.

For activities, ranking differences between intervals are smaller overall. For the overall duration, the randomly chosen activity set outperforms the top activity candidates for both metrics. The first 11 stories provide 191 activities, of which 22 exhibit overall degree equal or greater than 14. As additional user involvement in an activity does not occur, any source of distance changes are users engaging in new activities. Their actions, thus, have less effect on the distance between the top activities than on random activities exhibiting on average little user involvement. In addition, we observe a Pearson coefficient greater than 0.9 for all activity sets from interval 6 onwards.

We notice an general decrease of ranking differences when comparing results of persons and activities. As the graph accumulates actions throughout the period, additional actions towards the end have less impact on the action distribution and entity connectivity than early actions. Subsequently, we continue our analysis by introducing an aging mechanism.

4.4.3.4 Analysis of Aging Ranking Differences

In the previous experiment, actions remained in the bipartite graph for the complete duration. With the introduction of aging, we remove actions after a certain amount of time. The aging-aware analysis builds upon the basic growing graph experiment outlined in the previous section. The selection of candidates and ranking difference calculation remains the same. The activity building process requires us to view each complete story as a step in time. Thus, we cannot consider the actual time provided with each posting.

The main controlling parameter in this aging-aware experiment is the aging interval. It defines after which period of time an action is removed from the graph again. Reducing the interval too much reduces the analysis to entities that occur in every story. Extending the interval reduces the period where we can observe the effect of aging. We apply an aging interval of 21 based on observations from our experiments. Thus, actions from the first story drop from the graph as the $21st$ story is added.

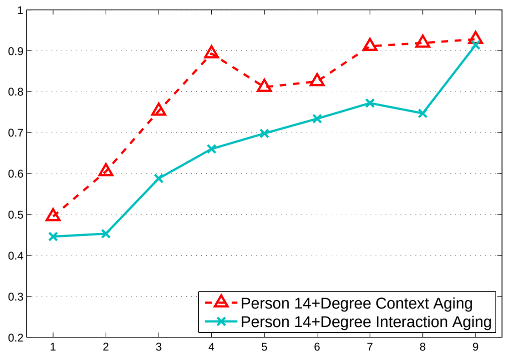
Removing an action does not necessarily imply that two connected entities will loose their common link. Instead, this process reduces the edge weight between these entities. Eventually the weight becomes 0, when no emerging action reenforces the link. We provide two alternatives for handling the removal of actions. The first method reduces the weight of an edge to a minimum of 1. This preserves all links in the graph. Consequently, only the interaction-based distance metric will yield different ranking results. The second method actually removes the edge. The distance between a candidate and one of its buddies becomes infinity when the the last path between them is removed. Multiple disconnected buddies are equally distance from their candidate.

For the context-based metric, there is no difference between keeping a minimal edgeweight and the non-aging procedure. We apply this technique to demonstrate the sensitivity of the interaction-based metric to changes in entity focus. Figure 4.15 (a) compares the two distance measurements for the top person candidates. Interaction-based rankings ($\times$) yield significantly larger differences for the aging-enabled experiment, than for the non-aging experiment in Figure 4.13 (a). The context-based ranking differences remain almost the same.

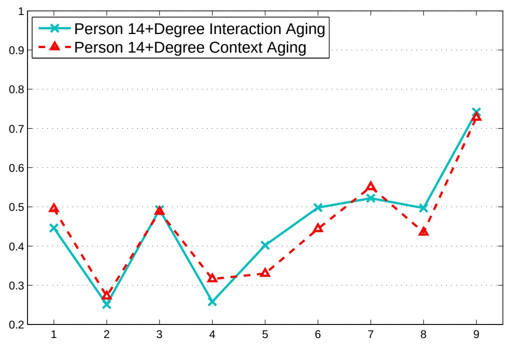
In subfigure (b), context-based and interaction-based distance metrics yield distinct ranking differences as we remove edges once their weights drop to zero. Compared to limited aging in subfigure (a), both metrics remain below or around 0.5 until the last aging iteration. Within slashdot, actions hardly carry weights larger than 3 (i.e., hardly any user posts three times within the same activity). Consequently, reduction of edge weights mostly corresponds to removing the respective edge. This effect causes the context-based metric to perform as good as the interaction-based metric.

We reduced the distance measurement sampling interval in subfigure (d) to 5, while keeping the aging interval at 21. Ranking differences become smaller, as in each interval only half of the stories provide additional actions. The characteristics of the difference curves, however, remain. The minimal ranking similarities at interval 2 and 4 in subfigure (b) correspond to the local minima in subfigure (d) at interval 4 and 8.

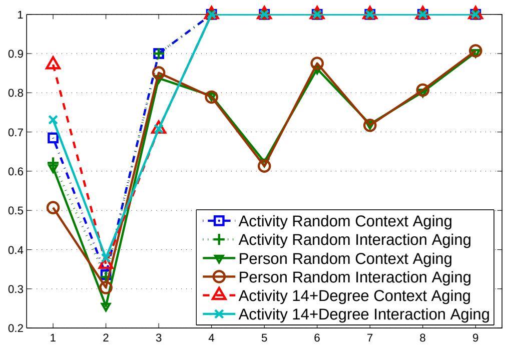
Aging-based analysis of distance differences is only sensible for entities, which we can expect to engage in future actions. This is not the case for randomly selected persons and activities in general. Figure 4.15 (c) demonstrates the steep raise in differences when the aging process comes into effect at interval 2. Immediately thereafter a sharp decline in differences and rapid convergence to 1 indicates the graph-wise separation of candidates and their buddies.



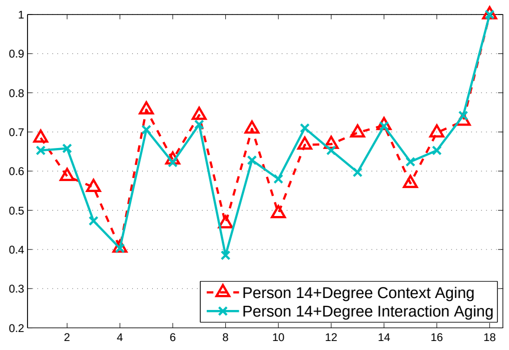
(a)



(b)



(c)



(d)

Figure 4.15: Ranking differences of top persons distances for limited aging (a), normal aging(b), and normal aging(d) with reduced difference sampling interval (5). Distance differences for normal aging for top and random activities, as well as random persons (c).

4.4.3.5 Summary on Distance Metric Differences

Context-based and interaction-based distance metrics consider different aspects of the underlying action data set. Context-based distance focuses on the ratio of joint and individual neighbors of two entities in a k-partite graph. In contrast, interaction-based distance considers only the magnitude of involvement with common neighbors.

We have analyzed the metrics under various conditions to outline their sensitivity towards changes in the action structure. The context-based metric yields the same distance measurements as long as the basic link structure remains the same. The interaction-based metric yields different results for different weight distributions on top of the same link structure.

In the case of k-partite graphs subject to aging, both metrics provide similarly changing ranking differences when the majority of edge weights between two entities remains close to 1.

4.5 Context Provisioning for Mobile Service Ensembles

Mobile ensembles comprise static and mobile entities. Services and humans exhibit context switching when involved in several contexts at the same time. These context changes include work on different joint activities, relocation, shifting workload, and available hardware. Coordination and synchronization between entities becomes ever more important. To this end, we require additional modeling and context distribution effort. In this section we motivate granular context modeling. We provide schemas for context hierarchies and introduce a hybrid push/pull context provisioning mechanism. Eventually, we evaluate the benefit of our granular approach compared to pure push or pure pull based provision techniques.

Suppose following mobile service ensemble. Alice collaborates with Bob, Carol, and Dave on a joint activity. They are employed at different companies working from their office, on the move, and also from home. In such a heterogeneous environment services reside on both mobile devices and static hosts.

At one point, Alice wishes to coordinate critical work with her colleges in a face-to-face fashion. She delegates this task to a *Coordination Web service*. This composite service possesses enough logic to coordinate persons, but requires further services for acquiring calendar data, checking availability, executing a scheduling algorithm, and resolving arising date conflicts. Figure 4.16 visualizes involved entities and steps of the following description.

1. Alice invokes the Coordination Web service stating the corresponding activity.
2. The scheduling service retrieves involved persons and services from the context service, then contacts the shared *Calendar Web service* (2a) to retrieve the calendars of all participating persons (including Alice). It also invokes the *Context Web service* (2b) to check for the users' current reachability. We assume the context service has subscribed to all members respectively their devices for high-level availability and device status context information. Currently, Dave's laptop and PDA as well as Carol's smartphone are online, while Bob is unavailable for the moment.
3. Next, the service queries all available devices for their system load and capabilities (3a) and finally invokes the *Scheduling Algorithm Web service* on Dave's laptop (3b), which is experiencing the least load.

4. In the meantime, the context service notifies (4a) the scheduling service that Bob is available now and Alice has become offline. In addition, Carol changes from her smartphone to her laptop, yet this information is not propagated as also the awareness service has not subscribed at such level of granularity.

5. The *Scheduling Algorithm Web service* detects (5a) a conflict that requires human intervention to be solved. As Alice is still offline, the cooridination service cannot contact all necessary members. Hence, it subscribes (5b) to action information concerning the whole team at a very coarse-grained level, as all members prefer to be contacted when at work and not during their freetime.

6. The context service notifies (6b) the coordination service that all members are online once Alice reports back. Thus using fine-grained reachability information directly from all connected devices a *Communication Web service* on the best suited device for each participant connects all involved persons to agree on the proposed date or another date. As the *Communication Web service* accesses activity information (task-related context information) it chooses the right means of communication: in this case synchronous chat.

7. After the four have agreed on the meeting details, their calendar is updated and the coordination service terminates.

This scenario highlights two ways of retrieving context information. The composite coordination service subscribes and queries context information at different levels of granularity. On the one hand it requires change events (for which it receives notifications) and on the other hand it accesses additional context facts once certain changes have occurred. For providing context in such a dynamic, non-deterministic environment, pure pull or pure push-based mechanisms yield extensive load on bandwidth and capacity constraint devices.

Combined granular structuring of context information with a hybrid sharing mechanism greatly reduces the amount of information transferred between nodes. We benefit from avoiding transferring unrelated context, or information on activities, devices, or persons at a too detailed level.

4.5.1 Hierarchical Context Model

A hierarchy describes context elements as layered pieces of information. A granular representation contains the most generic information at the highest level and the most detailed information at the bottom. Depending on the specific problem domain, such a hierarchy exhibits additional levels at the top and bottom. Each level contains one or more context types. Thus, levels describe the granularity and position within a hierarchy, whereas types describe the information structure.

The hierarchy metamodel distinguishes between hierarchy descriptions and hierarchy instances. For service ensembles, we specify both parts as XML schema documents (see

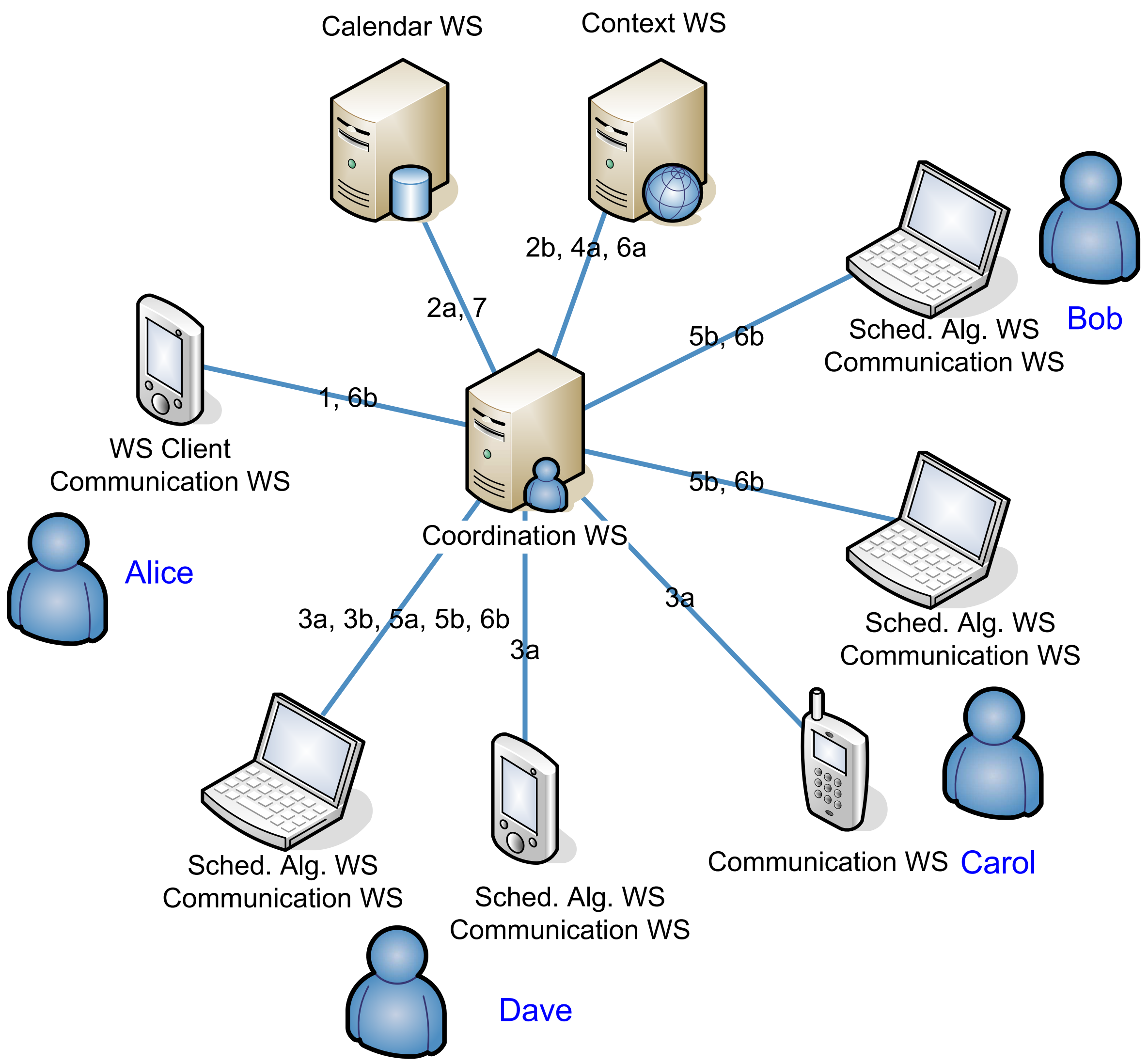


Figure 4.16: Coordination scenario in a mobile ensemble. Service clients and communication services reside on mobile devices. The composite *Coordination Web service*, the *Calendar Web service*, and the *Context Web service* are deployed either distributed or centrally provided by the infrastructure. The numbered lines represent the temporal information flow between nodes according to the textual description.

Figure 4.17 for UML class diagrams). We extend the metamodel to describe specific context types—thereby generating specific hierarchies. The generic hierarchy model comprises the following elements:

HierarchyDef The containment element *HierarchyDef* exhibits identifier and version property to enable adapting and evolving hierarchies. Name and a human readable description provide information on the general purpose. The maximum number of levels determines if the hierarchy can dynamically grow. The Hierarchy definition element refers to all defined levels.

Level Each hierarchy consists of a number of *Level* elements. Each level has an identifier, name and human readable description. Links to the parent level establish the hierarchical structure able to include additional levels later. Simple hierarchies consists of levels containing one *Type* each. Several types on the same level are treated as alternative context representations. This mechanism enables horizontal hierarchy expansion.

Type specifies the representation of a context element at the corresponding level of granularity. A type links to its parent type to express a dependency relationship enforced in a corresponding HierarchyInstance. This dependency relationship restricts use of valid types on the same level. Suppose a hierarchy containing three types T1 ... T3 on level L1 and three types T4 ... L6 on level L2. If T4 defines a parent type link to T3, any HierarchyInstance containing content of type T4 on level L2 must have content of type T3 on L1. Usually the number of branches and thus the complexity of the type tree will remain small.

HierarchyInstance contains the granular structure of a single context element—uniquely identified by entity type and URI. For each level, exactly one Context element provides the granular representation of the context element.

Content provides metadata on context source, confidence, and extraction timestamp. References to level and type facilitate validity checking against the hierarchy definition.

Table 4.6 lists different types of context hierarchies. Activity and Organization hierarchy consist of five levels. Identical context types apply to multiple levels as the level only identifies the expected granularity of context information, while the type describes the actual context data. The Activity model and entity model allow for unlimited hierarchies. We limit the hierarchies to five levels for practical reasons. Hierarchies for DeviceStatus and Reachability comprise four levels. Device Status provides increasing information about hosted services. Reachability defines (general) availability on the upper levels and specific device capabilities and communication channel details on the lower levels. Potential other hierarchies include location (similar to postal addresses including floor and room level), time, as well as temporal and spatial distance.

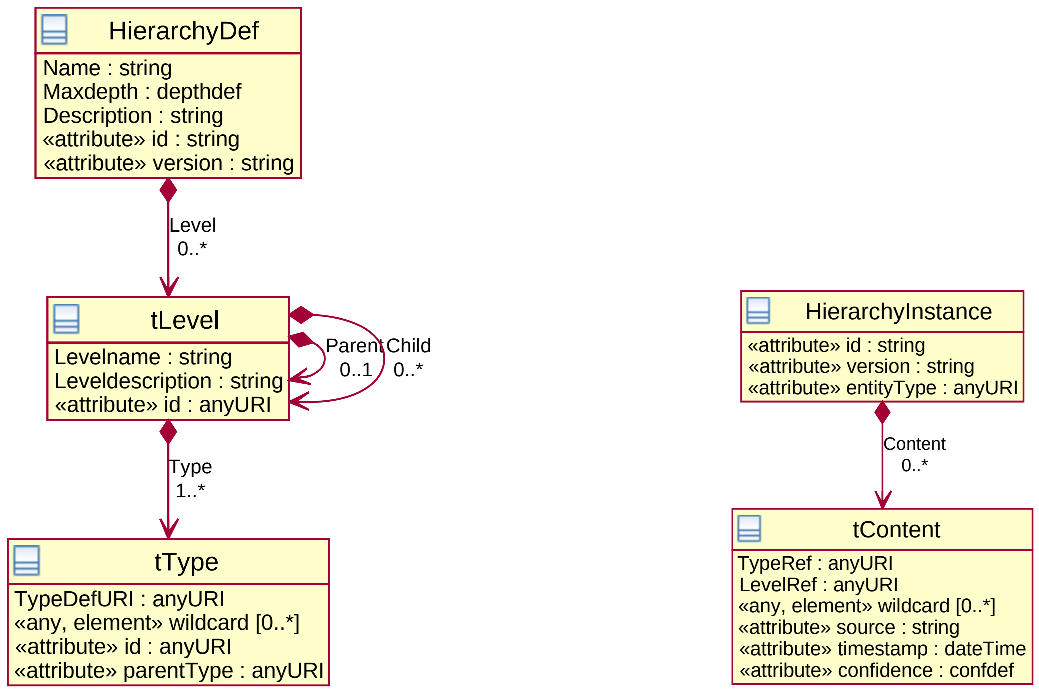


Figure 4.17: Hierarchy definition and hierarchy instance UML class diagram.

It is neither sensible nor possible to describe all available context information in a granular fashion. Only information subject to frequent changes should be structured this way to allow for a fine-grained access and update mechanism. The further up in a hierarchy an update occurs, the more significant it is.

Defining hierarchies that structure context of a single type such as location or time is rather straightforward. This process becomes more complex, once concepts from different domains are included that feature no natural ordering of granularity levels. Modeling ensemble status including humans, services, roles, activities and resource distribution is non-trivial. The context consumer decides whether, for example, information on collocated entities or their activities describes more detailed information. This situation is resolved by either defining an a-priori ordering of levels, or by dynamically arranging levels based on context information.

Context hierarchies exhibit three major beneficial characteristics. First, granularity enables fine-grained access mechanisms for bandwidth economical context provisioning. We present a hybrid context sharing mechanism in the next section. Second, context granularity allows resource constraint devices to focus on their manageable level of detail and thus limit context processing and storage. Third, context hierarchies provide a means to mitigate unreliable context information. In contrast to conventional context systems, granular context provides multiple confidence values for every context element. We require all confidence values to grow monotonically from the most fine-grained up to the most coarse-grained level. This reflects the accuracy of a piece of context information and not

	Activity	type	Organization	type
L1	Environment	[Work, Home]	Organization	Identifier
L2	Project	Activity	Section	Identifier
L3	Activity	Activity	Department	Identifier
L4	SubActivity	Activity	Group	Identifier
L5	Execution	Action	Team	Identifier
	DeviceStatus	type	Reachability	type
L1	AvailableServices	ServiceInfo	Connected	[Yes, No]
L2	AbstractLoad	[LOW,MED,HIGH]	Status	[Online status][Away status]
L3	PercentageLoad	[0,100]	Device	Device(s) details
L4	RunningServices	ServiceInfo	ChannelDetails	ContactInfo(s)

Table 4.6: Context hierarchy examples.

the sensor supplying raw data. Confidence values at every level yields another advantage. Context-aware applications need no longer consider the implicit confidence characteristics of each sensor but can rely entirely on the value for each level.

4.5.2 Hierarchy-based Sharing

We introduce a hybrid, hierarchy-aware context sharing mechanism in this section. Implementation specific details on framework architecture and interface descriptions are provided in Section 7.3.4.

Context provider and context requestor apply a combination of push and pull based mechanisms for context transfer. Pure push-based techniques generate unnecessary traffic when propagating context events at inconvenient time or at overly detailed granularity. Pure pull-based techniques need to trade off network load and polling intervals. Context events occur too irregular to efficiently poll at regular intervals. Thus, short intervals yield context in a timely fashion but cause excessive network load independent of available context events. We combine and enhance these mechanisms in two ways.

First, we enable subscribers to define event conditions. Context requestors specify hierarchy, level of detail and context class independent of a-priori predefined topic trees. Condition-based subscriptions are not new per se but lack the notion of information granularity.

Second, we couple context notifications with subsequent query requests. Local context determines the relevance of incoming remote context information. Thus, client-side context changes can require querying for additional—more detailed—context information from the context provider. A viable strategy is subscribing to coarse-grained availability information and subsequently retrieving fine-grained device status as required.

Our sharing mechanism builds on the usual three message types: Subscription, Query, and Notification (serving also as Query response).

Subscriptions define the entity (or role), level, and type for which to receive notifications. Optionally, it is possible to state a minimum confidence value, transition type (if an entity has reached a certain state, or left it), notification type (whether to receive an initial notification about the current state or just future events) and detail type (which segment of a hierarchy: only values at the exact given level, above, below or all). The meeting service's subscription on the team members' activity status is given in Listing 4.2.

Queries contain the same details as subscriptions except for confidence value and notification type.

Notifications contain context data of exactly one possible path through a particular hierarchy tree. Each level contains only one type object. A notification comprises of multiple type objects each stating their respective level and hierarchy. Each level provides context metadata such as confidence, context source, and timestamp. High-level context changes intrinsically include low-level context changes. Consequently, a context event at a particular level triggers notifications for all subscriptions on that level and below.

```
<Subscription xmlns:ns2="http://ns1/vimocos/sharing"
    detailtype="UPPERINCL"
    notificationtype="ALL"
    transitiontype="TO" xmlns="">
    <ns2:entity>Alice</ns2:entity>
    <ns2:hierarchyId>ns2.activity.ActivityHierarchy</ns2:hierarchyId>
    <ns2:levelId>L3</ns2:levelId>
    <ns2:typeId>ns2.activity.Activity</ns2:typeId>
    <ns2:minConfidence>50</ns2:minConfidence>
</Subscription>
```

Listing 4.2: Example subscription statement: request notifications for any activity events concerning Alice. *L3* and *UPPERINCL* restrict the notifications to changes in the top three levels of her activity hierarchy—expecting a minimum confidence of 50. Following namespaces substitutions apply: ns1 for www.vitalab.tuwien.ac.at and ns2 for at.ac.tuwien.vitalab.vimocos.

4.5.3 Evaluation of hierarchical context sharing

We observe message sizes in a series of test runs to derive the average size for each message type given in Table 4.8. We then analyze the benefit of hierarchy-based context sharing by calculating the reduction of transferred context data for the following three aspects.

1. A hybrid approach of queries and subscriptions to context information reduces protocol overhead compared to pure push-based solutions.

Nr	From	To	S/Q	Hierarchy	Level	Type
0a	Context	Alice, Bob, Carol, Dave	Sub	Reachability	L1	exact
0b	Context	All entities	Sub	DeviceStatus	L1	exact
2b	Coordination	Alice, Bob, Carol, Dave	Query	Reachability	L1	exact
3a	Coordination	DaveLaptop, Dave-PDA, CarolSmart-phone	Query	DeviceStatus	L3	lowerincl
5b	Coordination	Alice, Bob, Carol, Dave	Sub	Activity	L3	upperincl
6b	Coordination	Alice, Bob, Carol, Dave	Query	Reachability	L1	lowerincl

Table 4.7: Subscriptions and Queries in the motivating scenario applying matching on level (not exact values), as this is sufficient here.

Message type	Size (byte)
Subscription Request	1200
Subscription Response	810
Unsubscribe Request	690
Unsubscribe Response	690
Notification Envelope	900
Query Request	710
Query Response Envelope	400

Table 4.8: Mobile context sharing protocol SOAP message size (excluding HTTP overhead). The values for Notification and Query Response messages omit the context payload.

2. Granularity-based subscriptions reduce the amount of overly detailed context notifications.

3. Selection of partial hierarchies reduces context transfer to the requested levels of detail.

Our hybrid approach reduces the message overhead by substituting queries for short-lived subscriptions. We compare query request and response overhead to a subscription roundtrip (consisting of a subscribe request, response and one notification).

Based on the data from Table 4.8, the pull based approach outperforms short-lived subscriptions by almost 3 to 1 (1100 bytes to 2910 bytes). These calculations do not include context payload. The advantage of the pull mechanism is even higher if we consider unsubscribe requests and responses. The scenario involves queries and subscriptions listed in Table 4.7.

We compare level-based subscription and hierarchy-unaware subscription for two settings (Table 4.9). For a five-level hierarchy we assume subscriptions to be evenly spread. In case 1, events occur on all levels with equal likelihood. In case 2, fine-grained changes

Level Case 1	Sub.	Events	Nfy w/	Nfy w/o	Improvement
L1	1	1	1	5	
L2	1	1	2	5	
L3	1	1	3	5	
L4	1	1	4	5	
L5	1	1	5	5	
Total			15	25	40%
Case 2					
L1	1	1	1	15	
L2	1	2	3	15	
L3	1	3	6	15	
L4	1	4	10	15	
L5	1	5	15	15	
Total			35	75	53%

Table 4.9: Event count for level-based subscription mechanism (Nfy w/) and a hierarchy-unaware subscription mechanism (Nfy w/o). Subscriptions are evenly spread across levels (one at each level). Case (1) exhibits events occurring equally likely at each level. In case (2), L5 events are five times more likely than L1 events.

happening more often that coarse-grained changes. In both cases, level-based subscription significantly reduces the number of notifications, in case 1 by 40% and in case 2 by 53%.

Finally, we evaluate further message size reductions by means of transmitting partial hierarchies. Table 4.10 lists the average context content size for events at each level for three example hierarchies.

To obtain these data, we created random (within a certain scope of choice) hierarchy data for four (respectively six) entities[4]. Then, queries at each level and data type were issued and the response size collected. We then aggregated the value of each level from the available entities and test runs. For queries and subscriptions in our scenario (as listed in Table 4.7), we achieved an improvement of 29% up to 76% of payload reduction.

Notifications and query responses exhibit the same data structure. Thus, push and pull based context retrieval benefits from applying partial hierarchies on context data.

In general, the right choice of subscriptions and queries as well as the required level and return type greatly influence the amount of data transmitted and exhibits a lot of potential for improvement beyond these results.

[4]The entities were: Alice, Bob, Carol, Dave as well as AlicePDA, BobLaptop, CarolLaptop, Carol-Smartphone, DavePDA, and DaveLaptop, respectively.

	full	exact	lowerincl	upperincl
Activity				
L1	3368	636	3368	636
L2	3368	783	2958	1193
L3	3368	675	2442	1642
L4	3368	1068	1953	2484
L5	3368	1111	1111	3368
Reachability				
L1	2724	639	2724	639
L2	2724	615	2318	1026
L3	2724	831	1932	1624
L4	2724	1334	1334	2724
DeviceStatus				
L1	2508	1043	2508	1043
L2	2508	674	1705	1477
L3	2508	692	1271	1929
L4	2508	818	818	2508

Table 4.10: Average context query results in bytes for Activity hierarchy, Reachability hierarchy and DeviceStatus hierarchy.

Chapter 5

Service Adaptation Mechanisms

5.1 Service Adaptation Approach

Major challenges emerge from the unpredictable nature of interactions in service ensembles. Changing requirements cause some system properties to gain importance while other properties lose significance. We define the impact of a property as the extent to which services of one property value (e.g., location A) forward requests to services exhibiting a different property value (e.g., location B). One fundamental problem is to continuously identify the most important properties—location, organization, various service capabilities—for service adaptation.

Traditional approaches to service management are no longer feasible as ensembles provide services for joint efforts involving a few participants up to a few thousand participants. The emerging complexity no longer allows for manual tracing of requirements and execution of reconfigurations. Any approach to service self-adaptation needs to address following key service ensemble characteristics:

- The service's decision to pass on a request is context dependent (e.g., load, policies, neighboring services) and thus cannot be observed by looking at the service's capabilities alone.
- Services hide their internal state. Only a limited set of properties is publicly accessible (e.g., owning organization, location, type, capabilities).
- The flow of service requests is non-deterministic; there are no predefined process descriptions.
- A single service obtains merely a local view on all interactions. Due to scale, it observes only service interactions with direct neighbors.

We have discussed the two main design principles for autonomic adaptation in Section 2.4 of related work. Systems implementing an explicit feedback loop (Kephart and Chess 2003) work on a central set of goals thereby requiring a complete view of all managed elements. Emergence-based systems exhibit no central control and yield self-adaptive behavior arising from local interactions between elements (Wolf and Holvoet 2004). The former approach lacks scalability and requires centralized control, but enables simple detection of adaptation needs. The latter approach exhibits the inverse properties. Individual elements cannot perceive the requirements of the overall ensemble. Moreover, individual elements have great difficulty detecting changes in relevant system properties.

We envision a framework combining these two design principles (Figure 5.1). *Monitoring* captures service interactions and public service properties. The *Analysis* component identifies promising properties for further *Planning*. *Execution* provides basic management functions such as service service selection and ranking. The *Knowledge* part provides ensemble context and ensemble configuration. The framework approximates the MAPE-K cycle for autonomic elements (Kephart and Chess 2003). The lack of central control, however, requires the individual services to trigger the final execution phase.

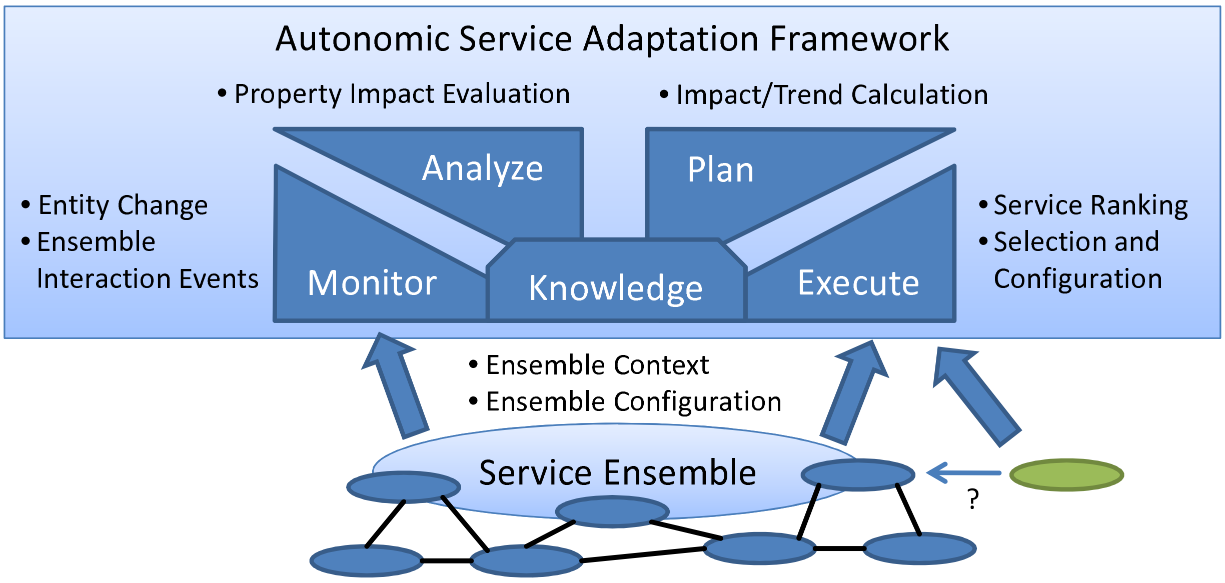

Autonomic Service Adaptation Framework
• Property Impact Evaluation
• Impact/Trend Calculation
Analyze
Plan
• Entity Change
• Ensemble Interaction Events
Monitor
Knowledge
Execute
• Service Ranking
• Selection and Configuration
• Ensemble Context
• Ensemble Configuration
Service Ensemble
?

Figure 5.1: Ensemble Adaptation framework.

5.1.1 Service Adaptation Scenario

The following scenario motivates service self-configuration. Assume a storage service provider participating in a global data service network. A research center becomes a customer in the early phases of a data-intensive project. At the beginning, the need for extensive storage space is low, retrieval requests origin at a single location, and updates occur frequently. Thus requests will mostly happen within the service provider's service network, locally concentrated.

The service interaction characteristics change once data intensive research results are made available for a broader audience. Requests cross the storage provider's boundaries, access to data occurs from multiple locations, while updates decrease.

Suppose a new storage service is about to join the ensemble. It does not know the clients it will serve. It is also unaware of the particular service interaction characteristics when serving these clients. The new service, however, needs to learn of the most significant impact factors to optimally select amongst the existing services for storing and querying data in the ensemble (Figure 5.1 right most service). Services provide storage for multiple clients, thus we need to establish the relevant set of existing services for each of these contexts. For the remainder of this chapter, we discuss our approach and findings in the scope of one client for sake of clarity.

5.1.2 Service Adaptation Process

For a freshly added service, the significant services are the ones most likely to accept forwarded requests. To this end, we need to identify the factors that determine whether a request is accepted or not. Our approach, thus, focuses on public service information and observable service interactions. In the early stages of our scenario, services with versioning capability are suitable receivers. In later stages, services at remote storage providers (i.e., different organizations) or different locations provide most benefit by distributing load.

Figure 5.2 visualizes the approach comprising the following steps: based on the distribution of property values across services, we derive candidate properties (1). These candidates yield high potential impact on service interaction. For example, when all observed services reside within a single data center, location yields no interaction impact. Similarly, the service identifier property yields no impact neither, as every service exhibits a distinct ID. Thus, neither location nor service identifier become candidates. Any changes in service properties (including new/leaving services) trigger recalculation of candidate properties. As long as requests traverse only services of one storage provider, *Organization* will not become a candidate. Once the customer in our example enables access to data for 3rd parties, requests from external services will occur. As multiple organization values emerge, the organization property becomes a potential impact factor.

The subsequent detailed interaction analysis (2) considers only the properties with highest potential impact (e.g., versioning capability). Interaction analysis determines whether services tend to interact with services exhibiting the same or different properties (3). In the early stages of our scenario, services without versioning capability will forward requests to services with versioning capability. These in turn, will forward only between their kind.

Impact magnitude influences the final ranking order of suitable services (4). The versioning capability will exhibit highest impact on the ranking result, when forwarding occurs only from non-versioning to versioning services. Later in the scenario, we replace

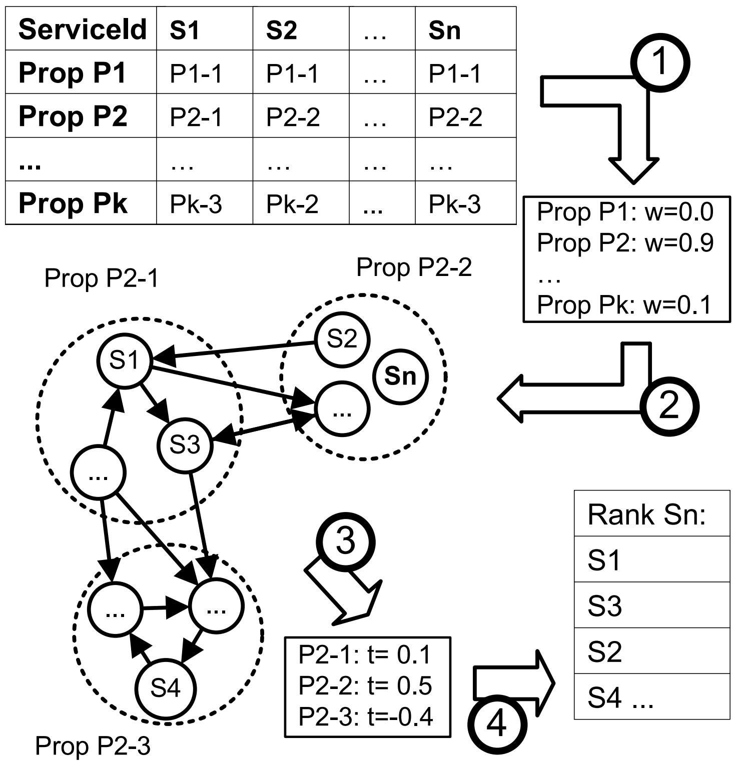


Figure 5.2: Property checking, evaluation, and ranking.

the ranking criteria as capabilities become less significant, while spatial and organizational properties emerge.

In (Dorn, Truong, and Dustdar 2008), we introduced specific human-centric ensemble metrics measuring location, organization, coordination, interaction, and resource utilization aspects. In the following sections, we concentrate on one especially versatile metric and demonstrate its applicability for emerging service selection. Specifically, this metric aims at determining the most relevant services to forward a request to. To this end, we identify and analyze service properties (i.e., potential system impact factors) with the most significant effect on service interactions.

5.2 Property Entropy Model

In large-scale service ensembles, service interaction analysis is a computationally intensive task. Knowing which aspects will yield the most significant findings maximizes the efficiency of the analysis process. The primary purpose of a suitable metric is thus to identify those properties that potentially have a measurable impact on interactions. Such a metric must work on properties consisting of any number of values, and enable comparison of properties that differ in their amount of values. Example service properties include the organization deploying the service, the service location, storage capacity, and request routing capability (e.g., none, random neighbor, round-robin).

The following model and entropy metric calculates the distribution of properties across services. Table 5.1 gives a summarized explanation of the symbols applied in the model and impact algorithms.

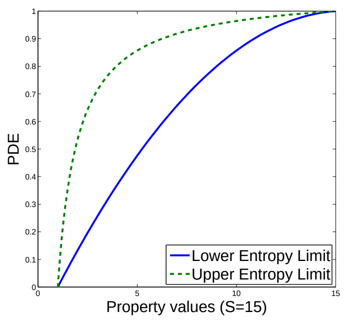
The metric output for each property is in the interval $[0, 1]$. A metric value v towards 0 describes a trend of services sharing the same property values, while a metric value towards 1 denotes services exhibiting individual property values. Extreme cases include all services having the same property value ($v = 0$) and each service having a distinct property value ($v = 1$).

Symbol	Meaning
$\mathcal{S}$	the set of services $s \in \mathcal{S}$ in a service ensemble $\mathcal{C}$
$\mathcal{P}$	the set of public properties in the service ensemble $\mathcal{C}$
P	a particular public property $P \in \mathcal{P}$ comprising any number of property values $p_i \rightarrow p_n \in P$
$\mathcal{F}$	a function mapping each service s to one property value p for each public property P
$PDE(S, P)$	the property distribution entropy for particular property P and service set $\mathcal{S}$
$PDE_{lower\|upper}$	a function describing the minimum (maximum) PDE values for a given number of property values $p \in P$
$util_{upper\|lower}$	a function describing the minimum (maximum) utility along the lower (upper) PDE limits.
E	set of interaction edges in the directed service interaction graph $\mathcal{G}$
$cluster_P(i)$	set of services exhibiting the same property value $p_i \in P$
$trend_P(i)$	interaction focus (internal or external) of a cluster associated to $p_i \in P$
$imp_P(i)$	interaction impact of a cluster associated to $p_i \in P$
imp_P	overall interaction impact of property P
z	iteration count within the zero model analysis

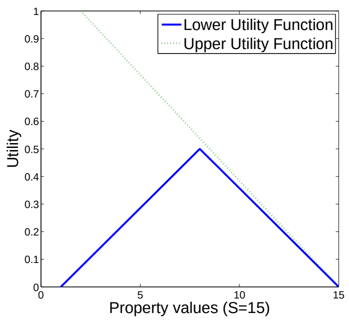
Table 5.1: Symbols applied in the entropy model (upper section) and evaluation algorithm (lower section).

In our model, a service ensemble $\mathcal{C}(\mathcal{S}, \mathcal{P})$ is defined as a set of services $\mathcal{S}$ exhibiting a set of public properties $\mathcal{P}$. Each property $P \in \mathcal{P}$ consist of a set of non-overlapping property values $p_1 \dots p_n$. In addition, for each property P there exists a mapping $\mathcal{F}(\mathcal{S} \mapsto \mathcal{P})$ such that each service $s \in \mathcal{S}$ is assigned to exactly one value instance $p \in P$. For each property P, we define the Property Distribution Entropy (PDE) as follows:

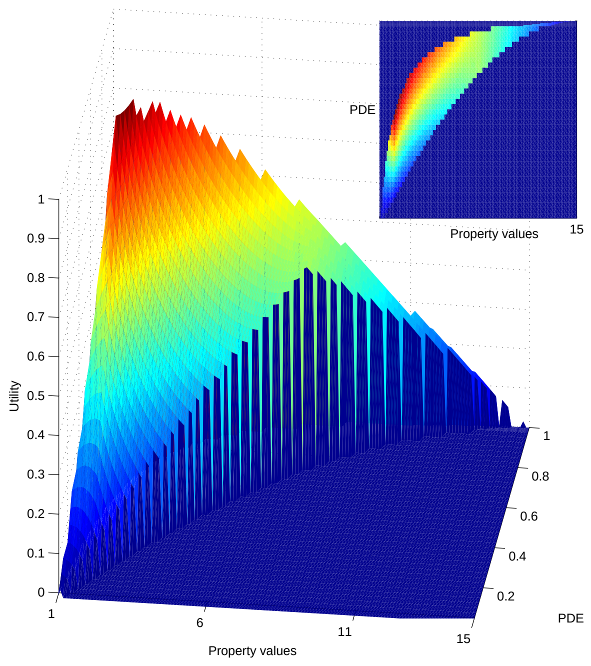
$$\text{PDE}(S, P) = 1 - \sum_{i=1}^{n} \binom{|p_i|}{2} * \binom{z}{2}^{-1} \tag{5.1}$$



(a)



(b)



(c)

Figure 5.3: Entropy limits (a), utility boundaries (b), and overall utility function (c) for $s = 15$

where $|p_i|$ is the number of services mapped to property value $p_i \in P$ and $z = |S|$ is the total number of services in C.

For this entropy metric, there exists a lower and a upper limit given $q = |P|$ and $z = |S|$. Figure 5.3 (a) visualizes the lower and upper entropy limits for $z = 15$ and $q = [1, 15]$. The lower limit describes the most asymmetric distribution of q property values across all services. For any $q = [1, \ldots, c]$ one large group of $z - (q - 1)$ services will share the same property value and $q - 1$ services will exhibit individual property values. The lower entropy limit PDE$_{lower}$ is defined as:

$$\text{PDE}_{lower}(z) = 1 - \frac{q^2 - (2z+1)q + z^2 + z}{z^2 - z} \qquad \textit{with } 1 \leq q \leq z \tag{5.2}$$

The upper entropy limit describes the most symmetric distribution of given property values across all services theoretically possible. There exist q groups of $\frac{z}{q}$ services having a distinct property value. The upper entropy limit PDE$_{upper}$ is defined as:

$$\text{PDE}_{upper}(z) = 1 - \frac{(z - q)}{q(z - 1)} \qquad \textit{with } 1 \leq q \leq z \tag{5.3}$$

The algorithm presented in the next section determines if the impact results in interactions occurring predominantly between services exhibiting the same property values, between services of different property values, or between services without any distinct interaction bias. First, we need to evaluates a property's likelihood of having an impact on interactions.

To this end, we introduce upper and lower entropy utility functions. These utility functions describe the ratio of services that have a choice to communicate either with services of the same property value or with services exhibiting different property values. Only these services generate interactions that exhibit potential property impact (Bollobas 2001, Mcculloh, Lospinoso, and Carley 2007).

The lower entropy utility function $util_{lower}$ corresponds to the lower entropy limit (PDE_{lower}). It reflects the fact, that $q-1$ services can only communicate with services exhibiting a different property value, and thus cannot be included in the impact calculation. Consequently, as individual property values become more common (i.e., entropy value $\rightarrow 1$), the likelihood reaches 0. In contrast, as services increasingly share the same property value (i.e., entropy value $\rightarrow 0$) any interactions across properties must be considered outliers and the likelihood similarly decreases towards 0.

$$\text{util}_{lower}(z) = 0.5 - |-0.5 - \frac{1-q}{z-1}| \qquad 1 < z, 1 < q \leq z \tag{5.4}$$

The upper entropy utility function $util_{upper}$ corresponds to the upper entropy limit (PDE_{upper}). It peaks where all entities are equally distributed across two property values and decreases steadily as the number of distinct property values rise.

$$\text{util}_{upper}(z) = \frac{z-q}{2-q} \qquad \forall 0 < z, 2 < q \leq z \tag{5.5}$$

Figure 5.3 (b) visualizes the lower and upper entropy utility function for $z = 15$ and $q = [1, 15]$. We aggregate upper and lower utility functions in the overall utility function util_{total} defined as follows:

$$\begin{aligned} \text{util}_{total}(z, pde) = & \frac{(pde - PDE_{lower})}{PDE_{upper} - PDE_{lower}} * util_{upper} \\ + & \frac{(PDE_{upper} - pde)}{PDE_{upper} - PDE_{lower}} * util_{lower} \end{aligned} \tag{5.6}$$

where $util_{upper}$ returns the utility value for the upper entropy limit, and $util_{lower}$ returns the utility value for the lower entropy limit. The total value combines the two utility values proportional to the distance of the entropy value and the respective upper and lower boundaries (PDE_{upper}, PDE_{lower}). Figure 5.3 (c) visualizes the overall utility function which provides a likelihood measurement in the interval $[0, 1]$.

5.3 Property Impact Evaluation Algorithm

The PDE model provides the means to identify promising impact factors. In the subsequent step we need to evaluate whether these candidate properties have indeed an impact on service interactions. We define a positive impact of a property value on a group of services when these services tend to communicate with each other (i.e., internal communication), rather than interacting with services exhibiting different property values. A negative impact implies a tendency towards external communication.

We capture interactions between services applying logging mechanisms. The sum of all logged service calls create an interaction network. We define this network as a directed[1] graph $\mathcal{G}(S, E)$ consisting of interaction edges E and services $\mathcal{S} \in \mathcal{C}$ deployed in the ensemble. We denote the set of services exhibiting the same property value a network *cluster*.

For the impact evaluation process (Algorithm 4), we select properties with highest $util_{total}$. For every cluster, the *cRatio* calculates the ratio of property internal to total communication links. The natural link ratio *nRatio* of a cluster in an unbiased network is $|cluster|\ /\ |S|$. To include the characteristics of the underlying interaction network, we create a zero model by distributing all services randomly across clusters of the same size. Multiple rounds of randomization yield a natural deviation of each cluster ratio from the natural ratio. To enable comparison of clusters independent of their natural ratio (*nRatio*) any deviation from *nRatio* is mapped to the interval $[-1, +1]$, where a $trend_P(c)$ of -1 indicates complete external orientation, and $+1$ complete internal orientation. This orientation is defined as:

$$trend_P(c) = \begin{cases} \frac{cRatio_c - nRatio_c}{1 - nRatio_c} & if\ cRatio_c > nRatio_c, \\ \frac{cRatio_c - nRatio_c}{nRatio_c} & if\ cRatio_c \leq nRatio_c \end{cases} \tag{5.7}$$

and the impact of cluster c for Property P is defined as:

$$imp_P(c) = \begin{cases} trend_P(c) & if\ |trend_P(c)| * util_{total}(P) > 2 * dev_c \\ 0 & otherwise \end{cases} \tag{5.8}$$

where $util_{total}(P)$ is the utility of property P and dev_c is the zero model deviation for the cluster c. Taking twice dev_c and reducing further by $util_{total}(P)$ ensures that also for low likelihood values the deviation is sufficiently distinct.

A property p needs not necessarily consist of uniform cluster trends. Internally oriented, externally oriented, and unbiased clusters can coexist. Aggregating all trends proportionally to their corresponding cluster size yields the overall property importance factor:

$$imp_P = \frac{\sum_{i=1}^{n} |imp_P(i)| * |cluster_P(i)|}{|S|} \tag{5.9}$$

We continue to consider only properties with the highest impact imp_P for further interaction analysis as outlined in the following section.

[1]The approach also applies to undirected graphs with the following adaptation: cluster internal links need to be counted twice.

Algorithm 4 Impact Evaluation Algorithm $\mathcal{A}(G(S,E),P)$.

```
function CalculateImpact($G(S,E),P$)
    $Dev \leftarrow$ call $AnalyzeZeroModel(P,G)$
    for all $Clusters\ c \in P$ do
        $nRatio \leftarrow |c|/|V|$
        $cRatio \leftarrow$ call $CalcLinkRatio(c,G)$
        $diff = |nRatio - cRatio|$
        if $diff * util(P) > 2 * Dev[c]$ then
            if $cRatio > nRatio$ then /* Trend towards internal communication. */
                $trend = diff/(1 - nRatio)$
            else /* Trend towards external communication. */
                $trend = diff/nRatio * -1$
            end if
            $setTrend(c, trend)$
        else
            $setTrend(c, 0)$
        end if
    end for
end function

function AnalyzeZeroModel($p,G$)
    $Dev[] \leftarrow \emptyset$
    for $i = 1\ to\ z$ do
        $R \leftarrow randomizeAcrossPartitions(G, clusterSizes(P))$
        for all $Clusterr \in R$ do
            $nRatio = |r|/|V|$
            $cRatio \leftarrow$ call $CalcLinkRatio(r,G)$
            $diff = |nRatio - cRatio|$
            if $cRatio > nRatio$ then
                $dev = diff/(1 - nRatio)$
            else
                $dev = diff/nRatio$
            end if
            $Dev[r] \leftarrow Dev[r] + dev$
        end for
    end for
    for $i = 1\ to\ |C|$ do
        $Dev[i] \leftarrow Dev[i]/z$
    end for
    return $Dev[]$
end function

function CalcLinkRatio($c,G$)
    $intra = countLinksWithinCluster(c,G)$
    $total = countLinksOfCluster(c,G)$
    $edgeRatio = intra/total$
    return $edgeRatio$
end function
```

5.4 Service Ranking Algorithm

The calculation and evaluation of property utility, impact, and impact trend is node independent. When a new service joins the ensemble, the ranking algorithm applies these global metrics to generate a recommendation specific to the newcomer. For the properties with highest impact, we select the cluster identified by the newcomer's properties. For each cluster, we derive its interaction affinity towards other clusters. The affinity function affinity(G, c_1, c_2) describes the likelihood of a new request in c_1 being forwarded to c_2. The special case $c_1 = c_2$ covers internal request delegation. The function is defined as:

$$\text{affinity}(G, c1, c2) = \frac{|links(c1 \rightarrow c2)|}{|links(c1 \rightarrow G)|} \tag{5.10}$$

where $links(c1 \rightarrow c2)$ selects all links starting in cluster c_1 and ending in cluster c_2, respectively ending anywhere in the network G including c_1. In a directed graph, affinity is not reciprocal, thus affinity$(G, c1, c2) \neq$ affinity$(G, c2, c1) \forall c1 \neq c2$.

Our ranking algorithm builds on top of any existing selection mechanism that fulfils following three conditions: (i) returned candidate services are potential communication partners, (ii) services are ranked by their domain specific capability, (iii) services map to ranking scores that reflect the relative match amongst all selected services. A mere list representing the service's rank is insufficient. In case of failing these conditions, our ranking algorithm considers all candidates as equally suitable.

The basic idea is to apply cluster affinity values to update the candidate's rank. Algorithm 5 demonstrates the precise steps. For each candidate and all properties of significant impact, as identified in the previous section, we select the newcomer's cluster $c_{newcomer}$ and the candidate's cluster $c_{candidate}$. We subsequently retrieve the affinity value of $c_{newcomer}$ towards $c_{candidate}$. Candidates in clusters with low affinity are penalized more than candidates in clusters of frequent request forwarding. Affinity values do not modify ranks to their full extend but only proportional to the respective property impact $imp_P(c)$. For each candidate the sum of all weighted affinity values determines the extend to which the ranking result is reduced or increased. Finally, the updated candidate list is sorted again. The newcomer service can then select among the top ranked existing services for successful request forwarding.

Recommending services from clusters that have received many requests in previous rounds achieves desirable preferential attachment characteristics. Independent from the number of services, the recommendation algorithm ensures its persisting applicability as the service network grows.

5.4.1 Discussion of Computational Complexity

The computational complexity of our approach depends on following factors: the total number of services S, the number of public properties $\mathcal{P}$ and their respective values $p \in P$,

Algorithm 5 Update Ranking Results $\mathcal{A}(new, R, PP)$.

```
function RankingResultUpdate($new, R, PP$)
    /* Modifies the ranking results based on property importance and affinity */
    for all $ResultEntry\ r \in R$ do
        /* affw collects all effects on candidate rank */
        $affw = 0$
        for all $Property\ P \in PP$ do
            $c_{newcomer} \leftarrow getCluster(P, new)$
            $c_{candidate} \leftarrow getCluster(P, r)$
            affinity = calcAffinity($c_{newcomer}, c_{candidate}$)
            affw = affw + affinity $*$ $impact(P)$
        end for
        $updateRank(r, getRank(r) * affw)$
    end for
    $sort(R)$
end function
```

the number of service interactions E, and the number of graph randomizations z. Table 5.2 lists the worst case runtime complexity for the various processing steps.

Step	Complexity
Service to Property Mapping	$\mathcal{O}(\mathcal{S} * \mathcal{P})$
Entropy Calculation	$\mathcal{O}(\mathcal{P} * p)$
Interaction to Cluster Mapping	$\mathcal{O}(E * \mathcal{P})$
Cluster Analysis	$\mathcal{O}(\mathcal{P} * p^2)$
Zero Model Analysis	$\mathcal{O}(E * \mathcal{P} * z)$

Table 5.2: Runtime Complexity

From this overview, the *Cluster Analysis* appears to inhibit scalability the most. However, by restrict analysis to properties with highest entropy value PDE, the maximum value of observed property values p will grow slower than the number of total services.

5.5 Evaluation of Service Adaptation

This section demonstrates the effectiveness of our approach based on the motivating scenario. This includes a step by step walk-through of metric computation and analysis of multiple properties. The second part of this section focuses on simulation of a service network.

Property	PDE	PDE_{lower}	PDE_{upper}	$util_{total}$
Loc	0.945	0.835	0.957	0.411
Org	0.802	0.396	0.808	0.826
Cap	0.626	0.275	0.718	0.772

Table 5.3: PDE, limits, and utility values for Location, Organization, and Capability properties.

Property										Total
Location	L1	L2	L3	L4	L5	L6	L7	L8	L9	
Impact	0	0	0	0	-1	0	0	0	0	0.07
Organization	O1	O2	O3	O4						
Impact	0	0	-0.95	0						0.20
Capability	C1	C2	C3							
Impact	-0.86	-1	-0.92							0.92

Table 5.4: Property Impact Evaluation Results

5.5.1 Scenario

We observe a limited number of services in the ensemble network for sake of clarity. The recommendation process observes three public properties: (i) Location (L1. . . L9), (ii) Organization (O1. . . O4), and (iii) Capability (C1. . . C3). Table 5.5 (upper part) outlines the mapping of 14 existing services and one newcomer (S15) to the three properties. This configuration yields the property distribution entropy metric (PDE), corresponding entropy limits (PDE_{upper}, PDE_{lower}), and respective utility in Table 5.3.

Analyzing the weighted interaction graph in Table 5.5 (lower part), we detect the impact values depicted in Table 5.4. For *Location* and *Organization*, we derive impact only for L5, respectively O3, in both cases a strong external trend. For *Capability*, the interaction graph results in a strong external trend for all three property values (C1, C2, and C3). Hence, for service S15 with properties (L9, O3, C1) and randomly chosen neighboring services (S2, S4, S7, S9, S11, S12, S14), we arrive at the ranking results printed in the rightmost column of Table 5.5.

Service S2 is ranked highest. As property *Capability* has the strongest impact on the interaction network, we put most weight on affinity values amongst property values C1, C2, and C3. In our scenario, services of type C1 tend to forward requests to service of type C2, C2 to C3, and C3 back to C1. The ranking result thus recommends service S15 to forward requests primarily to S2 as S2 is the only neighbor of S15 exhibiting property C2.

Id	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15
Loc	L1	L2	L1	L2	L3	L4	L5	L6	L7	L8	L9	L4	L8	L9	L9
Org	O1	O2	O3	O4	O1	O2	O3	O4	O1	O2	O3	O4	O1	O2	O3
Cap	C1	C2	C3	C1	C2	C3	C1	C2	C3	C3	C3	C3	C3	C3	C1

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	Rank
S1	0	0	0	0	35	0	0	0	0	0	0	0	0	0	-
S2	0	0	0	0	0	0	0	0	0	0	0	0	0	43	91.50
S3	33	0	0	6	0	0	0	0	0	0	0	1	0	0	-
S4	0	33	0	0	0	0	0	5	0	0	0	0	0	0	17.77
S5	0	0	0	0	0	0	0	0	0	26	0	0	8	0	-
S6	2	0	0	0	0	0	27	0	0	0	2	0	0	0	-
S7	0	0	0	3	16	1	0	0	1	3	0	0	0	0	3.05
S8	0	0	0	0	0	0	0	0	0	31	1	0	0	0	-
S9	3	0	0	0	0	0	29	0	0	0	0	0	0	0	15.37
S10	0	3	0	28	0	0	0	0	0	0	2	1	2	0	-
S11	0	0	1	32	2	0	0	0	0	0	0	1	0	0	4.95
S12	0	0	0	37	0	0	0	0	0	0	0	0	0	0	13.74
S13	47	0	0	0	0	0	0	0	0	0	0	0	0	0	-
S14	0	0	1	29	2	0	0	0	0	2	0	1	0	0	5.57

Table 5.5: Service network: weighted directed graph including ranking results for S15.

5.5.2 Simulation Setup

Simulation-based evaluation allows for analyzing our recommendation algorithm under changing conditions with respect to property count, property impact, service network size, and impact fluctuations. We focus only on the behavioral characteristics of our algorithm and do not consider the costs of network monitoring. Chen et al. (Chen, Bindel, Song, and Katz 2007) follow an algebra-based approach to efficient network monitoring.

The simulation environment consists of $|\mathcal{S}| = n$ services. Each service exhibits $|\mathcal{P}| = m$ property values, corresponding to m distinct properties. Services have the capability to forward a received request to another service from their service neighborhood h or reject it. For each property, an acceptance matrix $\mathcal{M}$ simulates the impact of current requirements on the service interaction structure. The matrix provides the likelihood of any service with property value p_i to accept a request from a service with property value p_j. As the simulation progresses, we adapt the importance weight of the various property matrixes to reproduce the dynamic requirement changes. Table 5.6 provides a snapshot of an acceptance matrix for property *Organization* comprising four property values. In this example, request forwarding occurs in a circle.

In each simulation round, services receive r randomly assigned requests. Each service then selects a member from its neighborhood to forward the request to. The receiving service then chooses to accept or deny the request. In the former case, the request is considered successfully completed. In the latter case, the sending service receives 1 penalty point and has to find another service to forward the request to.

from/to	O1	O2	O3	O4
O1	0	1	0	0
O2	0	0	1	0
O3	0	0	0	1
O4	1	0	0	0

Table 5.6: Example acceptance matrix $\mathcal{M}$ for four organization property values $O1 \ldots O4$ exhibiting maximal constraints.

Although services apply the acceptance matrix for incoming requests, they do not utilize this information for outgoing requests. Instead, they engage the proposed ranking algorithm. The algorithm then applies the analyzed public properties and service interactions as described in the previous sections. To eliminate any effects of domain specific ranking, the simulation assumes all services are equally able to process a request. We calculate the benefit of our recommendation algorithm by comparing the penalty a newcomer service receives when contacting neighbors by trial-and-error and when contacting the recommended neighbors.

In all experiment iterations, we assign random requests to services each round to simulate service load fluctuations. To keep the overall network load constant, however, the average number of assigned requests per service is fixed at $r = 20$.

5.5.3 Measuring Scalability

First, we demonstrate the scalability of our approach. We increase the number of services (n), service neighborhood (h), and property values (p_m). In each round, we measure for each newly added service the penalty received in the process of successfully forwarding a single request to a random neighbor, respectively a recommended neighbor.

The initial service network consists of $n = 50$ services, each having $h = 24$ random neighbors. Four properties (P1 ... P4) exhibiting $|p| = 7, 5, 4$, and 4 values respectively exert impact via their acceptance matrixes. As we add a new service, we connect it with random $20 + log(n)^2$ existing services. Additionally, we link random $log(n)$ existing services with the newcomer. For the four properties (P1 ... P4), the simulation introduces new property values at a growth rate of $log(n)$.

Figure 5.4 prints the average benefit for every 50 consecutive benefit measurements over multiple experiment runs. On average, the recommendation-based approach outperforms the trial-and-error approach across scales. At the end of the scalability experiment, the final service network comprises 10050 services, each linked to 105 neighbors on average. Each of the four properties exhibit nine more values, bringing the number of choices to $|p| = 16, 15, 14$, and 13 respectively. The recommendation algorithm yields similar good results for this configuration as for the initial service network.

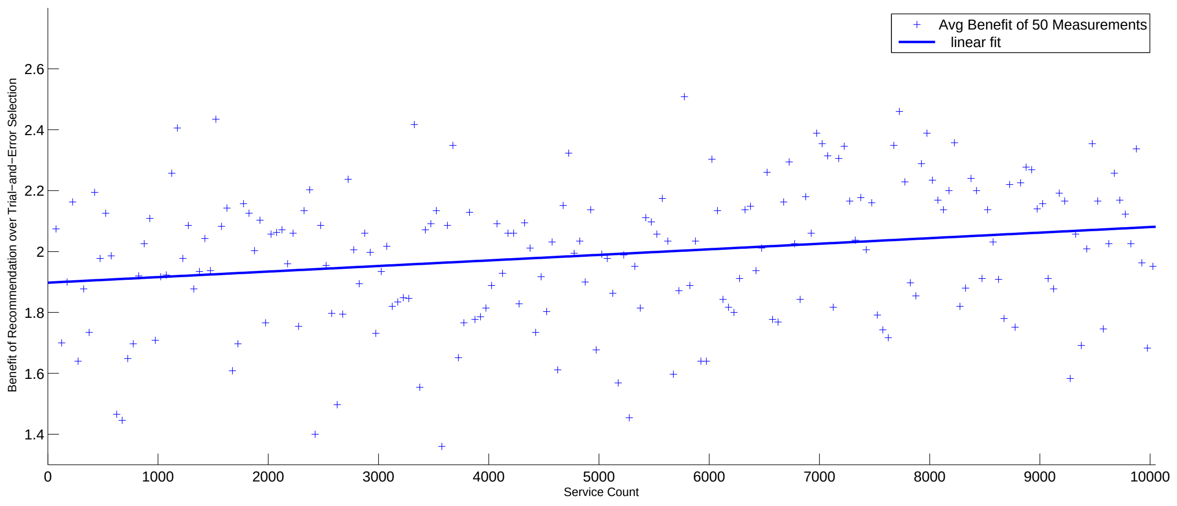


Figure 5.4: Average benefit for service recommendation compared to trial-and-error selection. Numbers display aggregation of 50 new services within a service network growing from 50 to 10050 services.

5.5.4 Measuring Adaptiveness

We have shown scalability for fixed impact of the four properties (P1 . . . P4). Here, we demonstrate the adaptability of our approach. Along these lines, we dynamically change the impact weights of the respective acceptance matrixes ($\mathcal{M}_1 \ldots \mathcal{M}_4$) every 10 rounds while measuring the quality of the recommendation result every round. The number of services $n = 50$, their neighborhood size $h = 24$, and the property values (p_m) remain constant. As we keep the number of services fixed, we select in each round a random existing service to measure the penalties for recommended and trial-and-error neighbor selection.

We analyze 30 experiment iterations, each comprising 100 impact changes. Figure 5.5 prints the benefit (and standard deviation) received for applying recommended selection for each of the 10 rounds after the property impact change. We observe lower—but still positive—benefit measurements for the first two rounds after a change. As the algorithm self-adjusts, average benefit increases to 2.

5.5.5 Measuring Constraint Impact

The realizable benefit heavily depends on the constraints on the service network. When lack of constraints result in high acceptance rates, any random neighbor will most likely be a suitable selection. The ranking algorithm will provide considerable benefit once constraints emerge and begin to increasingly restrict service interactions.

In the third experiment, we start with four properties (each having 10 property values) allowing interactions between any clusters (i.e., the corresponding acceptance matrixes are filled with 1s). Every 10 rounds, we randomly select one particular property and increase the constraints. As we continue to replace random 1s with 0s in the acceptance matrix, the trial-and-error approach yields increasing penalties. We continue increasing the constraints until every acceptance matrix $\mathcal{M}$ contains a single 1 on each row (e.g., Table 5.6). Thus, for every property P, a service of any particular property value $p_x \in P$ only accepts requests from services exhibiting a single other property value $p_y \in P$ (including $x = y$). Throughout the experiment, property impact and service count remain fixed.

Figure 5.6 presents the average penalty difference over 10 iterations of $n = 50$ services having on average $h = 24$ neighbors. Benefits start rising around round 1750. Around 3800, this growth levels off as the constraints can no longer be intensified.

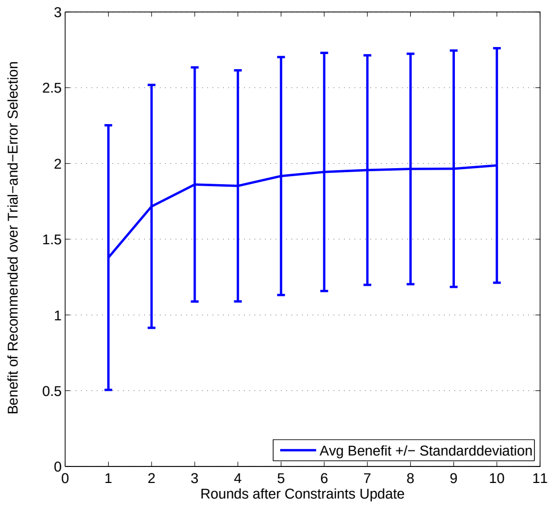


Figure 5.5: Average benefit for each round following a property impact change.

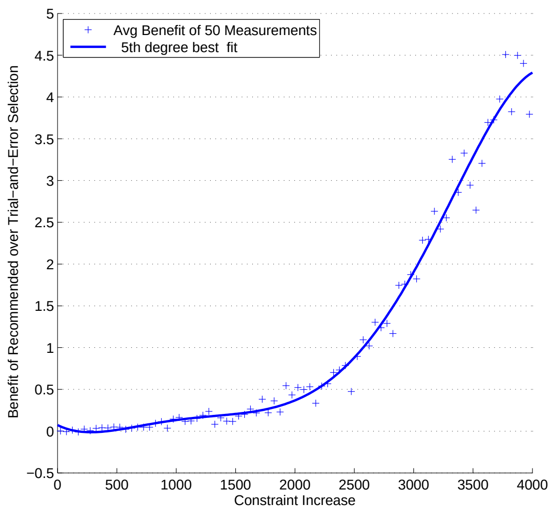


Figure 5.6: Average benefit for service recommendation compared to trial-and-error approach for increasing constraints. Numbers display aggregated benefit of 50 consecutive measurements.

5.5.6 Experiment Discussion

The simulation reflects the key challenges outlined in the introduction to reproduce the constraints found in real world service networks. First, services provide only public information on their various static properties. Second, the decision process for selecting a suitable receiving service relies purely on dynamic information. Third, services accept incoming requests based only on internal, non-observable information (i.e., defined by the acceptance matrix). Finally, no service obtains a complete view on service interactions.

Despite these challenges, our model and algorithms perform significantly better than trial-and-error service selection. When comparing average, absolute penalty measurements (Figure 5.7), the ranking algorithm results in 2.5 times lower penalties during the scalability experiment, and 2.7 times lower penalties during the adaptivity experiment, respectively. The third experiment displays 2.1 times lower penalties averaged over the final 1500 rounds. For both scalability and adaptivity experiments, our algorithm requires on average slightly more than a single forwarding retry (i.e., one rejected request). The trial-and-error approach, in contrast, results in approximately three retries. The constraint measurement displays higher failure rates. Our recommendation algorithm requires less than 2.5 retries, while trial-and-error selection causes 5 rejections.

At this stage, we cannot predict the algorithm's performance in real world implementations. However, our simulations yield very promising results and demonstrate both scalability and adaptiveness of our approach.

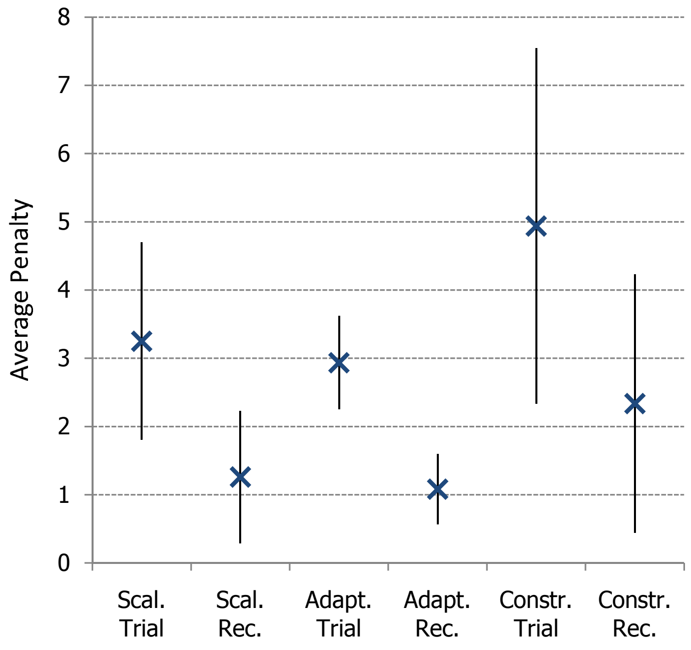


Figure 5.7: Average penalty measurements and ± standard deviation for scalability, adaptivity, and constraints experiments; comparing recommended versus trial-and-error selection.

Chapter 6

Service Infrastructure Adaptation Techniques

Infrastructure Adaptation Techniques target adaptation at the level of a complete ensemble. Individual services apply context to adapt within the scope of their immediate ensemble neighborhood. Services, however, cannot monitor the whole environment. They remain unaware whether their functionality and corresponding adaptive behavior is still appropriate, or whether another service is better suited. Consequently, we propose separation of concerns for adaptation techniques. Services focus on self-adjustment as required by their configuration and purpose. The infrastructure assumes responsibility for monitoring requirement fulfillment and selection of system-wide ensemble adaptation actions. In this chapter, we introduce:

Adaptation Process introducing our infrastructure adaptation methodology based on the MAPE-K cycle of autonomic computing.

Capability Model describing service metadata including configuration aspects.

Requirements Rules observe ensemble metrics and create appropriate capability constraints. These constraints are matched against currently deployed services to identify service capability mismatches.

Requirements Clustering groups a set of requirements when service utility values indicate that a composition of multiple services provides better requirements satisfaction than a single service.

Simulated Annealing-based Composition achieves an optimal trade-off between minimal aggregation costs and maximal requirements fulfillment.

6.1 Infrastructure Adaptation Approach

The infrastructure adaptation process (Figure 6.1) closely resembles the autonomic MAPE-K cycle comprising monitoring, analysis, planning, and execution phases. Specifically, we monitor service capability and ensemble events. Ensemble requirements tracking detects if these events trigger execution of the current set of requirement rules. Subsequent capability-requirement mismatch evaluation determines the need for infrastructure adaptation. Service utility calculation matches the set of available services against the set of unsatisfied requirements. This process step considers not only single services. We also apply clustering to detect service aggregation of higher overall utility. A tradeoff between best matching services, respectively aggregations, and deployment costs provides a ranked set of alterative adaptation configurations. Ultimately, management selects and executes one of the available choices. We outline only the very fundamental adaptation steps in Figure 6.1. The flow chart 6.2 in Section 6.2 contains additional information on involved components and branching conditions. The adaptation process relies upon five building blocks:

Ensemble Context events provide continuous information to update the ensemble context. Ensemble metrics provide various aggregated views of the overall ensemble. The ensemble context models have been discussed in Chapter 2.1.

Service Capabilities are metadata for effective service selection. Details include service classification, usage constraints, and configuration alternatives.

Ensemble Requirements describe necessary and desirable service capabilities for a given ensemble context.

Capability Matching evaluates currently deployed services against requirements for the underlying context. Capability mismatches trigger the adaptation process to find better suited services, respectively service aggregations.

Service Composition Recommendation trades off services best fulfilling required capabilities and costs for deployment, respectively aggregation. Previous service invocations and service aggregations determine suitable service candidates.

In the following section, we discuss the overall adaptation process in detail before providing a comprehensive description of these major building blocks.

6.2 Adaptation Process

This section outline role and place of the core building parts in the adaptation process. In this chapter's introduction, Figure 6.1 presented the general approach, here we go into more detail and discuss the process visualized in Figure 6.2. Chapter 7 provides additional implementation specific details.

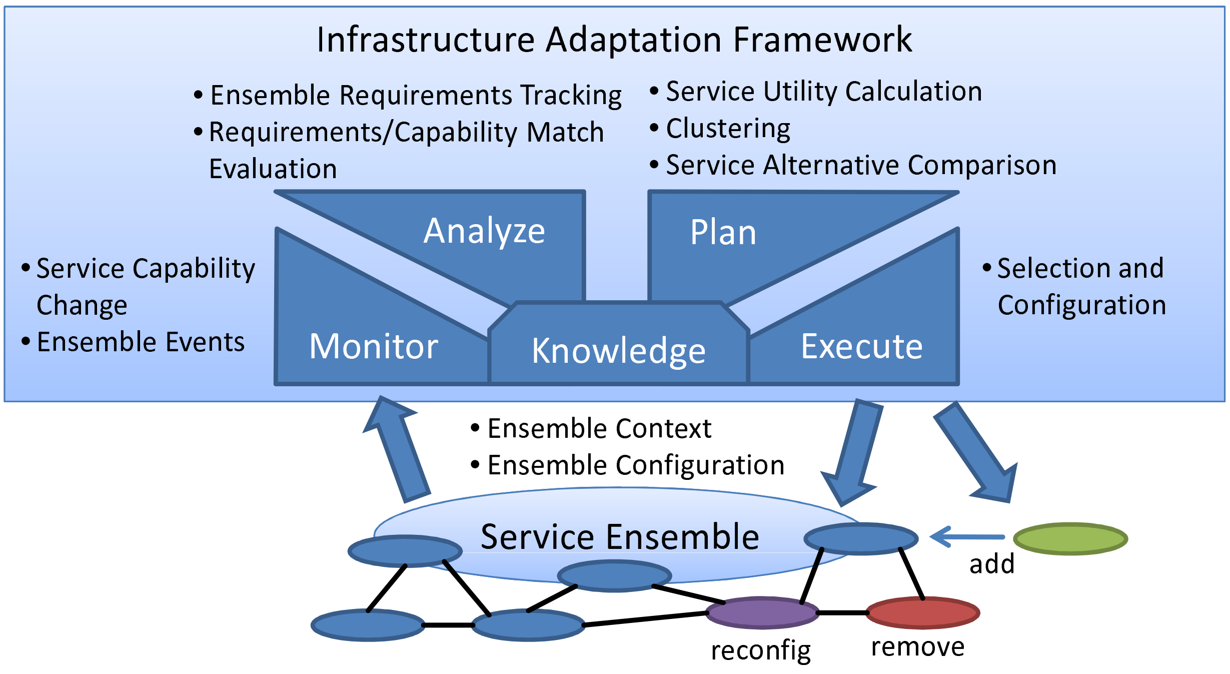

Infrastructure Adaptation Framework
• Ensemble Requirements Tracking
• Requirements/Capability Match Evaluation
• Service Utility Calculation
• Clustering
• Service Alternative Comparison
Analyze
Plan
• Service Capability Change
• Ensemble Events
• Selection and Configuration
Monitor
Knowledge
Execute
• Ensemble Context
• Ensemble Configuration
Service Ensemble
add
reconfig
remove

Figure 6.1: Infrastructure adaptation process overview

6.2.1 Monitoring

There are two types of events triggering the adaptation process. First, service capability events inform the adaptation framework on services subject to capability changes. Figure 7.2 in the implementation chapter visualizes the Capability Change UML model. The *ProfileChange* event lists the profiles that have changed, the individual affected components, and gives details on the specific capabilities that are new, removed, or updated. The *RepositoryChange* event provides references to new, removed, or changed profiles. Subsequently, interested entities need to retrieve event details directly from the repository.

Second, ensemble events cause updates of the ensemble context. These events do not triggering requirements tracking directly since not every event will trigger a change in the ensemble metrics. Section 4.1 contains the details on the ensemble context model, while Section 4.2 outlines the context capturing process. Ensemble metric events themselves simply list the changed metric and provide the corresponding new value.

6.2.2 Analysis

The ensemble configuration comprises two main parts. First, the service configuration lists currently provided services including their capability configuration and their requirement match. The configuration also states the service's requirement fulfillment degree (i.e., the membership) to allow for fuzzy clustering. Second, the *Ensemble Requirements* consist of requirement sets for the various service categories. In each category, a list of requirements

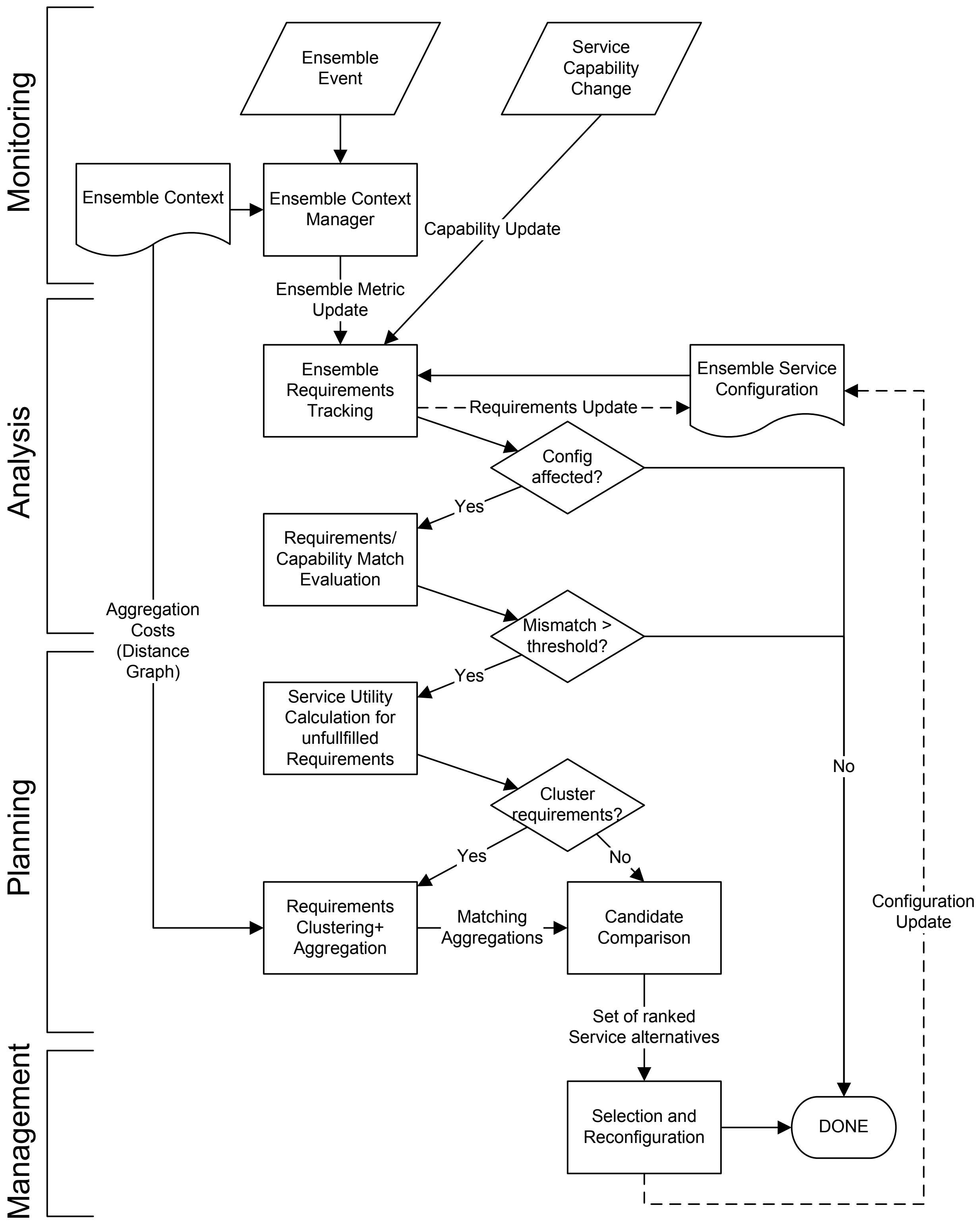

Monitoring
Analysis
Planning
Management
Ensemble Event
Service Capability Change
Ensemble Context
Ensemble Context Manager
Capability Update
Ensemble Metric Update
Ensemble Requirements Tracking
Ensemble Service Configuration
Requirements Update
Config affected?
Yes
Requirements/ Capability Match Evaluation
Mismatch > threshold?
Yes
Aggregation Costs (Distance Graph)
Service Utility Calculation for unfullfilled Requirements
No
Cluster requirements?
Yes
No
Requirements Clustering+ Aggregation
Matching Aggregations
Candidate Comparison
Configuration Update
Set of ranked Service alternatives
Selection and Reconfiguration
DONE

Figure 6.2: Infrastructure adaptation process flow

identify capabilities and properties along with the parameters for the selected utility function. Figure 7.9 in the implementation chapter visualizes the ensemble configuration UML class diagram.

Ensemble requirements tracking identifies those ensemble configurations that are affected by service capability events and ensemble metric events. In the latter case, we merely check if the changed ensemble metrics are decisive in any requirement (e.g., a threshold has now been crossed). If so, we updates the requirements in the ensemble configuration and proceed with matching of requirements to capabilities.

For capability changes, requirement tracking analyzes which ensemble configuration includes the respective service. It subsequently determines if the current set of requirements includes the changed properties or capabilities.

Optionally, we check ensemble configurations for services that provide the same capability, but have remained unchanged. We potentially replace the currently used service with the changed service when the service exchange promises to yield better requirement fulfillment.

In the next step, we rematch requirements and capabilities. In the case of changed service capabilities, we compare existing requirements against updated capabilities. In the case of changed ensemble metrics, we compare updated requirements against existing service capabilities.

We summarize across all mismatch values to evaluate if alterations in the set of provided services is justified. We trigger the components in the planning phase, when the mismatch exceeds a given (domain specific) threshold. From the complete set of requirements, we pass only those requirements to the next stage that are poorly fulfilled.

6.2.3 Planning

Planning is concerned with identifying the best available services for a given set of requirements. To this end, service utility calculation analyzes each service this is fulfilling at least one requirement. Each service receives a utility score for each requirement.

For multiple requirements, the threshold model subsequently decides whether to invoke the clustering process. Clustering assigns each constraint to a cluster with varying fuzzyness. Services are then ranked for each cluster. Simulated Annealing determines a near-optimal aggregation taking into consideration aggregation costs. We focus purely on service interaction distance as aggregation costs in the scope of this thesis. Ultimately, we obtain multiple aggregation candidates.

In case of skipping clustering, we continue directly to ranking the services according to their utility score. We recommend only those services (respectively service aggregations) that provide a better requirement-capability match than the currently configured service(s). When no service provides high enough utility, no service is recommended.

6.2.4 Management

The final procedure of selecting and configuring the recommended services remains outside the scope of this thesis. Potential approaches include automatic configuration such as the the selection algorithm (Alg. 5) in Section 5.4.

6.3 Service Capabilities

Service capabilities describe behavior properties which cannot be directly derived from the service's WSDL document. Example properties include limitations on simultaneous service use, supported resource access strategies, or reconfigurability. Capabilities usually change when a service undergoes major modification. Adding a new operation or extending service back-end resources provides new or better functionality. Services might also choose to reduce capabilities to remain available in spite of high load. This graceful degrading allows service clients to trade-off limited functionality and the cost of finding and invoking an alternative fully functional service. In any case, service capabilities explicitly exclude highly volatile information such as QoS parameters.

The service capability meta model shares some similarities with the *Composite Capability/Preference Profiles* (CC/PP) specification. The original purpose of CC/PP foresees clients to transmit their capabilities in order to allow service providers to adapt delivered content accordingly. In contrast, our approach envisions services to describe their capabilities to enable service clients to select the most suitable service.

The service capability model borrows the concepts of *component* and *property* but goes beyond describing simple service characteristics. Selectable capabilities and supported capability compositions are the main distinct differences to the CC/PP model. These properties are key to reconfiguration and adaptation. The capability meta model comprises the following elements:

Profile contains all capabilities of a single service. The *WSDLlocation* identifies the corresponding service instance. The *ServiceCategory* and *CategoryFit* describe how well a service fits into a given *ActionCategory* or any additional domain-specific category. A profile consists of one or more *Components*.

Component describes a certain function or non-functional aspect of a service. A notification service, for example, will distinguish amongst publication related capabilities and subscription related capabilities. The same mechanism separates operational components from management components. Each component identifies the set of WSDL operations for which the capabilities apply. Especially general purpose operations will appear in multiple components. A component specifies regular capabilities, selectable capabilities, and supported configurations on the selectable capabilities.

Capability comprises properties and optionally sub capabilities. Properties state the actual capability details while sub capabilities enable further structuring. Each capability exhibits a fitness factor. This factor states how well the service supports the particular capability. It ranks the service's behavior in the overall list of services exhibiting this capability. Any restrictions in applicability result in a lower fitness value. This mechanism enables fine-grained service matching and replaceability. To this end, services describe capabilities outside their core competency. They provide existing operations and components for a different purpose. Although they will not yield high fitness values, they become substitutes when specialized services are not available. For example, communication services can serve as coordination services to some extent for a limited time.

Property identifies and provides details such as maximum number of requests per minute. The meta model defines five simple properties for integer, decimal, boolean, timestamp, and string values. FileSize is an example complex property comprising size unit (e.g., kB, mB) and size value.

SelectableCapability describes capabilities that need selection and (optionally) configuration before they become available to the client. The list of alternative capabilities consists of regular capabilities or again selectable capabilities. For each choice, the Selectable Capability defines whether a selection is required and if there is a capability selected by default.

Combination originates in the WS-Policy specification to model valid compositions of policies. Here, a *Combination* defines valid combinations of selectable capabilities. Note, the SelectableCapability element only defines the set of available choices. *Selection* identifies a SelectableCapability (i.e., representing a set of capabilities) or a single Capability. *All* contains a set of SelectableCapabilities, expressing any possible capability combination. *OneOf* contains an exhaustive list of possible Combinations. If no other restrictions are specified, *NoneOf* implies that all combinations are valid except for the listed ones.

Transition describes valid reconfiguration paths. Specifically, a transition contains a minimum of one start configuration (i.e., Combination) and a minimum of one reachable end configuration. A set of positive transitions explicitly lists allowed reconfigurations. Negative transitions implicate all transitions are valid, except the listed ones.

6.4 Ensemble Requirements

Ensemble requirements depend on the current ensemble state and define a desirable ensemble configuration. To this end, we apply event-driven rules. A requirement rule describes metric conditions and subsequent constraints applied to a particular capability. Changes

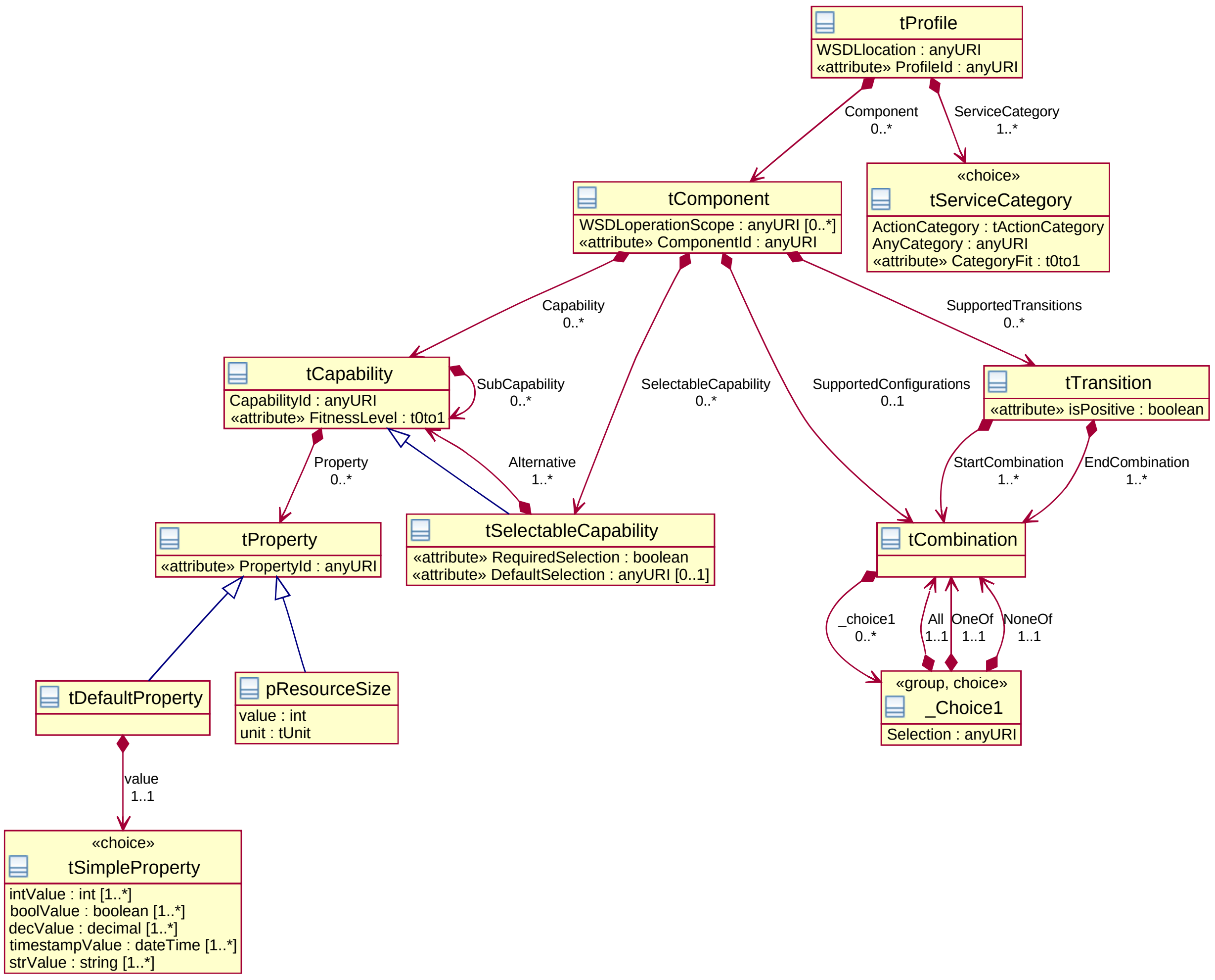

tProfile
WSDLlocation : anyURI
«attribute» ProfileId : anyURI
Component 0..*
ServiceCategory 1..*
tComponent
WSDLoperationScope : anyURI [0..*]
«attribute» ComponentId : anyURI
«choice» tServiceCategory
ActionCategory : tActionCategory
AnyCategory : anyURI
«attribute» CategoryFit : t0to1
Capability 0..*
SupportedTransitions 0..*
tCapability
CapabilityId : anyURI
«attribute» FitnessLevel : t0to1
SubCapability 0..*
SelectableCapability 0..*
SupportedConfigurations 0..1
tTransition
«attribute» isPositive : boolean
Property 0..*
Alternative 1..*
StartCombination 1..*
EndCombination 1..*
tProperty
«attribute» PropertyId : anyURI
tSelectableCapability
«attribute» RequiredSelection : boolean
«attribute» DefaultSelection : anyURI [0..1]
tCombination
_choice1 0..*
All 1..1
OneOf 1..1
NoneOf 1..1
tDefaultProperty
pResourceSize
value : int
unit : tUnit
«group, choice» _Choice1
Selection : anyURI
value 1..1
«choice» tSimpleProperty
intValue : int [1..*]
boolValue : boolean [1..*]
decValue : decimal [1..*]
timestampValue : dateTime [1..*]
strValue : string [1..*]

Figure 6.3: Capability meta model UML class diagram

in ensemble metrics fire corresponding rules which then define optimum service capabilities (i.e., constraints). In the proceeding sections, we then compare deployed services with calculated constraints and compose the best reconfiguration plan given the available capabilities.

When designing rules, we have to consider a number of challenges. First, different ensembles will exhibit different metrics. Thus, rules cannot rely on having all metrics available. Second, ensembles have various goals which reflect in customized additional rules and removal of nonessential rules. Rules must not rely on other rules being active or available. Given the complexity and heterogeneity of requirements, tight coupling of rules is not an option. Third, we need to provide the most fitting services regardless of the requirement fulfillment level available services exhibit. When services lack the required optimum capabilities, we need to find services that support the next highest requirements. Consequently, rules need to enable smooth degradation of provided capabilities.

To this end, we design loosely coupled, weighted rules. Rules depend only on metrics, they do not reference any other rule. Fine-grained rules do not override coarse-grained rules. Instead, they generate constraints of higher importance (i.e., constraints exhibiting a higher weight). When two requirements (not necessarily from the same rule) constrain the same capability, the more important one takes precedence. This mechanism is vital to smooth degrading. When most significant constraint cannot be satisfied, the next most important constraint becomes active. We mitigate any implicit dependency on the firing of other rules by introducing default constraints. The default constraints describe basic capabilities necessary for the service ensemble when no other rule generates more specific constraints on the particular capabilities.

A requirement rule specifies following elements:

Rule Identifier enables requirement tracing. All constraints generated by the same rule carry the same rule identifier.

Metric Conditions trigger the generation of constraints. Rules can aggregate any number of metrics, but must refrain from applying results generated by other rules.

Capability Identifier determines the capability.

Property Identifier determines the property within the capability.

Utility Function Type defines whether the utility function compares linear properties, overlap of selectable capabilities, or extent of selectable capabilities.

Utility Function Identifier defines the candidate comparison function. Besides the linear utility functions introduced in Section 4.3, we provide set comparison functions (see list below).

Utility Function Parameters for linear functions, the parameters provide the limits. For set functions, the parameters list the required capability elements.

Weight describes the importance of constraints. More specific constraints yield higher weights than general constraints. Constraints with weights equal to zero are ignored. We rate default constraints at 0.1 and specialized constraints between 0.5 and 1.

Service Category identifies the type of service the constraint applies to.

We provide the following set of utility functions:

ExistsUtility checks for the availability of a required capability, or non-existance of an undesired capability.

ChoiceUtilityHigh considers all services exhibiting the set of selectable capabilities, and increases the score by additional selectable choices. Thus, services with high configurability yield better scores than services providing the basic, required set.

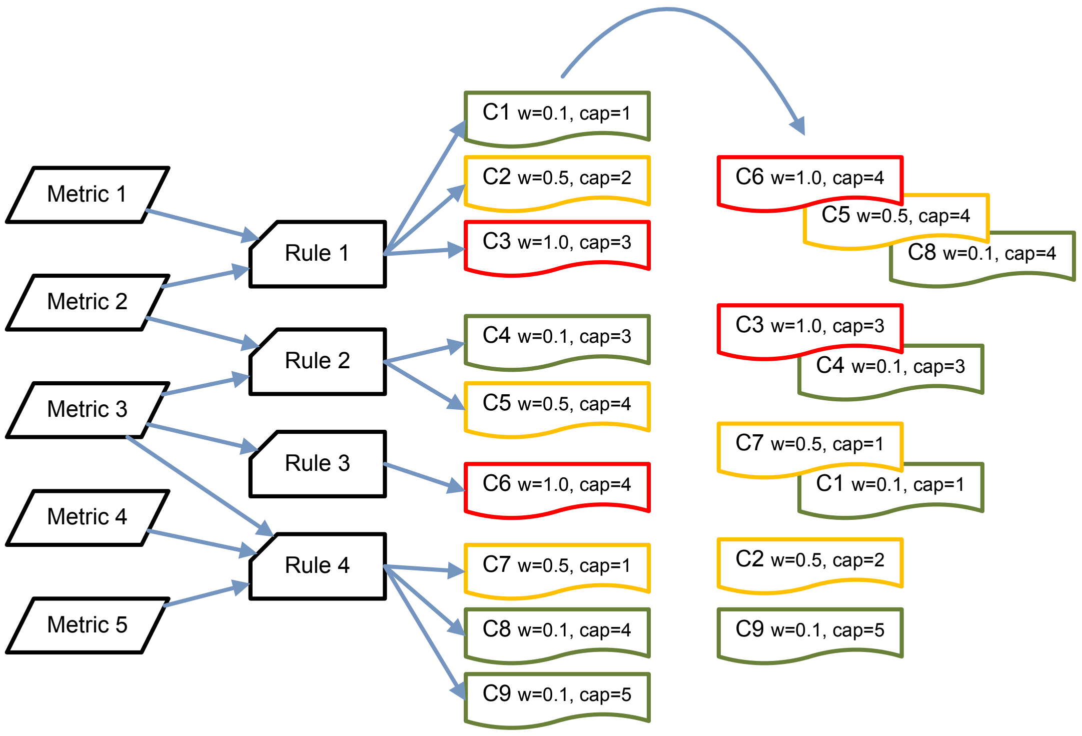


Figure 6.4: Metrics triggering rules which in turn generate constraints on capabilities (*cap*) with weight w.

ChoiceUtilityLow is the inverse of ChoiceUtilityHigh. Services that offer the required set of capabilities and nothing else yield better utility.

SelectionUtilityOne selects any service that exhibits one of the given capabilities. Matching more than one capability does not increase the rank.

SelectionUtilitySome extends SelectionUtilityOne. A higher overlap of given constraint capability and provided service capabilities results in higher utility values.

SelectionUtilityAll requires all constraint capabilities to match the provided capabilities. ChoiceUtilityHigh and ChoiceUtilityLow extend this function.

Consider following example rule—written in DROOLS (for more technical details see Chapter 7). When the Ensemble Location Entropy value exceeds 0, the rule in Listing 6.1 generates a resource storage constraint. The constraint specifies that a service with resource storage capability needs to support at least one folder for each location in the ensemble and defines any service providing more than ten times the required amount as equally suitable. Finally, the rule stores the constraint for the particular ensemble and service category.

We aggregated constraints on identical capabilities and then sort constraints in descending order of weight. Figure 6.4 visualizes the relations between metrics, rules, constraints, and constraint aggregation.

```
rule "ELE_above_Threshold"
dialect "java"
when
metrics : Metrics (metrics.ele > 0)
then
TSimpleDecimalConstraint r = RequirementsFactory.getConstraint(
    "ELE_abovethreshold_check-basic",
    URIs.CAP_ResStorage ,
    URIs.PROP_MaxFoldersPerAccount_ResStorage ,
    ValueUtilitySoftLowerStableLimitedOver.UTILITY_TYPE ,
    ValueUtilitySoftLowerStableLimitedOver.class.getSimpleName()
    new double[]{metrics.getLocationMetric().getClusters(),
                 metrics.getLocationMetric().getClusters()*10,
                 Double.MAX_VALUE},
    0.5d
    );
rcc.addRequirement( metrics.getEnsembleURI(),
                    TActionCategory.EXECUTION , r);
end
```

Listing 6.1: Example DROOLS requirement rule generating a resource storage constraint when the Ensemble Location Entropy (ELE) exceeds 0.

6.5 Capability Matching

6.5.1 Requirements Filtering

Loose coupling of requirements rules renders the rule engine unaware of multiple requirements constraining the same capability. Matching capabilities requires, therefore, prior filtering of multiple—potentially conflicting—constraints on the same capability.

The *Gracefully Degrading Matching Algorithm* 6 determines which constraint comes into operation. For sake of simplicity, suppose that each service profile consists of a single component. Further, let us define the set of candidate components $s \in S$ that we collect from all available service profiles for a particular service category. We capture the constraints aggregated for identical capabilities in $RL = \{R_1 \dots R_n\}$ such that all constraints $c \in R_i$ concern capability i. Each constraint c provides the details as outlined in the previous section.

We evaluate requirements in descending order of weight within each requirement list R. If no capability fulfills the top requirement in R, we remove that requirement and evaluates the next highest. Once we have identified a requirement that can be fulfilled by at least one service, we drop all other less important constraints (i.e., those with lower weight) on the same capability. Ultimately, each requirement list $R \in RL$ contains only a single requirement for each capability. The set of top requirements in RL become the set of constraints in the subsequent requirements cluster analysis.

Algorithm 6 Gracefully Degrading Matching Algorithm $\mathcal{GDM}(C, RL)$.

```
function MatchCandidates($C, RL$)
    /* Match candidate components against capability constraints. */
    for all $R_n in RL$ do
        $sortDescending(R_n)$
    end for
    $sortDecending(RL)$
    /* Lists of constraint and constraints within these lists are sorted descending. */
    for all $R_n \in RL$ do
        for all $c \in R_n$ do
            $maxUtil_n \leftarrow 0$
            /* Collect all utility values in the utility matrix $\mathcal{U}$ */
            $\mathcal{U} \leftarrow \emptyset$
            for all $s \in S$ do
                /* Calculate for each component the utility function as specified in the constraint. */
                $util_s = calcUtil(c, s)$
                $\mathcal{U} \leftarrow util_s$
                if $util_s > maxUtil_n$ then
                    $(maxUtil_n \leftarrow util_s)$
                end if
            end for
            if $maxUtil_n \leq 0$ then
                /* No component $s$ could satisfy the constraint $c$. */
                $removeFromList(R_n, c)$
            else
                /* At least one component $s$ could satisfy the constraint $c$, thus neglect lower weighted constraints on the same capability.*/
                $clearList(R_n)$
                $addToList(R_n, c)$
                $rankCandidates(\mathcal{U}, SC)$
            end if
        end for
    end for
    /* $RL$ contains now only one constraint element in each list. */
    return $RL$
end function
```

Symbol	Meaning
$c_i \in C$	constraint i belonging to set of constraints C.
n	number of constraints $n = \|C\|$.
$s_j \in S$	service j belonging to set of services S.
m	number of services $m = \|S\|$.
$u_{ij} \in \mathcal{U}$	service capability utility value for constraint i and service j.
$r_j \in \mathcal{R}$	normalized preliminary rank r_j for service s_j with $\sum_j r_j = 1$.
$f_i \in \mathcal{F}$	normalized constraint fulfillment degree f_i for constraint c_i with $\sum_i f_i = 1$.
$w_i \in \mathcal{W}$	normalized constraint weight w_i for constraint c_i with $\sum_i w_i = 1$.
$\tau_i \in \mathcal{T}$	normalized importance factor, aggregating f_i and w_i with $\sum_i \tau_i = 1$.
$H(s)$	Service utility entropy calculated on utility values for service s across all n constraints.
$H(c)$	Constraint utility entropy calculated on utility values for constraint c across all m services.
t_s, t_c	Thresholds for $H(s)$ and $H(c)$.
$\delta_s, \alpha_s, \delta_c, \alpha_c$	Configuration parameters for the threshold model calculating t_s and t_c.

Table 6.1: Symbols applied in requirements clustering.

6.5.2 Requirements Cluster Analysis

When we rank all services from a particular category, we implicitly assume they are able to fulfill all top-weighted constraints. However, the more specialized the constraints become, the less likely a single service exhibits all required capabilities to sufficient degree. Two or more services often compensate for their individual shortcomings. We extend the service matching and ranking approach to provide the optimum set of services matching the required constraints.

Consider following simple scenario comprising six constraints $c_1 \dots c_6$ compared to the capabilities of five services $s_1 \dots s_5$. Table 6.2 displays utility results for matching of each constraint and capability. In the extreme Case 1 services s_1 to s_3 each match two constraints completely ($x = 100$) and fail to match the remaining constraints ($x = 0$). Services s_4 and s_5 exhibit mediocre, respectively bad matching results across all constraints. In Case 2, all services match the constraints to some extent. Ranking the complete set of services will not yield practical results in Case 1. Splitting the services into multiple groups, and ranking them separately, will provide more useful service recommendations.

The following process discovers whether services belong to different categories, and if so, defines service membership in these categories. In short, we introduce the service utility entropy and constraint utility entropy to detect potential existence of clusters. Subsequently we cluster related constraints and execute rankings for each cluster separately. Ultimately, we recommend the top service(s) from each cluster.

The service utility entropy applies Shannon's entropy definition (Shannon 1948) on the

	S_1	S_2	S_3	S_4	S_5	f_c		S_1	S_2	S_3	S_4	S_5	f_c
1							**2**						
C_1	100	0	0	40	10	0.156		100	80	70	20	50	0.191
C_2	100	0	0	60	10	0.177		80	20	80	30	50	0.155
C_3	0	100	0	40	10	0.156		30	100	0	50	45	0.134
C_4	0	100	0	60	10	0.177		80	20	30	50	45	0.134
C_5	0	0	100	40	10	0.156		70	100	80	20	80	0.208
C_6	0	0	100	60	10	0.177		80	20	50	70	80	0.179
r_s	0.208	0.208	0.208	0.313	0.063			0.262	0.202	0.185	0.143	0.208	
3							**4**						
C_1	80	80	90	5	10	0.342		80	10	30	0	10	0.168
C_2	90	80	90	0	10	0.348		90	10	30	0	10	0.181
C_3	30	20	30	0	10	0.116		30	80	10	0	10	0.168
C_4	25	25	30	0	10	0.116		25	80	0	0	10	0.148
C_5	5	10	0	0	10	0.032		5	20	90	0	10	0.161
C_6	5	10	10	0	10	0.045		5	25	90	5	10	0.174
r_s	0.303	0.290	0.323	0.007	0.077			0.303	0.290	0.323	0.007	0.077	

Table 6.2: Constraint c_i to service s_j capability match (Utility matrix $\mathcal{U}$) including unweighted, preliminary service rank r and constraint fulfillment degree f_c. In all four cases, constraints are equally important ($w_i = 1/6 \; \forall \; i = 1 \rightarrow 6$).

results of the basic matching process. The service utility entropy $H(s)$ is defined as:

$$H(s) = -\sum_{i=1}^{n} \frac{u(s)_i}{\sum u(s)} * log(\frac{u(s)_i}{\sum u(s)}) \tag{6.1}$$

where $u(s)_i$ is the function for deriving the utility of service s for constraint i. Table 6.3 lists the service rank entropy for our scenario. Maximum entropy for n constraints is $log(n)$.

We introduce a threshold value t to decide when to engage in cluster analysis. We assume existence of two or more clusters if the arithmetic mean of all service utility entropy values drops below t_s, with $0 \leq t_s \leq log(n)$. Lower values of t_s require more distinct services. Extreme cases include $t_s = 0$, where services need to match exactly one constraint and none else, and $t_s = log(n)$ where services exhibiting only minor differences in their utility values are considered belonging to different categories.

The entropy average is a necessary but not a sufficient measurement to determine the presence of different categories. The metric highlights merely unequal distribution of constraint satisfaction. It cannot distinguish in-between services matching the same or different conditions. Case 3 and Case 4 in Table 6.2, for example, yield the same service entropy value for the same service.

We approach this shortcoming by calculating the constraint utility entropy $H(c)$. The definition of $H(c)$ is the same as for $H(s)$ but calculating across constraints instead of services. Constraint utility entropy values will be high when services exhibit high overlap

$H(s)$	s_1	s_2	s_3	s_4	s_5	
Case 1	0.693	0.693	0.693	1.792	1.748	
Case 2	1.742	1.560	1.555	1.687	1.758	
Case 3	1.399	1.471	1.373	0	1.792	
Case 4	1.399	1.471	1.373	0	1.792	
$H(c)$	s_1	s_2	s_3	s_4	s_5	s_6
Case 1	0.803	0.846	0.803	0.846	0.803	0.846
Case 2	1.506	1.489	1.285	1.508	1.518	1.524
Case 3	1.289	1.215	1.311	1.322	1.055	1.352
Case 4	1.032	0.991	1.032	0.797	0.861	1.020

Table 6.3: Service utility entropy $H(s)$, $(maxH(s) = 1.792)$ and constraint utility entropy $H(c)$, $(maxH(c) = 1.609)$ for unbiased utility values $\mathcal{U}$.

of matching constraints (see Table 6.3 Case 2 and Case 3). Whereas $H(c)$ will be low when services yield little overlap of matching constraints (see Table 6.3 Case 1 and Case 4).

Analog to t_s, we derive the arithmetic mean of $H(c)$ and define a threshold t_c, such that $0 \leq t_c \leq log(m)$, with m the number of services. Similar to $H(s)$, the arithmetic mean of $H(c)$ is required but not sufficient to determine the presence of multiple categories.

6.5.2.1 Cluster Threshold Model

We define a joint criteria on $H(s)$ and $H(c)$ to decide when to trigger cluster analysis. We require the arithmetic mean of $H(s) < t_s$ and the arithmetic mean of $H(c) < t_c$. The combined condition requires both entropy measurements to remain below the corresponding thresholds:

$$cluster\ if \qquad \frac{\sum_{i=1}^{m} H(s_i)}{m} < t_s \qquad AND \qquad \frac{\sum_{j=1}^{n} H(c_j)}{n} < t_c \tag{6.2}$$

The threshold model enables selection of appropriate entropy threshold values t_s and t_c. It works with three parameters: n (m), δ_s (δ_c), and α_s (α_c). The number of utility values included in calculating the entropy (n, respectively m) has significant impact on the entropy value and thus also on any threshold. Suppose a service exhibiting utility values $[100, 100, 30, 30]$ yielding entropy 1.233. In this example, we consider such a configuration as qualifying for cluster analysis and set the respective threshold to 0.9 of maximum entropy ($log(4) = 1.386$). We then increase the constraint set to eight. We assume the service exhibits the same distribution of matching and non-matching constraints, i.e., $[100, 100, 30, 30, 100, 100, 30, 30]$. This yields an entropy of 1.926 and amounts to 0.93 of the maximum entropy ($log(8)$). The service would no longer qualify for cluster analysis.

The other factors, δ and α, specify the desired distribution of utility values that indicate potential clusters. Specifically, δ determines how much higher the average of best rated utility values need to be compared to the average of remaining values. To this end, α divides the set of values into top-rated and bottom rated. Both, α and δ are within interval $[0, 1]$. The threshold model mimics a prototype utility distribution. It assumes $\alpha\%$ of elements yielding maximum utility ($x = 100$), and the remaining $1 - \alpha\%$ elements yielding δ utility ($x = 100 * \delta$). The following function returns the threshold for any given n, α, and δ. The calculation left of the + sign derives the entropy for the top $n * \alpha$ elements, while the right side calculates the entropy for the remaining $n * (1 - \alpha)$ elements. The right most factor normalizes the total entropy to the interval $[0, 1]$:

$$t = -\left(n * \alpha * \frac{1}{xsum} * log(\frac{1}{xsum}) + n * (1 - \alpha) * \frac{\delta}{xsum} * log(\frac{\delta}{xsum})\right) * log(n)^{-1} \quad (6.3)$$

where $xsum = n * \alpha + n * (1 - \alpha) * \delta$, the sum of all prototype utility values. An example: $n = 5$, $\alpha = 0.2$ and $\delta = 0.4$ selects the single best element (20% of 5), assumes its utility to be 100, and expects the average of the remaining utility values to equal 40. This configuration equals to calculating the entropy for the utility set $[100, 40, 40, 40, 40]$. The corresponding relative entropy value (here 0.94) becomes the threshold. The entropy for a utility set with equal bottom average ($\overline{u_{bottom}} = (1 - \alpha) * 40$) provides an upper bound. Any other set having the same average bottom partition (e.g., $[100, 50, 50, 30, 30]$) yields entropy values below the threshold.

Figure 6.5 displays various combinations for α and δ for $n = 2 \rightarrow 20$. Configuration of the threshold model focuses on selecting α and δ as n derives automatically from the number of constraints. Parameter selection becomes a tradeoff between tolerating false negatives—potentially missing less distinct categories—and accepting false positives.

Figure 6.5 highlights the trend of any threshold configuration approaching the maximum entropy as n grows. It becomes increasingly hard to distinguish between a service set with and one without potential categories. To mitigate this shortcoming, we penalize services that match hardly any constraints as they distort the average of entropy values. We also penalize little supported and less important constraints. This bias applies solely for deriving the threshold values t_s and t_c. The clustering process utilizes the unbiased utility matrix ($\mathcal{U}$).

Specifically, we multiply the service utility table with the relative preliminary service rank, constraint fulfillment degree and constraint weight. When calculating the entropy values on the biased matrix $\mathcal{U}_b$, barely matching services yield little impact on the entropy average. Well matching services will exhibit high impact and raise the threshold for cluster analysis. There is no need to search for clusters when there are sufficiently many well matching services. In addition, more important constraints (i.e., high constraint weight w_c) will yield high impact. Constraints that are hardly supported (i.e., low fulfillment degree f_c) will exhibit little impact as they should not trigger cluster analysis. The biased utility matrix $\mathcal{U}_b$ is defined as:

$$\mathcal{U}_b = \mathcal{U} \times \mathcal{R} \times \mathcal{T} \quad (6.4)$$

where vector $\mathcal{R}$ contains the normalized preliminary rank r_i for each service s_i with $\sum_{i=1,...,n} r_i = 1$; the importance vector $\mathcal{T}$ aggregates normalized constraint weights w_j

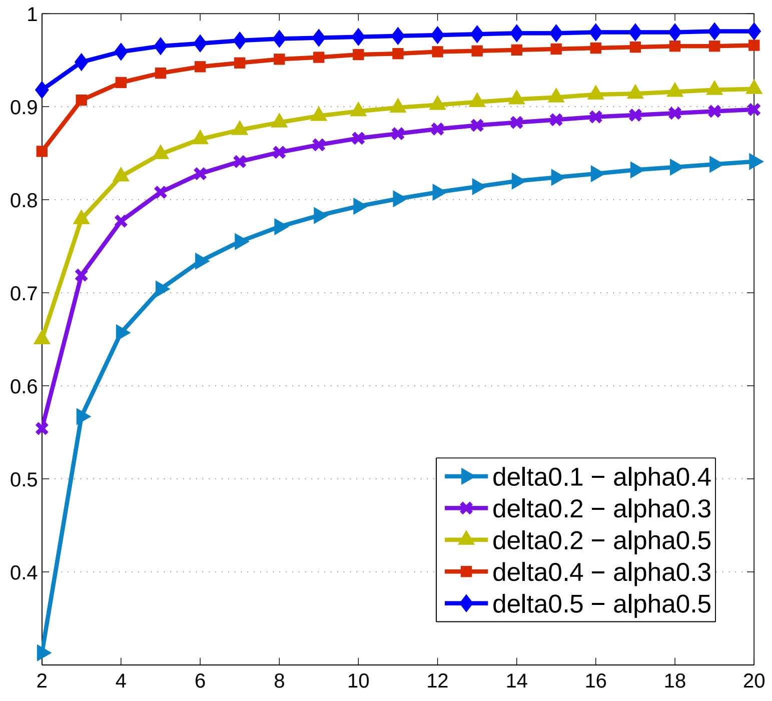


Figure 6.5: Clustering threshold for different combinations of α_s and δ_s with $n = 2 \rightarrow 20$.

	Case 1	Case 2	Case 3	Case 4	α	δ	t
$Mean_a(H(s))$	0.627	0.912	0.517	0.674	0.3	0.2	0.828
$Mean_a(H(c))$	0.467	0.887	0.713	0.498	0.3	0.1	0.653

Table 6.4: Arithmetic mean for service utility entropy $H(s)$, and constraint utility entropy $H(c)$ for biased utility values $\mathcal{U}_b$.

$(\sum_{j=1,\ldots,m} |w_j| = 1)$ and normalized constraint fulfillment degree f_j with $\sum_{j=1,\ldots,m} f_j = 1$ such that:

$$\tau_i = \frac{f_i * w_i}{\sum_n f_j * w_j} \tag{6.5}$$

Table 6.4 lists the arithmetic mean of $H(c)$ and $H(s)$ for all four scenario cases, maximal entropy values, and respective thresholds.

6.5.3 Introduction to Fuzzy C-Means Clustering

Clustering algorithms distribute data elements into a set of meaningful partitions. They fall into two main categories: assigning each data element to exactly one particular cluster

Symbol	Meaning
$x_{i=1\ldots n} \in \mathcal{X}$	data elements to be clustered. Here, x_i is the set of service utility values u for constraint i.
d	dimension of the data elements.
z	number of clusters to distribute the data set across.
ϖ	fuzzy factor determining crisp or fuzzy cluster boundaries.
ε	convergence limit.
$maxIt$	maximum number of iterations when convergence is not achieved.
$k_{j=1\ldots z} \in \mathcal{K}$	clusters centers (i.e., centroid) of same dimension d as the data elements $\mathcal{X}$.
$\mathcal{M}$	membership table, with μ_{ij} defining the membership degree for data element i for cluster j.
$\Vert \bullet \Vert$	distance function, measures distance between any two data elements, any two clusters centers, or between any element and any cluster center.
$v_b(X), v(X)$	(biased) variance in the data elements.
cmp_b, cmp	(biased) cluster compactness measure.
sep_b, sep	(biased) cluster separation measure.
$q(\beta)$	clustering quality function applying *cmp* and *sep* with preference parameter β, with $0 \leq \beta \leq 1$ and $0 \leq q \leq 1$.
u_{ijk}	Final utility for service j respective to constraint i within cluster k.

Table 6.5: Symbols applied in Fuzzy C-Means clustering.

(hard clustering), or assigning data elements to multiple clusters (soft clustering). We focus on the latter category of fuzzy clustering algorithms for grouping constraints according to implicit service categories.

Fuzzy C-Means (FCM) (Bezdek 1981) associates each data element x_i to every cluster $k_{j=1\ldots z}$. The membership table $\mathcal{M}_{ij}$ describes the degree of data element x_i belonging to a particular cluster k_j, such that $\sum_{j=1\ldots z} \mu_{ij} = 1$. Elements close to the cluster center yield higher membership values for that particular cluster than elements farther away. Table 6.5 lists symbols and meaning involved in FCM clustering.

Consider an example data set comprising two-dimensional elements $x_{1\ldots 13}$ displayed in Figure 6.6 (a). The algorithm's objective is minimizing the overall distance of elements to the cluster centers. This *within-class least squared-error function* is defined as:

$$J_\varpi = \sum_{i=1}^{n} \sum_{j=1}^{z} \mu_{ij}^{\varpi} * \Vert x_i, k_j \Vert^2 \tag{6.6}$$

where $\varpi > 1$ is the fuzzy factor [1] and $\Vert \bullet \Vert$ is a distance measurement between data element x and the cluster center k. FCM iteratively recalculates cluster centers and mem-

[1] In clustering literature the fuzzy factor is denoted as m. We apply ϖ because in this thesis m represents the number of services $|S|$.

bership degree until the objective function converges $|J^t_\varpi - J^{t-1}_\varpi| < \varepsilon$ (where ε denotes the convergence limit) or until the maximum number of iterations $maxIt$ is reached.

For our purpose, the distance function is the euclidian distance, defined as:

$$distance = \left(\sum_{i=1}^{d} |x_i - y_i|^2 \right)^{1/2} \tag{6.7}$$

with d the dimensions of the data elements x and y (in our example $d = 2$).

The cluster center (i.e., the centroid) is the means of all elements weighted by their membership degree. Elements further way—thus having lower membership degree—yield lower impact on the center than closer elements. The centroid is defined as:

$$k_j = \frac{\sum_{i=1}^{n} \mu_{ij}^{\varpi} * x_i}{\sum_{i=1}^{n} \mu_{ij}^{\varpi}} \tag{6.8}$$

The membership of an element x belonging to a particular cluster k depends on the ratio of distance between x and k and the distance from x to all centroids $\mathcal{K}$:

$$\mu_{ij} = \left(\sum_{l=1}^{z} \left(\frac{\|x_i, k_j\|}{\|x_i, k_l\|} \right)^{2/(\varpi-1)} \right)^{-1} \tag{6.9}$$

Specifically, FCM applies the fuzzy factor ϖ to define the crispness of membership degree. In general, high values of ϖ implicate very fuzzy cluster boundaries whereas low values result in clear cluster limits. For ϖ close to 1, FCM replicates the behavior of K-Means clustering (Macqueen 1967). With $\varpi = 2$, distance measurements are normalized linearly and for $\varpi \rightarrow \infty$, elements will belong to every cluster with equal degree. The basic FCM process applies the steps in Algorithm 7 to determine cluster membership degree.

In Figure 6.6 we cluster elements into two, three, or four clusters ($z = 2 \ldots 4$) for fuzzy factor $\varpi = 3$ (b) and $\varpi = 1.2$ (c)(d)(e). In subfigure (b) the top pie chart comprises the average membership of elements $1 \ldots 4$, the right, middle, and lower left pie charts describe elements $5 \ldots 8$, $9 \ldots 12$, and 13, respectively. For each pie chart, the inner most circle visualizes fuzzy membership to two clusters, while the middle and outer most circles describe membership for three, respectively four, clusters. For $\varpi = 1.2$, cluster membership becomes binary as elements belong completely to one cluster. Subfigures (c), (d), and (e) demonstrate how clusters break into smaller segments as we raise the value of z from 2 to 4.

FCM exhibits some idiosyncrasies. Changes in initial membership randomizations may yield different clustering results. Consequently, we derive data for any figure or table from multiple iterations of the fuzzy clustering process.

The overall quality of the clustering process depends on appropriately selecting the configuration parameters $(z, \varpi, \varepsilon, maxIt)$. The number of maximum $maxIt$ iterations and

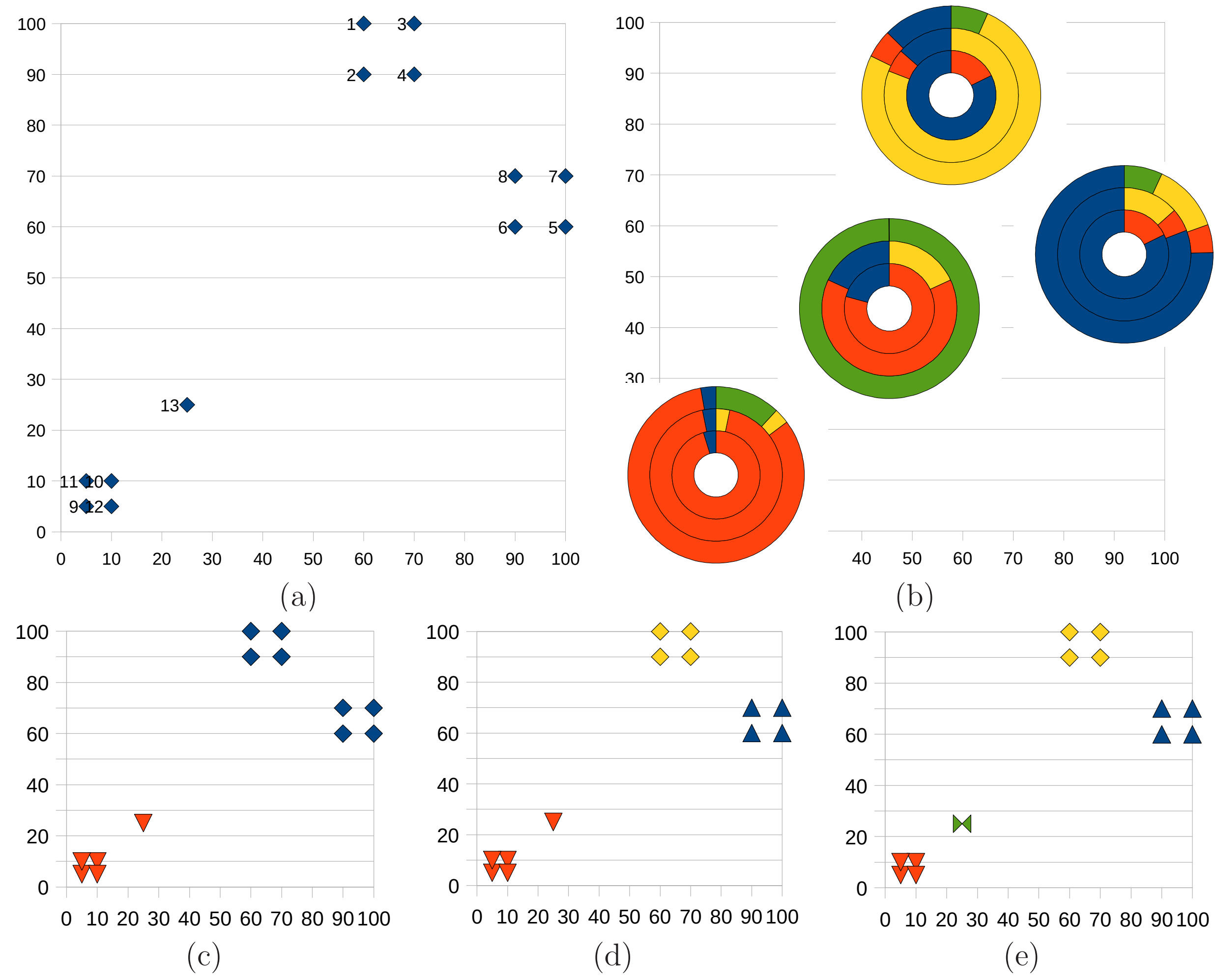


Figure 6.6: FCM clustering result on data set (a) for two, three, and four clusters with fuzzy factor $\varpi = 3$ (b) and $\varpi = 1.2$ (c)(d)(e). Same colors and same icons represent mutual cluster membership.

convergence limit ε define the termination condition for any given selection of z and ϖ. The amount of data and available time restrict the applicable values but other considerations are not necessary. (Liu, Li, and Li 2008) suggest $\varepsilon = 0.001$. The number of clusters that optimally describe the data, and the optimal fuzzyness of the cluster boundaries are more sensitive choices.

We can select the number of iterations arbitrarily high, respectively ε arbitrarily low; we will not obtain sensible clustering results when choosing z inappropriately. When grouping the elements in Figure 6.6 (a) into two clusters (b, innermost circle) or (c), we cannot detect the difference between elements $1 \ldots 4$ and $5 \ldots 8$.

A rule of thumb (McBratney and De Gruijter 1992) recommends selecting $max_z \approx n^{1/2}$ with n the number of elements. A computationally more intensive approach calculates the clustering quality for increasing number of clusters until reaching maximum quality.

Algorithm 7 Basic Fuzzy C-Means Clustering Algorithm $\mathcal{FCM}(X, z, \varpi, maxIt, \varepsilon)$.

```
function PERFORMCLUSTERING($X, z, \varpi, maxIt, \varepsilon$)
    $\mathcal{M} \leftarrow initRandomMembership(X, z)$
    $lastJ \leftarrow 0$
    for $round = 0; round < maxIt; round++$ do
        $K \leftarrow calculateClusterCenters(\mathcal{M}, \varpi)$
        $updateClusterMembership(X, K, \varpi)$
        $J = calculateObjectiveFct(X, K, \varpi)$
        if $|J - lastJ| < \varepsilon$ then
            $break$
        else
            $lastJ \leftarrow J$
        end if
    end for
    return $\mathcal{M}$
end function
```

(He, Tan, Tan, and Sung 2003) propose a combination of cluster compactness and cluster separation for crisp clustering as a viable overall quality measure. Compactness describes how well the clusters explain the variance in the data. The variance v of a set of vectors (here constraints) is defined as:

$$v(X) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|x_i, \bar{x}\|^2} \tag{6.10}$$

where $\|x_i, \bar{x}\|$ computes the distance of x_i to the mean $\bar{x}$ of all elements in X, with $\bar{x} = \frac{1}{n}\sum_i x_i$. The less dispersed the elements, the smaller the variance. Next, we compare the variance found in each cluster to the overall variance. We alter the definition of compactness *cmp* to consider fuzzyness:

$$cmp = \frac{1}{z}\sum_{j=1}^{z} \sqrt{\frac{\sum_i \mu_{ij} * \|x_i, k_j\|^2}{\sum_i \mu_{ij}}} * v(X)^{-1} \tag{6.11}$$

where $\sqrt{\frac{\bullet}{\bullet}}$ calculates the variance of elements weighted by their degree of membership in cluster k_j. Compactness yields 0 for one cluster. With increasing clusters, the compactness eventually increases to 1 at which point each data element resides in a separate cluster. We prefer higher compactness over lower compactness, but we need to avoid introducing too many clusters. To this end, we reuse the cluster separation metric by (He, Tan, Tan, and Sung 2003).

Cluster separation describes the heterogeneity between clusters. Clusters further apart exhibit more distinct elements than clusters close together. Separation is the coefficient of total pairwise distance between cluster centers and maximum possible distance. Separation reaches its maximum ($sep = 1$) when each cluster contains exactly one element. When one cluster comprises all elements, separation is zero. Again, we include the membership table $\mathcal{M}$ in the definition:

$$sep = \frac{\sum_{j=1}^{z-1} \sum_{l=j+1}^{k} \|k_j, k_l\|^2}{\sum_{i=1}^{n-1} \sum_{p=i+1}^{n} \|x_i, x_p\|^2} \tag{6.12}$$

The sum of pairwise distance between all elements yields computational complexity $\mathcal{O}(n^2)$. However, the distance remains unchanged for all iterations of cluster counts $z = 1 \ldots n$ and thus needs computation only once.

The combined metrics identify the maximum clustering quality. For one cluster, compactness equals 1 and separation equals 0. For all elements in individual clusters, compactness yields 0 and separation 1. The quality function $q(\beta)$ identifies the number of clusters that best describe the underlying distribution:

$$q(\beta) = 1 - (\beta * cmp + (1 - \beta) * sep) \tag{6.13}$$

where β defines a preference on compactness or separation. A β value below 0.5 assigns more weight on distinct clusters (sep) than on (lower) intra-cluster variance (cmp), and vice versa. The maximum quality value describes the best number of clusters k.

As outlined above, choosing ϖ too high yields inconclusive examples. Setting ϖ too low results in FCM assigning elements equidistant to two cluster centers arbitrarily to one of them. For increasingly low values, the quality metric yields the highest values for additional clusters, one of them containing only 13.

This concludes the introduction to fuzzy c-means. We discuss constraint-specific automatic selection of best values for ϖ and z in the following subsection.

6.5.4 Biased Clustering Algorithm

Numerous papers improve the fuzzy c-means algorithm to achieve robustness (Chintalapudi and Kam 1998, Zhang and Leung 2004, Leski 2003). These techniques apply data distribution intrinsic metrics to identify and mitigate the effect of outliers and noise. We focus on achieving optimum clusters where significant services and constraints should influence the result more than insignificant services or constraints. The basic FCM algorithm considers all data elements of equal importance; this is where we introduce our biased clustering algorithm.

Let us interpret the example visualized in Figure 6.6 as two services matching 13 constraints. Service s_1 matches well constraints $c_1 \rightarrow c_4$ and s_2 primarily matches constraints $c5 \rightarrow c_8$. Constraints $c_9 \rightarrow c_{13}$ are hardly supported by either s_1 or s_2.

A pure visual analysis of Figure 6.6 identifies two, three, or four clusters as sensible constraint partitions. For selecting the best number of service categories, we need to focus on clusters that contain (a) services which tend to fulfill complementary constraints well

	s_1	s_2	f	$H(c)$	w_c	$\mu(K_{1a})$	$\mu(K_{2a})$	$\mu(K_{1b})$	$\mu(K_{2b})$
c_1	60	100	0.115	0.662	0.12	1	0	0.857	0.125
c_2	60	90	0.108	0.673	0.12	1	0	0.848	0.152
c_3	70	100	0.122	0.677	0.12	1	0	0.870	0.130
c_4	70	90	0.115	0.685	0.12	1	0	0.832	0.168
c_5	100	60	0.115	0.662	0.12	0	1	0.125	0.875
c_6	90	60	0.108	0.673	0.12	0	1	0.152	0.848
c_7	100	70	0.122	0.677	0.12	0	1	0.130	0.870
c_8	90	70	0.115	0.685	0.12	0	1	0.168	0.832
c_9	5	5	0.007	0.693	0.008	0.501	0.499	0.500	0.500
c_{10}	10	10	0.014	0.693	0.008	0.501	0.499	0.500	0.500
c_{11}	5	10	0.011	0.637	0.008	0.536	0.464	0.503	0.497
c_{12}	10	5	0.011	0.637	0.008	0.466	0.534	0.497	0.503
c_{13}	25	25	0.036	0.693	0.008	0.501	0.499	0.500	0.500
r	0.5	0.5	$wAvgH(c)$	0.673	K_{S1}	62.770	91.317	65.450	91.491
$H(s)$	2.122	2.122	$wAvgH(s)$	2.122	K_{S2}	91.282	62.791	95.491	65.450

Table 6.6: Constraints, weights, utility, and fulfillment for Case 5. For $z = 2$, $\mu(K_{1a})$ and $\mu(K_{2a})$ display membership degree for clustering with $\varpi = 1.2$; $\mu(K_{1b})$ and $\mu(K_{2b})$ with $\varpi = 3$.

and (b) services that tend to satisfy important constraints. On the one hand, we need to avoid partitioning according to different levels of constraint significance. On the other hand, we need to avoid clustering according to overall constraint fulfillment which results in partitions of low, medium, and highly satisfied constraints.

First, we ensure that services determine the clustering result proportional to their utility. To this end, we transform the utility values ($\mathcal{X}$) before clustering to reflect the preliminary service rank. We multiply each x_{ij} with the service rank r_j and renormalize the matrix such that services with average utility $x_i = \bar{x}$ maintain their utility value ($\mathcal{X} \cdot \mathcal{R} \cdot |\mathcal{R}|$). We thereby exploit the FCM's sensitivity towards outliers. After the transformation better ranked services exhibit higher utility values compared to lower ranked services and thus yield more impact during the subsequent clustering process.

Second, we make sure that the cluster result comprises only the important and well supported constraints. Specifically, we integrate constraint weights and fulfillment into the clustering process. For the sake of argument, suppose the constraint weights w_c in Table 6.6 for the data underlying Figure 6.6.

The basic FCM algorithm is ignorant of constraints. For $\varpi = 3$, the quality measure recommends three clusters with significant crispness ($max(\mu_{ij}) > 0.75\ \forall\ i = 1 \ldots 12$) $c_1 \ldots c_4$, $c_5 \ldots c_8$, and $c_9 \ldots c_{12}$, while putting c_{13} in the middle. For lower ϖ values, the same partitioning persists with exception to c_{13} establishing a separate cluster. Considering constraint weight w_c: c_1 to c_8 yield high significance, and the remaining constraints yield low significance. In this case, a better result yields only two clusters $c_1 \ldots c_4$ and $c_5 \ldots c_8$, without any preference for the the exact allocation of the less significant constraints $c_9 \dots c_{13}$.

We adapt the clustering algorithm to drop the condition that $\sum \mu_i = 1$. We bias the membership according to the importance vector $\mathcal{T}$. After multiplying the membership table $\mathcal{M}$ with the importance vector, less significant elements yield little impact when calculating the cluster center. Subsequent evaluation of membership degree resets $\sum \mu_i = 1$, hence the need to bias the membership table in every iteration. The importance bias also effects calculation of the mean vector $\bar{x}$, variance $v(X)$, total distance measurement, and separation.

The biased vector centroid $\bar{x_b}$ for importance vector $\mathcal{T}$ and fuzzy factor ϖ is defined as:

$$\bar{x_b} = \sum_i \frac{x_i * \tau_i^{\varpi}}{\sum_i \tau_i^{\varpi}} \tag{6.14}$$

The biased variance v_b of a set of constraints and importance vector $\mathcal{T}$ is defined as:

$$v_b(X) = \sqrt{\sum_{i=1}^{n} (\|x_i, \bar{x_b}\|^2 * \tau_i^2) * \left(\sum_i \tau_i^2 \right)^{-1}} \tag{6.15}$$

We update the function for calculating the total distance between constraints accordingly. Distance between important constraints gains significance, while distance between less important or mixed important elements has little effect on the overall distance.

$$dist_X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \|x_i, x_j\|^2 * \frac{\tau_i + \tau_j}{2} \tag{6.16}$$

For cluster separation, we have to adapt the distance measurement between clusters. For each cluster we compute the importance of the contained elements and apply the same biased distance function as introduced above.

$$sep_b = \sum_{j=1}^{z-1} \sum_{p=j+1}^{z} \left(\|k_i, k_j\|^2 * \frac{\sum_i(\tau_i * \mu_{ij}) + \sum_i(\tau_i * \mu_{ip})}{2} \right) * dist_X^{-1} \tag{6.17}$$

where $\sum_i \tau_i * \mu_{ij}$ defines the importance of cluster j.

In the last step, we normalize the membership degree. Algorithm 8 elaborates the differences to the basic FCM algorithm. The functions *calculateClusterCenters*, *update-ClusterMembership*, and *calculateObjectiveFct* remain unchanged.

We observe two phenomena when applying the importance vector $\mathcal{T}$. First, with increasing z, clusters comprising the most significant elements rapidly split up into separate, roughly equal clusters. Clusters of less important elements form rather late (i.e., for high numbers of z, close to $|\mathcal{X}|$) if ϖ is low and do not form at all for $\varpi > 3$.

Figure 6.7 compares cluster entropy H_k for biased (a) and unbiased (b) clustering, with $\varpi = 2$. Cluster entropy measures for each element the membership degree distribution

Algorithm 8 Biased FCM Clustering Algorithm $\mathcal{BFCM}(\mathcal{X}, z, \varpi, \varepsilon, maxIt, \mathcal{T}, \beta)$.

```
function PerformBiasedClustering($\mathcal{X}, z, \varpi, \varepsilon, maxIt, \mathcal{T}, \beta$)
    $\mathcal{M} \leftarrow initRandomMembership(X, z)$
    /* Bias membership according to importance. */
    $\mathcal{M} \leftarrow \mathcal{M} * \mathcal{T}$
    $lastJ \leftarrow 0$
    $maxDist \leftarrow calculateTotalDistance(\mathcal{X}, \mathcal{T})$
    for $round = 0; round < maxIt; round++$ do
        $K \leftarrow calculateClusterCenters(z, \varpi)$
        $updateClusterMembership(\mathcal{X}, K, \varpi)$
        /* Recalculating the cluster membership resets $\sum \mu_i = 1$, $\rightarrow$ bias membership again according to importance. */
        $\mathcal{M} \leftarrow \mathcal{M} * \mathcal{T}$
        $J = calculateObjectiveFct(\mathcal{X}, K, \varpi)$
        if $|J - lastJ| < \varepsilon$ then
            break
        else
            $lastJ \leftarrow J$
        end if
    end for
    $calculateQuality(\mathcal{X}, \mathcal{T}, K, \beta, \mathcal{M}, maxDist, \varpi)$
    $normalizeMembership(\mathcal{M})$
    return membership
end function

function calculateTotalDistance($\mathcal{X}, \mathcal{T}$)
    for $i = 0; i < |X| - 1; i++$ do
        for $j = i + 1; j < |X|; j++$ do
            $total \leftarrow total + calcDistance(x_i, x_j)^2 * (\tau_i + \tau_j)/2$
        end for
    end for
    return total
end function

function normalizeMembership($\mathcal{M}$)
    /* Recalculate membership such that $\sum \mu_i = 1$. */
    for all constraint $i \in C$ do
        $sum_\mu \leftarrow 0$
        for all cluster $k \in K$ do
            $sum_\mu \leftarrow sum_\mu + \mu_{ik}$
        end for
        for all cluster $k \in K$ do
            $\mu_{ik} \leftarrow \mu_{ik}/sum_\mu$
        end for
    end for
end function
```

Algorithm 9 Continuing $\mathcal{BFCM}(\mathcal{X}, K, \varpi, \varepsilon, maxIt, \mathcal{T})$.

```
function CALCULATEQUALITY($\mathcal{X}, \mathcal{T}, K, \beta, \mathcal{M}, maxDist, \varpi$)
    /* 1. Calculate compactness. */
    $cmp \leftarrow 0;\ var \leftarrow 0;\ totalBias \leftarrow 0;\ \bar{x_b} \leftarrow \emptyset$
    /* 1.1 Calculate biased center. */
    for $i = 0; i < |X|; i++$ do
        $bias \leftarrow s_i^{\varpi}$
        for $j = 0; j < dimensions(X); j++$ do
            $\bar{x_{bj}} \leftarrow \bar{x_{bj}} + x_{ij} * bias$
        end for
        $totalBias \leftarrow totalBias + bias$
    end for
    $\bar{x_{bj}} \leftarrow \bar{x_{bj}}/totalBias;\ div \leftarrow 0$
    /* 1.2 Calculate biased maximum variance. */
    for $i = 0; i < |X|; i++$ do
        $var \leftarrow var + calcDistance(x_i, \bar{x_b})^2 * \tau_i^2$
        $div \leftarrow div + \tau_i^2$
    end for
    $var \leftarrow var^{1/2}/div$
    /* 1.3 Calculate biased intra-cluster variance. */
    $dist \leftarrow 0$
    for all cluster $k \in K$ do
        $sum_\mu \leftarrow 0$
        for all constraint $i \in C$ do
            $dist \leftarrow dist + calcDistance(i, k)^2 * \mu_{ik}^2$ /* We need not include importance
vector $s$ as we have multiplied it with membership already before. */
            $sum_\mu \leftarrow sum_\mu + \mu_{ik}^2$
        end for
        $cmp \leftarrow cmp + (dist/sum_\mu)^{1/2}/var$
    end for
    /* 2. Calculate separation. */
    $sep \leftarrow 0;\ clImp \leftarrow \emptyset$
    /* 2.1 Calculate cluster importance. */
    for all cluster $k \in K$ do
        for all constraint $i \in C$ do
            $clImp_k \leftarrow clImp_k + \mu_{ik}$
        end for
    end for
    /* 2.2 Calculate inter-cluster distance. */
    for $i = 0; i < |K| - 1; i++$ do
        for $j = i + 1; j < |K|; j++$ do
            $clDist \leftarrow clDist + calcDistance(k_i, k_j)^2 * (clImp_i + clImp_j)/2$
        end for
    end for
    $sep \leftarrow clDist/maxDist$
     return $1 - (\beta * cmp + (1 - \beta) * sep)$
end function
```

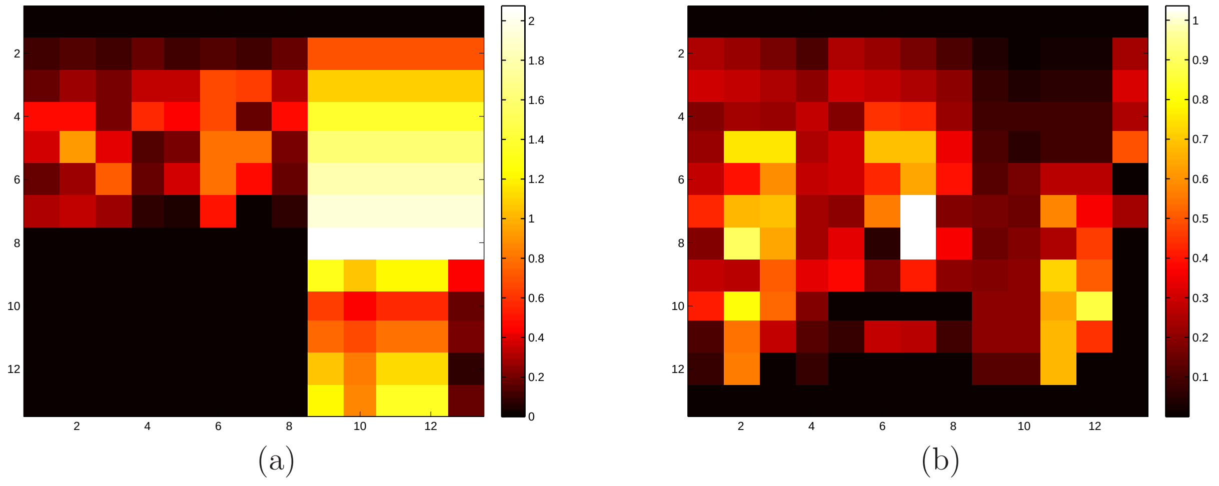


Figure 6.7: Cluster entropy H_k for biased (a) and unbiased (b) clustering.

across all available clusters. Low entropy value (dark colors) indicate focus on one or a few clusters. Bright colors highlight elements that (equally) belong to many clusters. Each column comprises the entropy values for a particular element. The top row contains the entropy values for $z = 1$ clusters, down to the bottom row containing $z = 13$ clusters.

In the biased case, we notice how elements $1 \ldots 8$ break into smaller clusters before populating individual clusters in row 8. The remaining elements equally belong to an increasing number of cluster until after row 8 element 13 separates into a distinct cluster. Interestingly, elements $9 \ldots 12$ never form a cluster themselves.

In the unbiased case, elements $9 \ldots 12$ exhibit a similar behavior as element $1 \ldots 8$ in the unbiased case. Round 2 and 3 yield crisp cluster membership (generally low cluster entropy values). After round 4, element 13 remains in a distinct cluster, the other elements yield shifting membership. In contrast to the biased case, all elements eventually end up in individual clusters.

Second, we notice an early, sharp decline in compactness opposed to a late, steep increase in separation. Compactness is minimal when elements populate individual clusters. As observed above, the most significant elements quickly scatter into separate groups. If insignificant elements eventually occupy their own cluster, they barely reduce compactness.

The same effect determines the late, steep incline of cluster separation. Separation is maximal when each cluster contains a single element. As long as clusters of significant elements split into increasingly smaller clusters, the centroids remain close together, adding little to separation. The distance between centroids grows once less significant elements form individual clusters clearly separated from the existing cluster centers.

Figure 6.8 displays compactness and separation for $\varpi = [1.5, 2, 3]$ with biased and unbiased clustering side by side. We notice biased compactness reaching its minimum once all important elements reside in separate clusters. Unbiased compactness drops similarly fast at the beginning, but then phases out, reaching its minimum at $z = n$. Biased separation remains low until all important elements populate individual clusters, then rising sharply. In contrast, the unbiased separation metric displays near-linear growth.

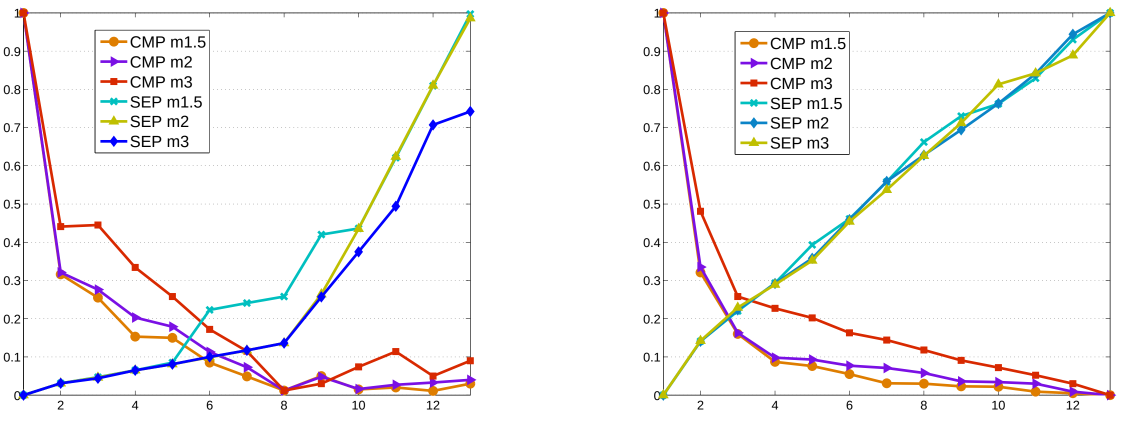


Figure 6.8: Compactness and separation for biased (a) and unbiased (b) clustering.

We exploit the sharp incline in separation, respectively decline in compactness, for selecting the maximum number of clusters z. Specifically, we derive the amount of elements exhibiting a importance value higher than the average of τ minus ϕ (where ϕ is twice the standard deviation σ divided by the number of constraints) such that:

$$z_{max} = |\mathcal{T}_{top}| \qquad \tau_i \in \mathcal{T}_{top} \qquad \forall \tau_i \geq (\bar{\tau} - \frac{2 * \sigma_\tau}{n}) \tag{6.18}$$

The design of ϕ ensures that for a large number of constraints, low-performing elements are still detected.

We subsequently measure the quality q for each additional cluster for $z = 2 \longrightarrow z_{max}$. And select the cluster count that yields the maximum increase in quality. We detail the selection process in Algorithm 10.

Finally, we determine the fuzzy factor ϖ. The entropy-based threshold values indicate the existence of clusters. Hence, we restrict ϖ to the interval $]1, 3]$. For two data sets, the one with lower weighted average service entropy $Mean_a(H(s))$ yields crisper clusters. The ratio of service entropy to entropy threshold determines the exact value for ϖ, specifically:

$$\varpi = 1 + 2 * \frac{Mean_a(H(s))}{t_s} + \gamma \tag{6.19}$$

where q_s is the service entropy threshold, and γ ensures that $\varpi > 1$. We set γ to 0.0001.

Table 6.8 lists the clustering result for case 1 . . . 4. We included case 2 —failing to meet the either threshold criteria— and case 3—exceeding the constraint entropy threshold t_c— for demonstration purpose. All constraint weights w_c are 1/6.

Algorithm 10 Best Cluster Quality Algorithm $\mathcal{CQA}(\mathcal{X}, \mathcal{T}, \mathcal{R})$.

```
function SELECTBESTMEMBERSHIP($\mathcal{X}, \mathcal{T}$)
    $z \leftarrow calcMaxCluster(\mathcal{X}, \mathcal{T})$
    $\varpi \leftarrow calcFuzzyFactor(\mathcal{X}, \mathcal{T})$
    $\mathcal{X}_b \leftarrow \mathcal{X} * \mathcal{R} * \mathcal{T}$
    $\mu_{best} \leftarrow \emptyset$
    $q_{max} \leftarrow 0$
    for $k = 1 \rightarrow z$ do
        /* For each round, initialize empty cluster centers $K$. */
        $K \leftarrow \{k\}$
        $\mu \leftarrow$ call $PerformBiasedClustering(\mathcal{X}_b, K, \varpi, \varepsilon, maxIt, \mathcal{T}, \beta)$
        $q \leftarrow calcQuality(\mu)$
        if $q > q_{max}$ then
            $q_{max} \leftarrow q$
            $\mu_{best} \leftarrow \mu$
        end if
        if $q > q_{max}$ then
            $q_{max} \leftarrow q$
        end if
    end for
    return $\mu_{best}$
end function
```

	$K_{1.1}$	$K_{1.2}$	$K_{1.3}$	$K_{2.1}$	$K_{2.2}$	$K_{2.3}$	$K_{3.1}$	$K_{3.2}$	$K_{4.1}$	$K_{4.2}$	$K_{3.3}$
C_1	**0.882**	0.059	0.059	0.08	**0.849**	0.07	**0.992**	0.008	0.028	0.032	**0.941**
C_2	**0.931**	0.035	0.035	0.032	0.019	**0.949**	**0.993**	0.007	0.019	0.022	**0.959**
C_3	0.059	**0.882**	0.059	0.261	**0.543**	0.196	0.007	**0.993**	0.022	**0.952**	0.026
C_4	0.035	**0.931**	0.035	0.318	0.312	**0.37**	0.005	**0.995**	0.031	**0.932**	0.037
C_5	0.059	0.059	**0.882**	0.312	**0.454**	0.233	0.111	**0.889**	**0.979**	0.01	0.011
C_6	0.035	0.035	**0.931**	**0.922**	0.032	0.046	0.087	**0.913**	**0.984**	0.008	0.008
ϖ	2.516			3.0			2.250		2.629		
z_{max}	3			3			2		4		
H_k	0.335			0.592			0.092		0.18		
$\bar{\mu}$	$0.\dot{3}$	$0.\dot{3}$	$0.\dot{3}$	0.321	0.368	0.311	0.634	0.366	0.33	0.343	0.327

Table 6.7: Biased cluster algorithm configuration (z_{max} and ϖ) and results for case 1 to 4. Bold numbers highlight the top cluster membership degree.

As discussed above, the cluster entropy H_k describes how well constraints fit into their clusters—the more focused on one cluster, the lower the entropy. Low H_k values for case 1,3, and 4 reflect the crisp membership degree. As indicated by the threshold, we do not gain any insight from clustering Case 2. Clustering of constraints in case 3 demonstrates nicely the role of the constraint entropy. The resulting two clusters separate the constraints according to high and low constraint fulfillment degree.

6.5.5 Cluster-specific ranking

Before the clustering, we match a single list of constraints to a set of services, calculating utility values. The clustering process then partitions the set of constraints into multiple groups for individual ranking. In the subsequent ranking phase, we evaluate all services within each cluster. We combine cluster weights w and membership degree μ, thus avoiding defuzzyfication of the clustering result. A constraint belonging equally to two clusters will thus influence the ranking result in both clusters to the same degree. The service utility for a given constraint and cluster u_{ijk} is defined as:

$$u_{ijk} = x_{ij} * w_i * \mu_{ik} \tag{6.20}$$

where μ_{ik} is the membership of constraint i in cluster k; for all constraints $i \in C$, all services $j \in S$, and clusters $k \in K$.

We integrate the basic LSP ranking algorithm (Alg. 2) introduced in Section 4.3. For each cluster, Algorithm 11 updates the constraint weights, calls the LSP algorithm, and finally returns a ranked service list for each cluster.

Evaluation of ranked cluster results is twofold. First we compare ranked clusters results to regular ranking results for case 1 to 4 in Table 6.8. Second, we compute the Pearson product-momentum coefficient on ranks as a measure of correlation.

Algorithm 11 Biased Ranking Algorithm $\mathcal{BA}(\mathcal{X}, \mu, w)$.

```
function PerformClusterspecificRanking($\mathcal{X}, \mu, w$)
    /* Initialize array of rank results. */
    $RR \leftarrow \emptyset$
    for all $Cluster\ k \in K$ do
        /* Initialize biased constraint weights. */
        $biasedW \leftarrow \emptyset$
        for all $Constraint\ i \in C$ do
            $biasedW_i \leftarrow \mu_{ik} * w_i$
        end for
        $RR[k] \leftarrow$ call $LSPRankingAlgorithm(\mathcal{X}, biasedW)$
    end for
    return $RR$
end function
```

6.5.5.1 Measuring Clustering Benefit

We neglect composition costs and focus on comparing the benefit of selecting the top services from each cluster to selecting the top service deriving from the regular ranking process. The clustering benefit J_{best} is defined as the sum of top ranked service utility weighted according to the relative size of cluster k where $\bar{\mu}_k = \sum_i \mu_{ik}$ (listed in Table 6.6):

$$J_{best} = \sum_k (\bar{\mu}_k * max(R_k)) \tag{6.21}$$

where set R_k contains the ranked service utility values in cluster k. The ranking results in Table 6.8 remain non-normalized to preserve the differences in utility values. Also, we do not need normalization as all utility values derive from the same value range ($[0, 100]$).

In case 1, combining S_1, S_2, and S_3 yields a 81.3% benefit increase over selecting S_4. Even a combination of S_4 (the second best choice in any cluster) and any top rated service provides a 29% raise. Clustering in case 2 exhibits only marginal benefits (+6%). Also, service S_1 is ranked best in cluster 1 and 3, placed second in cluster 2. We do not achieve any benefit in case 3, where the ranking order in both clusters equals the order originating from the regular ranking process. Case 4 exhibits benefit from clustering. Combination of S_1, S_2, and S_3 result in 97% better constraint support than selecting S_3 alone. For the 2-tuples of these services we gain 57% (S_1, S_2), 57% (S_1, S_3), and 56% (S_2, S_3) in benefit.

We apply the Pearson's coefficient—introduced in Chapter 4.2—to measure the difference in ranking positions. We expect clusters to exhibit a distinctly different service order. We derive the overall difference between cluster and non-clustered ranks by aggregating the weighted (applying $\bar{\mu}_k$) Pearson's correlation found between each set of cluster rankings and the non-clustered ranking. We utilize the absolute rank positions for calculating the Pearson coefficient ρ. Relative ranking result $R_1[50, 33.33, 33.33, 33.33, 10]$, for example, becomes $[1, 3, 3, 3, 5]$. Table 6.8 includes cluster specific correlation and total correlation

	$K_{1.1}$	$K_{1.2}$	$K_{1.3}$	R_1	$K_{4.1}$	$K_{4.2}$	$K_{4.3}$	R_4
1	$S_1(90.64)$	$S_2(90.64)$	$S_3(90.64)$	$S_4(50.0)$	$S_3(86.46)$	$S_2(77.53)$	$S_1(82.42)$	$S_3(41.67)$
2	$S_4(50.0)$	$S_4(50.0)$	$S_4(50.0)$	$S_1(33.33)$	$S_2(23.68)$	$S_1(28.87)$	$S_3(29.76)$	$S_1(39.17)$
3	$S_5(10.0)$	$S_5(10.0)$	$S_5(10.0)$	$S_2(33.33)$	$S_5(10.0)$	$S_5(10.0)$	$S_2(12.36)$	$S_2(37.5)$
4	$S_3(4.68)$	$S_3(4.68)$	$S_2(4.68)$	$S_3(33.33)$	$S_1(7.35)$	$S_3(6.54)$	$S_5(10.0)$	$S_5(10.0)$
5	$S_2(4.68)$	$S_1(4.68)$	$S_1(4.68)$	$S_5(10.0)$	$S_4(2.39)$	$S_4(0.02)$	$S_4(0.02)$	$S_4(0.83)$
ρ	0.224	0.224	0.238	0.229	0.7	0.3	0.9	0.628
	$K_{2.1}$	$K_{2.2}$	$K_{2.3}$	R_2	$K_{3.1}$	$K_{3.2}$		R_3
1	$S_1(73.15)$	$S_2(78.06)$	$S_1(81.14)$	$S_1(73.33)$	$S_3(81.98)$	$S_3(18.43)$		$S_3(41.67)$
2	$S_5(59.86)$	$S_1(65.48)$	$S_2(63.29)$	$S_5(58.33)$	$S_1(77.5)$	$S_1(17.07)$		$S_1(39.17)$
3	$S_4(52.89)$	$S_5(64.88)$	$S_3(53.97)$	$S_2(56.67)$	$S_2(73.39)$	$S_2(16.81)$		$S_2(37.5)$
4	$S_3(44.74)$	$S_3(57.49)$	$S_5(50.25)$	$S_3(51.67)$	$S_5(10.0)$	$S_5(10.0)$		$S_5(10.0)$
5	$S_2(32.84)$	$S_4(32.25)$	$S_4(32.72)$	$S_4(40.0)$	$S_4(2.26)$	$S_4(0.01)$		$S_4(0.83)$
ρ	0.6	0.7	0.7	0.668	1	1		1

Table 6.8: Clustered Ranking algorithm results for case 1 to 4 compared to unclustered ranking results.

values. As expected, examples with distinct clusters and high clustering benefit (case 1 and 4) exhibit lower correlation than case 2 and 3. We notice the small difference between ρ_{Case2} and ρ_{Case4} when we correlate the overall ranking set. Correlation is unbiased and differences (or lack thereof) at the bottom of the ranking list impact the result to the same extent as differences at the top. Interested in changes amongst the best fitting service, we limit the selection to the top two services. Subsequently correlation of S_1 and S_5 in R_2 yields $\rho = 1$, as in every of the three clusters, S_1 is ranked higher than S_5. The same limitation on R_4 yields $\rho = -0.33$, as S_1 ranks higher than S_3 in two out of three clusters.

6.6 Service Composition Recommendation

The clustering process offers a set of best suited services fulfilling the given requirements. Together, the top services from each cluster provide the most qualified aggregation, but not necessarily the cheapest. Selecting the top members ignores any form of aggregation costs.

In this thesis, we focus only on one form of aggregation costs: service distance. Services yielding close proximity have proven to function well in joint efforts. We, therefore, consider services that have often been utilizes in a common context to be more suitable for aggregation than services that were rarely used together. Specifically, we apply the distance metrics introduced in Chapter 4. Ultimately, we need to find a tradeoff between minimal aggregation costs (i.e., low overall service proximity) and high service utility.

The top cluster elements do not necessarily exhibit low service distance. We, therefore, require an algorithm to test other services aggregations for similarly high utility but considerably lower distance.

Brute-force testing of every possible combination yields unpractical for large sets of services and clusters as the underlying problem is NP hard. Testing the top m services of k clusters has $\mathcal{O}(m^k)$ computational complexity. Our goal is to find a better solution than the aggregation of the top element in each cluster, not necessarily the best possible solution. For this purpose, we select *Simulated Annealing* (Kirkpatrick, Gelatt, and Vecchi 1983, Černý 1985), an optimization heuristic.

6.6.1 A brief Introduction to Simulated Annealing

Simulated Annealing (SA) is a heuristic for approximating a global optimum in complex mathematical problems. It is well suited for problems with discrete search space such as the order of cities in the traveling sales man problem.

Simulated annealing is an iterative process building on following basic components:

Candidate Solution contains the current best problem solution which is gradually improved.

Solution Energy Function measures the quality of a given solution. SA aims to find a solution with the lowest possible energy.

Neighborhood Function provides a new candidate solution based on the current solution. A good neighborhood function traverses the search space quickly, but produces new solutions that yield similar energy level to the preceding solution.

Transition Function decides whether to accept a new solution or to stick with the current one.

Cooling function gradually reduces the temperature. Large solution changes are less likely for lower temperatures.

We briefly outline the iterative process in Algorithm 12 as provided in the JUNG 1.7.6 framework[2]. We omit some configuration parameters for sake of clarity. Transition function and Cooling function are problem independent, thus introduced here. We discuss neighborhood function and energy function in the subsequent subsections. For now, we treat these as blackboxes.

Simulated annealing takes an initial solution (i.e., the top service from every cluster) and derives the corresponding energy. Simulated Annealing continues to evaluated similar solution as long as the temperature hasn't reached zero and there are more available iterations. A new solution is always accepted when it yields lower energy. Worse solutions are accepted with probability p_{SA} defined as:

$$p_{SA} = e^{\frac{-1*\delta_{energy}}{temp}} \tag{6.22}$$

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

where δ_{energy} is the energy difference between the current and new solution, *temp* is the current annealing temperature, and e is Euler's number $2.718\ldots$. Transitions to solutions with higher energy are possible as long as the temperature remains high, or the energy difference is very small.

The freezing process depends on the cooling rate and current iteration state. As long as the number of successful transitions is high (i.e., *success* close to *tries*) the system remains in a search space region that still provides many solutions with lower energy. The function for the temperature in the next iteration is defined as:

$$temp_n = r_{cooling}^{(limit_{accept} - \frac{success}{tries}) * tries} * temp \tag{6.23}$$

where *tries*, $r_{cooling}$, and $limit_{accept}$ are configuration parameters. For our experiments, we apply $tries = 100$, $r_{cooling} = 0.99$, and $limit_{accept} = 0.97$

Algorithm 12 Simulated Annealing Algorithm $\mathcal{SA}(maxIt, startTemp)$.

```
function ANNEALING($maxIt, startTemp$)
    $\mathcal{A} \leftarrow calcNewSolution(startTemp)$
    $nrg \leftarrow calcEnergy(S)$
    $temp \leftarrow startTemp$
    $iteration \leftarrow 0$
    while $temp > 0$ AND $iteration < maxIt$ do
        $success \leftarrow 0$
        for $tries$ do
            /* Neighborhood function provides a new solution. */
            $newSolution \leftarrow calcNewSolution(S, temp)$
            $nrg_{new} \leftarrow calcEnergy(newSolution)$
            $\delta_{energy} = nrg - nrg_{new}$
            if $doTransition(\delta_{energy}, newSolution, temp)$ then
                $S \leftarrow newSolution$
                $nrg \leftarrow nrg_{new}$
                $success++$
            end if
        end for
        $temp \leftarrow calcTemperature(temp, success)$
        $iteration++$
    end while
    return $\mathcal{A}$
end function
```

6.6.2 Simulated Annealing Energy Function

The energy function provides the tradeoff between requirement fulfilment (i.e., total weighted utility u_{agg}) and service distance (i.e., average interaction-based distance be-

Symbol	Meaning
$\mathcal{A}$	Solution consisting of one selected service in each cluster.
p_{SA}	Transition probability to accept solutions with higher energy.
δ_{energy}	Energy difference between two solutions.
$temp_n$	New temperature given the current temperature $temp$, and configuration parameters $r_{cooling}, limit_{accept}$, and $tries$.
x_{max}	Maximum utility as provided by the top ranked service from each cluster.
$dist_{max}$	Upper limit for interaction distance between top ranked services.
nrg	Energy of a given solution depending on $x_{max}, dist_{max}$ and solution specific utility $x_{agg}(\mathcal{A})$ and distance $dist_{avg}(\mathcal{A})$.
φ	Preference parameter for trade-off between maximum utility or minimum distance.
t_{nh}	Neighborhood selection threshold.
p_{nh}	Neighborhood selection probability.

Table 6.9: Symbols applied in Simulated Annealing.

tween selected services $dist_{avg}$). A solution consists of a service from each clusters. The total weighted utility u_{agg} combines the service utility values u for a given cluster weighted according to the clusters significance ($sig_{cluster}(k) = \sum_i \mu_i k * w_i$). A cluster's significance raises with increasing membership of important constraints. The average distance $dist_{avg}$ is the sum of distance between any two services in the corresponding interaction-based distance graph, divided by the cluster count.

First we need to scale u_{agg} and $dist_{avg}$ to the interval $[0, 1]$ to combine them in a single function. We obtain the maximum achievable weighted utility u_{max} from selecting the top service from every cluster. The maximum distance between service is unknown, but we have an upper limit: the distance for the top services $dist_{max}$. As no solution can yield higher utility than the top services, any solution with higher distance than $dist_{max}$ can safely be discarded. The subsequent energy function for solution $\mathcal{A}$ is defined as:

$$nrg = \varphi * \frac{u_{max}}{u_{agg}(\mathcal{A})} + (1 - \varphi) * \frac{dist_{avg}(\mathcal{A})}{dist_{max}} \tag{6.24}$$

where φ determines the preferences for achieving high overall quality, or rather low intra-service distance. With φ approaching 1, the top service in every cluster creates the best solution. Having φ approach 0, simulated annealing selects the same service for every cluster, thereby reducing the overall distance to zero.

The combination of top services yields an energy value of 1. Any better combination must exhibit lower energy by reducing the distance. Combinations that additionally come with lower utility need to yield proportionally lower distance.

6.6.3 Simulated Annealing Neighborhood Function

The neighborhood function generates a new solution given a current solution. The function needs to be able to (a) traverse the search space in short time and (b) find neighboring configuration with similar energy. The first requirement guarantees that the simulated annealing algorithms is able to reach all states in a timely manner, thus potentially identifying the optimum solution. The second requirement ensures the algorithm's convergence. A random solution is more likely to be worse (rather than better) than the current solution. Jumping between high energy states maintains a high temperature level, thereby keeping the system from cooling down and finding the desired areas of low energy.

Our neighborhood function addresses both concerns. We randomly select a cluster and exchange the current element with another element with probability p_{nh}. The neighborhood probability depends on the service distance and is defined as:

$$p_{nh}(s) = \begin{cases} \frac{1}{m-1} & if\ dist(s_{new}, s_{old})_{norm} \leq t_{nh} \\ \frac{1-\psi}{m-1} & otherwise \qquad with\ \psi = \frac{dist(s_{new},s_{old})_{norm}-t_{nh}}{1-t_{nh}} \end{cases} \tag{6.25}$$

where m is the number of services within each cluster, $dist(s, s_{current})_{norm}$ is the distance between two services normalized to interval $[0, 1]$ with the most distant service yielding 1 and the closest service yielding 0. The temperature ratio $\frac{temp}{2*maxTemp}$ serves as threshold t_{nh}. The probability functions resembles utility function (d) in Figure 4.8 with $limit_a = 0$, $limit_b = t$, and $limit_c = 1$.

Services that are in proximity of the current solution are more likely to be selected, than services further away. Besides distance, also the current temperature affects this probability. In the beginning, when temperature is still high, short distance and far distance jumps equally likely. Later in the process, this probability decreases linearly with distance.

This function enables to quickly traverse the complete search space at the beginning. Later, we still can reach every solution, but require more steps to do so. We assume two services in proximity to yield similar distances to common neighbors. Thus, as we increasingly select services that are close to their predecessor, the total distance will raise on average less than randomly selecting services. Subsequently, two candidate solutions will yield similar energy values. This avoids fruitless testing of solutions with high energy.

6.7 Evaluation of Weighted Clustering Techniques

We demonstrate the effect of weighted clustering on the Slashdot data set. Specifically, we compare for different importance weight sets the resulting constraint distribution across clusters. Our experiments also include an analysis of ranking differences between clustered and non-clustered constraints.

First, we outline the mapping of Slashdot data onto requirements and utility values. Subsequently, we present the general experiment procedure before we discuss our findings.

6.7.1 Mapping Slashdot to Constraints and Utility functions

As briefly outlined in subsection 4.4.3.1, slashdot postings are subject to a moderation system. Postings receive scores between -1 (low quality) and $+5$ (high quality). Postings by known Slashdot members are rated 1 by default. Anonymous posts initially receive score 0.

In addition, predicates enable classification of postings according to insightful, interesting, informative, funny, etc. content (Table 6.10). As the classification process remains optional, mostly valuable postings are scored. Notice the low count of negative postings tagged with Troll, Offtopic, Flamebait, or Redundant in Table 6.10. Constructive postings usually receive scores higher than 1. Most postings, however, remain without predicate at all (75%).

Predicate	Total Count	$Score \geq 2$
None	55484	156
Insightful	5494	5038
Interesting	3599	3264
Informative	3596	3294
Funny	3383	3056
Troll	678	1
Offtopic	501	0
Flamebait	461	0
Redundant	287	0
Total	73483	14809

Table 6.10: Total Slashdot posting count and postings of minimum score 2 count from the subdomains Ask, Entertainment, and Mobile between Jan 1st, 2008 and July 1st, 2008, grouped by predicates.

We treat slashdot users as entities in an ensemble similar to the evaluation in Section 4.4. In our case, total scores (i.e., *Score*) and total posting counts (i.e., *Count*) serve as service requirements. Specifically, we derive for a user the total posting count and the total score summarized across these postings for a given subdomain and predicate. We prefer to keep these two constraints separated, as relying on a single average user score favors users with very few postings which were lucky to receive high ratings. In contrast, users contributing regularly are unlikely to receive continuously high scores. We consider total scores to be equally important to total postings throughout our experiments. Thus constraints weights for pairs of these statistics (respective to predicates) are always identical.

Within each constraint, users with the highest total score (respectively total posting count) receive a utility value of 100, with the worst users having utility 0. The general question we can answer applying our weighted clustering approach to the Slashdot data set is: Which clusters do arise from a given set of subdomains and predicates, and how much do we benefit from selecting the best users from clusters compared to a regular ranking process.

6.7.2 Weighted Clustering Experiment Setup

We select a set of subdomains SD and identify all users who have submitted at least 5 postings in these subdomains that scored 2 or higher. Given a set of predicates P we arrive at the constraints set C of size $|P| * |SD| * 2$. For each user, we derive the utility values for all constraints.

We select three subdomains—Ask, Entertainment, and Mobile—and focus in particular on the scores of funny, interesting, and insightful postings arriving at 12 constraints (i.e., Ask-Fun-Count, Ask-Fun-Score, Ask-Ins-Count, etc.). Across the three subdomains, we select users having 5 or more postings rated 2+. We treat users below this initial threshold as services exhibiting capabilities we are not interested in. As a side effect, reducing the initial set of candidates (here 255 users) reduces the duration of the clustering process.

The clustering threshold model configuration with $\alpha = 0.3; \delta = 0.4$ for both constraint entropy and service entropy yields thresholds of $t_c = 0.98$ and $t_s = 0.96$, respectively. Both, constraint entropy and service entropy remain well below these limits ($H(c) = 0.76; H(s) = 0.64$) thus clustering takes place with fuzzy factor $m = 1.826$.

6.7.3 Unbiased, Non-weighted Clustering Experiment Results

First, we analyze unbiased clustering, where we ignore constraint weights and work with the unbiased utility matrix $\mathcal{X}$. The cluster quality metric identifies 12 clusters to optimally describe the constraints. Specifically, the resulting cluster membership places *Count* and *Score* constraints of every subdomain and predicate in the same cluster except for the statistics describing Ask-Insightful, Entertainment-Funny and Entertainment-Interesting which populate individual clusters. Cluster membership μ is larger than 0.9 for all constraints.

We calculate the pairwise Jaccard similarity between any two clusters for the top 10, 50, and 100 users. Figure 6.9 visualizes the resulting similarity matrix. Row 13 and column 13 contains the unclustered ranking set. We notice that even for the very small set of top 10 users some clusters yield high similarity. Specifically cluster 3 and 5 share eight users, cluster 4 and 10 share eight users, and cluster 6 and 11 have 9 users in common. This overlap increases with 50 and 100 top users. Although all clusters yield significant differences to the unclustered ranking, the clustering process has created three pairs of clusters that should belong together. Incidentally, these pairs comprise of the above mentioned constraints, where Score and Count of the same subdomain and predicate end up in different clusters (Ask-Insightful, Entertainment-Funny and Entertainment-Interesting). Merging these pairs would not reduce the overall clustering benefit.

6.7.4 Biased, Non-weighted Clustering Experiment Results

Second, we cluster with equal constraint weights but biased utility matrix $\mathcal{X}_b$. Constraints that exhibit low service fulfillment yield less importance during clustering than well supported constraints. Table 6.11 provides the constraint membership in the resulting six

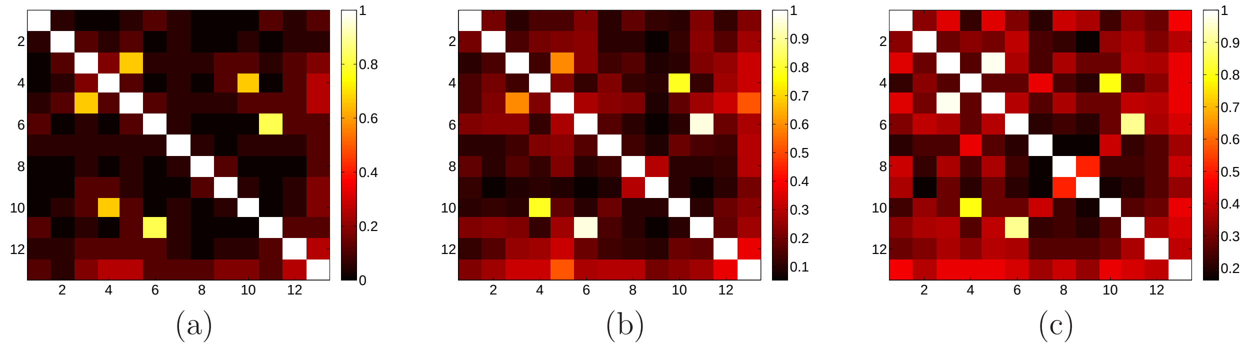


Figure 6.9: Cluster Jaccard similarity for Top 10 (a), Top 50 (b), and Top 100 (c) users for unbiased, non-weighted constraints.

clusters. Most constraints yield crisp cluster membership ($\mu > 0.9$) with exception to Entertainment Funny, Entertainment Interesting, and Mobile Funny which do not strongly belong to any cluster. The importance vector $\mathcal{T}$ is a good indicator on which constraints are likely to yield crisp clusters. A low importance value by itself, however, is not sufficient. The constraints Ask-Funny-Score ($\tau = 0.804$) exhibits lower importance than Entertainment-Funny-Count ($\tau = 0.836$) but ends up clearly assigned to cluster 1. Here, the close correlation of count and score values is decisive. The cluster weight aggregates the constraint weights proportional to their membership in that particular cluster.

We ensure that clusters provide more fitting elements than the unclustered ranking result by pairwise comparing the top-k elements with Pearson's correlation coefficient (ρ) and Jaccard similarity (J). Table 6.12 lists the ranking differences of the top 10, 50, and 100 users. The Jaccard similarity measures the set overlap of users regardless of their rank. Pearson's coefficient requires both sets to contain the same elements. We therefore take the union of elements from both rankings (given in brackets in Table 6.12) and then compute ρ.

Average Jaccard similarity increases with growing k, but remains still low for the top 100 users (of 255 in total). In other numbers, of the top-10 nonclustered users, only 30% occur in both rankings. For the top-50 and top-100 users this percentage is marginally higher: 33% and 35%, respectively.

The average Pearson's coefficient stresses the ranking differences even more. We observe a slight negative correlation for the top 10 users, no correlation in ranks for the top 50 users, and only a slight correlation for the top 100 users.

We compare also pairwise the six clusters to ensure that they constitute indeed distinct collections of constraints. We print the Jaccard similarity between any combination of clusters in Figure 6.10. The last row and column provides the overlap of the unclustered ranking as also provided in Table 6.12. In constrast to the first experiment, all overlaps between clusters remain low, even for the top 100 users. The highest similarity exists between clusters rankings and non-cluster ranking.

Evidently, clustering promotes distinctively different candidates than unclustered rank-

Constraint	$\mathcal{T}$	Cl 1	Cl 2	Cl 3	Cl 4	Cl 5	Cl 6
Ask-Fun-Count	0.864	**0.959**	0.005	0.011	0.012	0.008	0.006
Ask-Fun-Score	0.805	**0.968**	0.004	0.009	0.009	0.006	0.005
Ask-Ins-Count	1.394	0.019	0.01	0.023	0.025	**0.914**	0.01
Ask-Ins-Score	1.735	0.005	0.003	0.008	0.007	**0.975**	0.003
Ask-Int-Count	1.185	0.009	0.004	0.006	**0.969**	0.008	0.005
Ask-Int-Score	1.226	0.011	0.004	0.007	**0.964**	0.009	0.005
Ent-Fun-Count	0.836	**0.24**	0.12	0.176	0.158	0.104	0.202
Ent-Fun-Score	0.702	**0.253**	0.116	0.17	0.146	0.099	0.215
Ent-Ins-Count	1.035	0.01	0.004	**0.965**	0.007	0.01	0.004
Ent-Ins-Score	1.008	0.007	0.003	**0.976**	0.004	0.007	0.003
Ent-Int-Count	0.513	**0.338**	0.09	0.179	0.156	0.105	0.132
Ent-Int-Score	0.748	**0.291**	0.085	0.215	0.158	0.143	0.108
Mob-Fun-Count	0.563	0.197	0.136	0.123	0.112	0.087	**0.346**
Mob-Fun-Score	0.563	0.205	0.13	0.127	0.11	0.087	**0.341**
Mob-Ins-Count	1.459	0.003	**0.98**	0.003	0.003	0.003	0.007
Mob-Ins-Score	1.503	0.003	**0.983**	0.003	0.003	0.003	0.007
Mob-Int-Count	0.814	0.006	0.011	0.005	0.006	0.004	**0.969**
Mob-Int-Score	1.046	0.008	0.016	0.006	0.008	0.005	**0.957**
Cluster Weight		0.196	0.15	0.167	0.159	0.143	0.185

Table 6.11: Cluster membership and importance vector $\mathcal{T}$ for biased constraints from subdomains Ask, Entertainment, and Mobile with predicates Funny, Insightful, and Interesting.

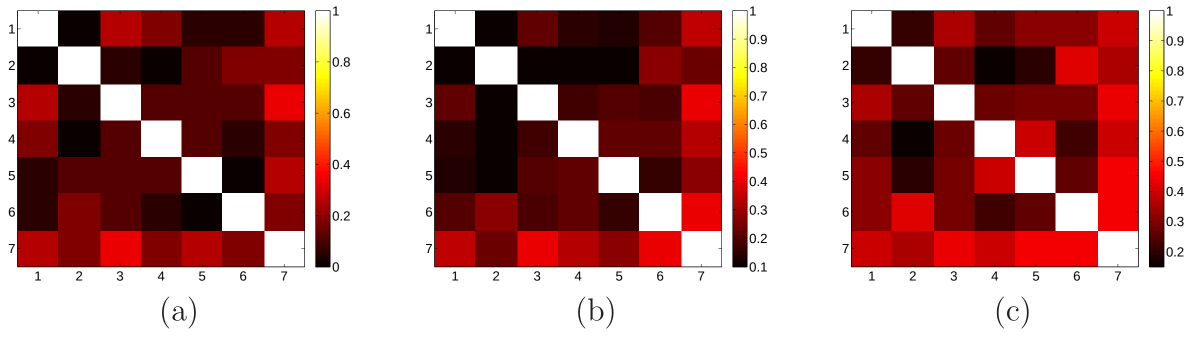


Figure 6.10: Cluster Jaccard similarity for Top 10 (a), Top 50 (b), and Top 100 (c) users for biased, non-weighted constraints.

ing. Table 6.13 provides a detailed view on the top 10 users of each cluster and the non-clustered rank. For each cluster, the table lists user id, cluster specific score, and position in the non-clustered set. Bold user ids highlight the top 10 elements from the non-clustered set.

There happens to be an outstanding user (957197) being ranked first in cluster 1, 3, 4, and 5. However, the user resides on position 35 and 41 in cluster 1 and cluster 6, respectively. Other elements in the non-clustered top 10 list, perform much worse. The

Cluster	Top 10 ρ	J	Top 50 ρ	J	Top 100 ρ	J
Cl1	-0.379 (16)	0.25	0.016 (74)	0.351	0.105 (144)	0.389
Cl2	-0.596 (17)	0.176	-0.152 (81)	0.235	-0.011 (147)	0.361
Cl3	0.314 (15)	0.333	0.193 (71)	0.408	0.217 (139)	0.439
Cl4	-0.316 (17)	0.176	0.014 (75)	0.333	0.005 (143)	0.399
Cl5	0.065 (16)	0.25	0.031 (78)	0.282	0.119 (138)	0.449
Cl6	-0.368 (17)	0.176	-0.061 (71)	0.408	0.194 (138)	0.449
Avg	-0.213 (16.3)	0.227	0.007 (75)	0.336	0.105 (141.5)	0.414

Table 6.12: Ranking differences of top 10, 50, and 100 users between each cluster and the unclustered ranking order measured with Pearson's correlation coefficient (ρ) and Jaccard similarity (J). Unweighted, biased constraints from subdomains Ask, Entertainment, and Mobile with predicates Funny, Insightful, and Interesting.

3rd user (595695) is on position 53 in cluster 1 and 38th in cluster 6. User 65584 is 155th in cluster 4 and 137th in cluster 6. Non-clustered 7th place becomes 128th in cluster 2, while 8th place ends up at position 142 in cluster 3. Every top 10 element, however, ranks better in at least one particular cluster (e.g., User 22995 (7th) is ranked 4th in cluster 3).

We observe similar differences in the opposite direction. Well positioned clustered users rank very low in the unclustered set. As listed in Table 6.13, user 963289, 3rd in cluster 2, is otherwise ranked 76th. User 1304191, on 7th place in cluster 6, ends up at position 199. Additional differences exist in the ordering of element. Positioned 6th in the unclustered set, user 22995 ranks before position 5 and 2 in cluster 3. In cluster 2, initial rank 8 outperforms rank 4 and 6. The analysis of ranking orders highlights considerable differences between the intra cluster order and the unclustered ranking order. We, thus, can conclude that clustering successfully promotes specialized users.

Next, we inspect the intra cluster utility values to measure the increase in benefit compared to the unclustered utility values. Table 6.13 includes the biased average for every position across all six clusters. Utility values in each cluster are weighted according to the clusters contribution (see Table 6.12). There is a 23% benefit increase when selecting the top users from every cluster compared to selecting the top unclustered user. The benefit increase for the second position is 62%. Selecting an aggregation across the top 10 users increases utility by 38% on average. This value neglects any sort of aggregation costs. However, it neither includes the advantage of redundancy from having multiple elements.

6.7.5 Biased, Weighted Clustering Experiment Results

We introduce constraint weights to demonstrate the effect on the clustering result. In this third experiment, we increase the importance of following four constraints: Entertainment-Interesting-[Count|Score] ($w = 0.08$) and Mobile-Funny-[Count|Score] ($w = 0.07$). The remaining constraints exhibit identical weights ($w = 0.05$) so the total weight equals 1 (see

NonCl		Cl 1			Cl 2			Cl 3		
Id	u	Id	u	Pos	Id	u	Pos	Id	u	Pos
957197	60.73	**957197**	76.17	1	238306	73.25	27	**957197**	76.33	1
835522	39.33	817932	48.75	24	25149	71.67	31	**655584**	71.90	4
595695	35.67	21727	47.46	36	963289	67.37	76	166417	60.29	13
655584	34.59	722131	41.87	44	1207026	67.13	25	**22995**	54.86	7
513215	34.29	898314	41.63	12	**641858**	67.10	8	**513215**	48.72	5
135745	33.29	945258	36.94	41	**655584**	63.58	4	**835522**	46.20	2
22995	31.87	**912633**	36.48	10	717556	58.15	73	869638	42.56	15
641858	31.26	**835522**	36.32	2	**135745**	55.89	6	987471	42.02	22
597831	31.16	**513215**	35.82	5	820751	54.60	11	95088	39.56	23
912633	30.50	869638	32.82	15	132727	50.32	16	122034	39.27	48
	All Cl	Cl 4			Cl 5			Cl 6		
	Avg	Id	u	Pos	Id	u	Pos	Id	u	Pos
	74.57	**957197**	78.15	1	**957197**	84.09	1	132727	61.92	16
	63.86	**597831**	70.05	9	**595695**	79.01	3	**835522**	49.20	2
	53.50	1015143	47.14	14	74366	61.00	89	**641858**	42.19	8
	49.40	12016	46.60	21	**135745**	57.82	6	680178	33.96	85
	47.53	**513215**	45.27	5	762201	54.13	82	1015143	33.68	14
	45.50	908688	44.62	100	**655584**	53.98	4	**22995**	33.46	7
	43.22	627338	42.96	137	611928	51.75	17	1304191	32.47	199
	42.29	898314	41.12	12	50515	51.65	74	238306	31.60	27
	41.25	784150	39.78	72	121541	50.93	56	942385	31.46	37
	39.71	611928	38.78	17	15695	50.64	149	965620	31.13	90

Table 6.13: Top 10 ranked users for unclustered and clustered evaluation for biased, unweighted constraints. *Pos* indicates the clustered element's position in the unclustered ranking.

also Table 6.14).

Clustering with these weights yield one more cluster. Also cluster membership has changed for some constraints. Both, Entertainment-Interesting and Mobile-Funny populate now their own cluster exhibiting high crispness. On the other hand, Ask-Funny looses its clear membership in a single cluster, now yielding fuzzy membership across all clusters. Entertainment-Funny maintains its fuzzyness but shares its largest membership with Mobile-Funny instead of Entertainment-Interesting.

The clustering process does not provide the same mapping of constraints to clusters for subsequent reruns. We reordered the cluster membership matrix in Table 6.14 to provide the best matching to the membership matrix of the previous experiment (Table 6.11).

Constraint	w	τ	Cl 1	Cl 2	Cl 3	Cl 4	Cl 5	Cl 6	Cl 7
Ask-Fun-Count	0.05	0.822	**0.253**	0.07	0.169	0.186	0.117	0.092	0.114
Ask-Fun-Score	0.05	0.766	**0.26**	0.067	0.166	0.183	0.115	0.091	0.118
Ask-Ins-Count	0.05	1.325	0.019	0.011	0.024	0.027	**0.901**	0.01	0.009
Ask-Ins-Score	0.05	1.649	0.005	0.003	0.007	0.007	**0.973**	0.003	0.002
Ask-Int-Count	0.05	0.901	0.008	0.004	0.007	**0.961**	0.009	0.006	0.004
Ask-Int-Score	0.05	0.932	0.008	0.004	0.007	**0.961**	0.01	0.006	0.004
Ent-Fun-Count	0.05	0.795	0.167	0.099	0.143	0.134	0.087	0.155	**0.215**
Ent-Fun-Score	0.05	0.667	0.165	0.092	0.133	0.12	0.08	0.156	**0.253**
Ent-Ins-Count	0.05	0.984	0.008	0.004	**0.967**	0.006	0.008	0.004	0.004
Ent-Ins-Score	0.05	0.958	0.008	0.003	**0.97**	0.005	0.007	0.003	0.004
Ent-Int-Count	0.08	0.732	**0.799**	0.023	0.045	0.041	0.027	0.032	0.032
Ent-Int-Score	0.08	1.066	**0.945**	0.006	0.014	0.011	0.01	0.007	0.007
Mob-Fun-Count	0.07	0.91	0.002	0.002	0.002	0.002	0.001	0.004	**0.987**
Mob-Fun-Score	0.07	0.911	0.002	0.002	0.001	0.001	0.001	0.003	**0.99**
Mob-Ins-Count	0.05	1.387	0.003	**0.976**	0.003	0.003	0.003	0.008	0.004
Mob-Ins-Score	0.05	1.429	0.003	**0.98**	0.003	0.003	0.003	0.007	0.003
Mob-Int-Count	0.05	0.774	0.008	0.014	0.006	0.007	0.005	**0.943**	0.018
Mob-Int-Score	0.05	0.994	0.003	0.007	0.003	0.003	0.002	**0.976**	0.006
Cluster Weight			0.177	0.119	0.135	0.115	0.119	0.126	0.209

Table 6.14: Cluster membership and importance vector $\mathcal{T}$ for biased, weighted constraints from subdomains Ask, Entertainment, and Mobile with predicates Funny, Insightful, and Interesting.

Again, we pairwise compare the non-clustered ranking and each cluster for ranking differences. We find low average Jaccard similarity increasing with growing k—similar to the unweighted experiment. We notice even stronger ranking order differences when comparing the average Pearson's coefficients in Table 6.15 and Table 6.12. Both, Top 50 and Top 100 users show hardly any rank correlation with the non-clustered user set.

Cluster 7 emerges not only due to the changed weights. We evaluate the pairwise cluster similarity to ensure that the underlying data justifies this additional cluster. Figure 6.11 provides the similarity matrix including the non-clustered set in the last row and column.

	Top 10		Top 50		Top 100	
Cluster	ρ	J	ρ	J	ρ	J
Cl1	-0.104 (15)	0.333	0.218 (71)	0.408	0.275 (138)	0.449
Cl2	-0.635 (17)	0.176	-0.199 (83)	0.205	-0.062 (147)	0.361
Cl3	0.207 (15)	0.333	0.097 (72)	0.389	0.161 (142)	0.408
Cl4	-0.373 (17)	0.176	-0.043 (79)	0.266	-0.046 (150)	0.333
Cl5	0.000 (16)	0.25	-0.062 (78)	0.282	0.011 (146)	0.37
Cl6	-0.622 (18)	0.111	-0.164 (73)	0.37	0.099 (142)	0.408
Cl7	-0.385 (18)	0.111	-0.148 (76)	0.316	0.117 (136)	0.471
Avg	-0.273 (16.6)	0.213	-0.043 (76)	0.319	0.079 (143)	0.400

Table 6.15: Ranking differences of top 10, 50, and 100 users between each cluster and the unclustered ranking order measured with Pearson's correlation coefficient (ρ) and Jaccard similarity (J). Weighted, biased constraints from subdomains Ask, Entertainment, and Mobile with predicates Funny, Insightful, and Interesting.

Cluster 7 remains distinctively different from the other clusters for the top 10, 50, and 100 users.

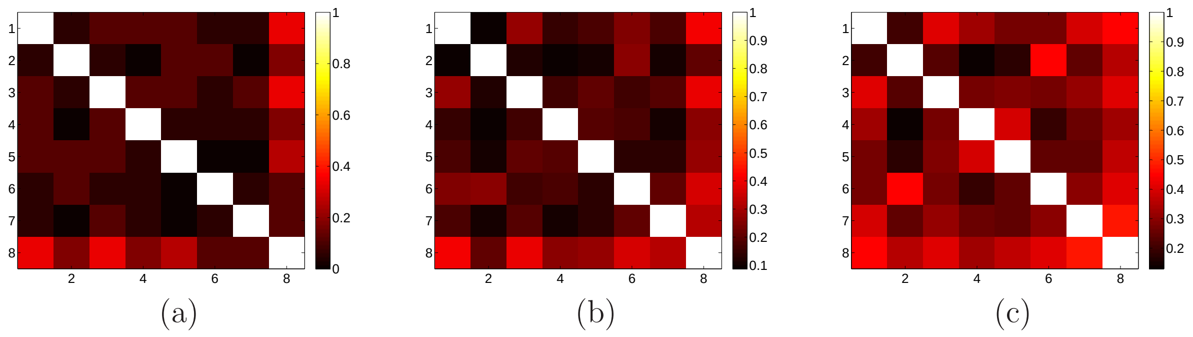


Figure 6.11: Cluster Jaccard similarity for Top 10 (a), Top 50 (b), and Top 100 (c) users for biased, weighted constraints.

The additional cluster allows for further specialization compared to the previous experiment. The benefit increase in the previous experiment is given in brackets. There is a 47% (23%) benefit increase when selecting the top users from every cluster compared to selecting the top unclustered user. The benefit increase for the second position is 14% (62%). Most importantly, however, the set of top 10 users yield a utility increase by 51% (38%) on average.

Table 6.16 lists the top 10 members in the seven clusters, their cluster specific utility value, and their position in the unclustered ranking. In the top 10 unclustered elements experience, new user 898314 (9th) replaces 597831 (formerly 9th). Across the seven clusters, 13 new users arise (in italics in Table 6.16), whereas three former users drop out. In total, the six clusters from the previous experiment exhibit 41 distinct users. With an additional cluster, this experiment exhibits 51 distinct users.

Clusters that display little membership differences to the previous experiment yield mostly the same users in slightly different order. Cluster 5, for example, exhibits the same top 10 users, of which the lower five lost (respectively gained) no more than three places. Clusters 2, 3, and 4 experience a single new user, with clusters 1 and 6 containing two new users each. Within cluster 1, focus shifted from Ask-Funny to Entertainment-Interesting and in cluster 6 Mobile-Funny has no longer any impact as it now populates its own cluster (7). Unsurprisingly, cluster 7 yields seven new users. Users 86149, 243267, 945888, 622222, 844560, 933028 reside on places between 27 and 105 in the non-clustered ranking.

The set of novel users in cluster 7 demonstrates the effectiveness of constraint weights. Promoting previously low fulfilment constraints—such as Mobile-Funny—creates cluster configurations that subsequently recommend users with the required capabilities.

NonCl Id	u	Cl Avg	Cl 1 Id	u	Pos	Cl 2 Id	u	Pos	Cl 3 Id	u	Pos
957197	58.20	84.45	**957197**	84.36	1	238306	82.63	33	**957197**	84.52	1
835522	43.60	68.82	**898314**	83.74	9	25149	78.13	28	**655584**	76.72	5
22995	34.25	56.71	*645701*	55.65	62	963289	75.73	82	166417	64.44	16
595695	33.70	52.54	**912633**	48.70	8	1207026	74.30	23	**22995**	53.39	3
655584	33.11	50.94	817932	47.39	18	**641858**	73.15	10	**513215**	52.87	6
513215	31.96	47.92	*679338*	46.99	26	**655584**	70.19	5	869638	46.89	15
135745	31.39	45.68	**22995**	40.08	3	717556	65.71	84	987471	46.27	21
912633	30.54	44.31	**595695**	39.78	4	**135745**	61.69	7	*33014*	43.24	30
898314	29.69	43.17	132727	39.27	14	820751	59.61	11	122034	41.75	54
641858	29.47	41.34	25149	39.04	28	*1280296*	55.65	39	**835522**	41.47	2
Cl 4 Id	u	Pos	Cl 5 Id	u	Pos	Cl 6 Id	u	Pos	Cl 7 Id	u	Pos
957197	85.00	1	**957197**	90.88	1	132727	77.45	14	**835522**	85.76	2
597831	71.23	13	**595695**	81.54	4	**641858**	47.92	10	*86149*	49.82	58
513215	48.00	6	74366	65.65	115	680178	44.68	99	*243267*	48.73	43
12016	46.36	25	**135745**	62.36	7	1304191	42.76	202	*945888*	46.57	40
1015143	44.97	12	762201	58.23	116	238306	41.66	33	**22995**	44.80	3
908688	42.58	124	121541	56.79	71	597831	39.51	13	622222	39.67	27
602015	42.19	24	**655584**	55.89	5	*1294206*	39.31	227	722131	38.58	38
784150	41.48	106	50515	55.44	92	*913150*	38.24	125	1015143	37.84	12
627338	40.93	156	15695	54.53	161	**835522**	36.59	2	*844560*	36.75	67
898314	37.81	9	611928	52.99	17	965620	36.55	74	*933028*	33.26	105

Table 6.16: Top 10 ranked users for unclustered and clustered evaluation for biased, weighted constraints. *Pos* indicates the clustered element's position in the unclustered ranking.

6.7.6 Discussion of Clustering Experiments

The three experiments have shown how constraint weights influence the clustering result. Biased, non-weighted clustering highlights the constraints that are fulfilled by most users (i.e., services in an ensemble). Subsequently, biased, weighted clustering successfully shifts the focus onto the constraints considered more important. New, or changed, clusters emerge only when the preferred constraints indeed meet a distinct difference in the underlying capability data compared to the remaining constraints.

Regular clustering, as tested in the unbiased experiment, produced too many clusters. However, both biased experiments exhibited very distinct clusters. Inter cluster Jaccard similarity remained in the range of $[0, 0.25]$, $[0.02, 0.28]$, and $[0.13, 0.42]$ for the top-10, top-50, and top-100 users (out of 255), respectively. The corresponding similarity between the unranked result and each cluster yielded slightly higher values.

Clustering also promotes users to the top elements in a cluster which are badly ranked in the non-clustered set. The Pearson's correlation coefficient emphasized the element positioning difference between non-clustered and clustered ranking order for both biased experiments. We observed a negative correlation for the top-10 users, no correlation for the top-50 users, and hardly a correlation for the top-100 users. The average benefit for selecting the top-10 users in every cluster amounts to a 38% to 51% utility increase compared to the unclustered ranking result.

6.8 Evaluation of Service Recommendation

In this section, we finally combine the Slashdot-based distance graph from Section 4.4, and the Slashdot user statistics from Section 6.7. Before we provide the aggregation results for the experiments in Section 6.7, we discuss one fundamental factor determining Simulated Annealing's ability to provide better solutions than the set of top ranked elements: *Capability Assortativity.*

6.8.1 Capability Assortativity

Simulated Annealing is most helpful when users in a common neighborhood yield substitutable capabilities. Substitutable capabilities occur when closely connected users exhibit similar capabilities, while maintaining only weak links to users of different capabilities. The top users exhibit thus high intra-cluster connectivity, but low inter-cluster connectivity. Under these circumstances, Simulated Annealing can find a good tradeoff between shortest distance between clusters and selecting suitable elements.

On the other hand, when users yield complementary capabilities, the top elements from the clusters will most likely provide one of the best solutions. When the top ranked users maintain high inter-cluster connectivity, applying a heuristic such as Simulated Annealing might not provide better results than brute force testing of the top few elements.

The distance graph's topology is one of the key factor determining a trend towards complementary or substitutable capabilities. We measure this trend with the assortativity metric.

Assortativity, in general, describes the degree characteristics of nodes. An assortative graph contains nodes of high degree connecting to other nodes of similarly high degree, while nodes of low degree have neighbors of equally low degree (Newman 2002, Newman 2003). Dissortativity describes the inverse effect with high degree nodes linking to low degree nodes.

For our purpose, we apply the concept of assortativity on the neighborhood cluster rank. Specifically, we define intra-cluster assortativity and inter-cluster assortativity. The former describes if elements tend to connect to elements of similar rank position within a cluster. The latter describes whether elements in one cluster tends to link to similar ranked elements in other clusters, or if they rather attach to better ranked elements.

The capability assortativity considers only ranked elements. Thus other users in the distance graph that do not show up in the ranking order (because the do not fulfill a single requirement) are ignored. They would distort the result.

In addition, capability assortativity includes edge weights. Regular assortativity measurements focus only on degree. Here, we apply the distance between elements to give more weight to ranks of closer neighbors, than to more distance neighbors.

The intra-cluster capability assortativity for an element s in cluster k is defined as:

$$CA_{intra}(s,k) = \left[\sum_{i}^{h} score(i,k) * \frac{1}{dist(s,i)} \right] * h^{-1} \tag{6.26}$$

where h is the number of neighbors of s that are also ranked, $score(i,k)$ determines the score of element i in cluster k, and $dist(s,i)$ provides the distance between elements s and i in the interaction-based distance graph.

The inter-cluster capability assortativity for an element s in cluster k is defined as:

$$CA_{inter}(s,k) = \left[\sum_{j}^{|K|} \sum_{i}^{h} score(i,j) * \frac{1}{dist(s,i)} \right] * (h * |K|)^{-1} \qquad \forall\, j \neq k \tag{6.27}$$

where $|K|$ is the number of available clusters.

In Figure 6.12 we print the user's ranking score against the intra-cluster and inter-cluster assortativity for each cluster. A negative slope indicates dissortativity, while a positive slope indicates assortativity (Lee, Kim, and Jeong 2006). Highlighted with linear best fit, we observe a distinct capability assortativity within each cluster, and distinct capability dissortativity across clusters. The clusters correspond to the biased, unweighted clustering experiment in the previous section (6.7) for predicates funny, interesting, and insightful for the subdomains ask, entertainment, and mobile. Assortativity measurements for unbiased, unweighted and biased, weighted experiments yield similar trends for intra-cluster and inter-cluster assortativity.

Intra-cluster assortativity indicates that the top users are well connected within their cluster. Inter-cluster dissortativity highlights lower connectivity across the clusters' top users. Slashdot data thus yields closely connected users with substitutable capabilities. Subsequently, Simulated Annealing offers considerable benefit in finding optimum aggregations.

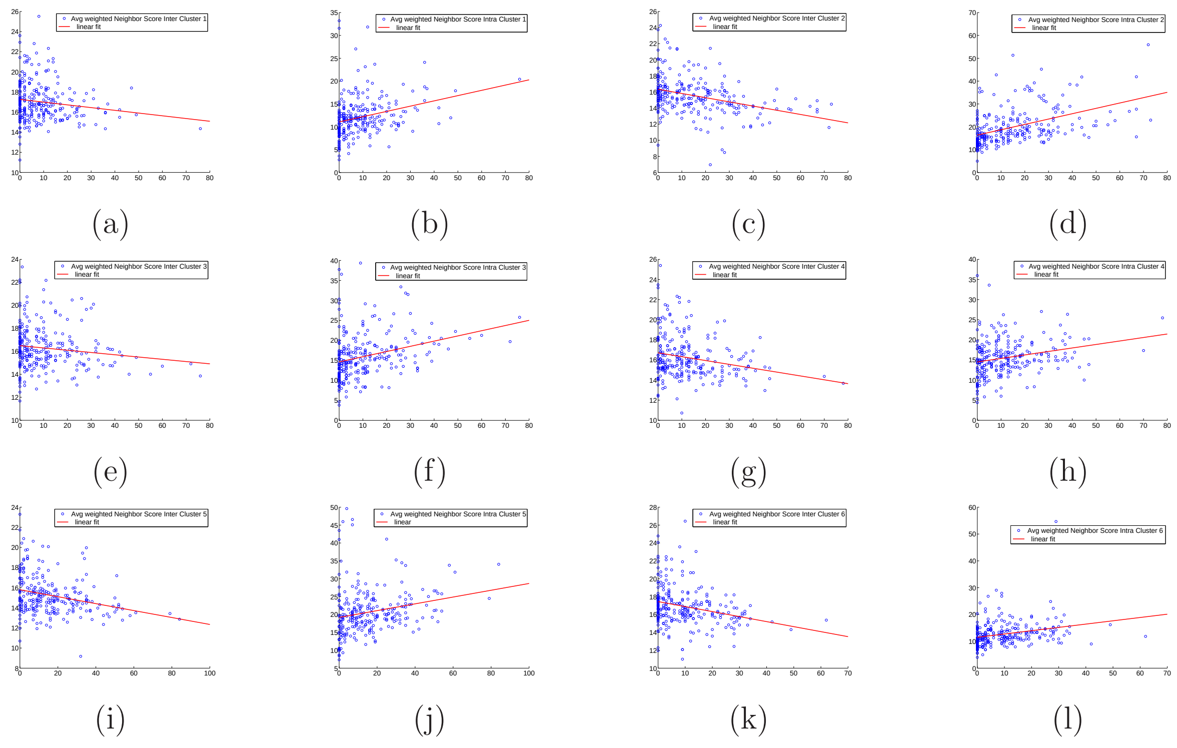


Figure 6.12: Intra-cluster and inter-cluster Capability Assortativity for biased, unweighted clustering results of predicates Funny, Interesting, and Insightful for subdomains Ask, Entertainment, and Mobile.

6.8.2 Simulated Annealing Aggregation Experiments

We apply Simulated Annealing on the three experiments previously outlined in Section 6.7. In all three cases, we utilize the same underlying interaction network. We extract the bipartite graph as present in Section 4.4.3.1 utilizing an activity aggregation energy parameter of 1, and including only users with a minimum of 5 postings.

The complete action graph contains the subdomains ask, entertainment, and mobile and exhibits 2497 users. We require a complete graph comprising more users than are contained in the cluster rankings to determine the correct user distance. We apply the interaction-based distance metric as we are interested in the focus of joint collaboration between users rather than their general involvement in common activities.

6.8.2.1 Aggregation of unbiased, non-weighted clustering results

Selection of top elements across 12 clusters yields a maximum weighted utility $x_{max} = 97.93$ for a maximum distance $dist_{max} = 24.11$. The set of top users comprises user 835522, 3x 957197, 898314, 2x 912633, 597831, 132727, 238306, 595695, and 655584.

The simulated annealing process reduces the average utility to 90.84 but lowers the distance even more: $dist = 20.65$. The optimized set consists of the same members with exception to user 820751 (position 38 with score 32.55) replacing 597831 (position 1 with score 98.79) in cluster 7. The next best solution considered by the simulated annealing process is already the set of top elements.

6.8.2.2 Aggregation of biased, non-weighted clustering results

Biased, non-weighted clustering created six clusters. The best achievable capability aggregation consists of users 4x 957197, 238306, and 132727 yielding a utility benefit of $x_{max} = 74.57$ with distance: $dist_{max} = 4.79$.

Again, Simulated Annealing provides a better tradeoff between distance and utility. The set of users 3x 957197 (Cl1:1, Cl3:1, Cl4:1), 135745 (Cl2:8), 655584 (Cl5:6), 835522 (Cl6:2) provides a slightly reduced utility ($x_{agg} = 65.31$), as considerably reduced distance ($dist = 1.23$). Cluster membership and ranking position are given in brackets.

Here, adding (and replacing) users provides a better tradeoff than aggregating with fewer users. Simulated Annealing also identifies solutions with a less optimal tradeoff. These solutions either exhibit lower distance (combined with lower utility), or exhibit higher utility (with greater distance). Among the top 20 solutions, distance becomes as little 0.48, then generating utility 56.6. For this experiment, no solution (except the top cluster elements) provides better utility than the optimal tradeoff.

6.8.2.3 Aggregation of biased, weighted clustering results

The final clustering experiment highlighted seven clusters for biased, weighted constraints. Best top cluster elements comprise the familiar users of the previous experiments: 132727, 4x 957197, 238306, and 835522. The corresponding aggregation yields utility $x_{max} = 84.44$ at a distance of $dist_{max} = 6.07$.

The best tradeoff comprises user 957197 for clusters 1 to 6, and user 835522 for the additional cluster 7. These two users yield an aggregated utility value of 72.84 at distance 0.34.

Sieving through the top 20 solutions, we find no aggregations with lower distance but some with higher utility. These solutions include one aggregation instance additionally including user 135745 for cluster 2 at rank position 8. It thereby replaces user 957197 formerly in cluster 2 at position 35. This combination provides a weighted utility value of 75.01 but comes with a distance of 0.65. Another combination, albeit with a worse tradeoff, additionally adds users 595695 and 513215 in cluster 5 and 6, respectively. This aggregation yields a utility of 73.92 at distance 1.76.

6.8.3 Simulated Annealing Evaluation Summary

Simulated Annealing provides benefit when services with similar capabilities are loosely connected. We applied the assortativity analysis of he Slashdot data underlying the clustering experiments. We found intra-cluster assortativity and inter-cluster dissortativity, subsequently demonstrating the need for Simulated Annealing.

Simulated Annealing found aggregations that provided a better tradeoff between capability utility and distance than the top cluster elements for all of the three clustering experiments. The optimal tradeoff found for the unbiased, non-weighted experiment exhibited an energy value of 0.97, only a slight improvement over the top cluster elements (which always yield energy 1). We achieved an SA energy value of 0.7 for the biased, non-weighted experiment, and energy 0.62 for the third, biased and weighted clustering process.

Chapter 7

Design and Implementation

7.1 Architecture

The *Service Ensemble Adaptation Architecture* encapsulates the main capabilities as Web services. The architecture comprises core services for management, context provisioning, and adaptation of ensemble entities (see Figure 7.1).

The *Ensemble Management Services* enable modeling and tracking of service capabilities (Service Capability Mgmt), organization and structuring of activities (Activity Mgmt), managing of human communication channels (Account Mgmt), and aggregating humans and services according to organizations, teams, or groups (Group Mgmt). *Logging* components capture all interactions between core management services and other ensemble entities.

Context Provisioning Services consist of context sensors, context aggregation functionality, ranking capabilities, and context provisioning endpoints. Logging is the main source of raw interaction data. *Context Sensors* analyze interactions and generate the appropriate actions. *Context Aggregation* determines ensemble-centric metrics. *Context Ranking* establishes the relevant set of context for a specific situation and client. Finally, *Context Provisioning* supplies push and pull based context updates.

Adaptation Services react to context and ensemble changes. *Property Interaction Impact* determines the significance of impact factors. *Ensemble Service Recommender* establishes on demand the set of most suitable services to forward a request to given the current context and property impact. *Infrastructure Reconfiguration Recommender* continuously tracks ensemble requirements and publishes optimal service changes.

In a service ensemble, it is natural to provide context and adaptation capabilities as Web services. Exchangeability and composability are amongst the main reasons to apply SOA principles also within the adaptation framework. Customized adaptation services can provide recommendations tailored to the particular ensemble domain. We are able to aggregate sensor services from different providers, thereby adding new context sources or increasing context accuracy. Third party services can utilize existing context and recommendation services to address specialized adaptation requirements.

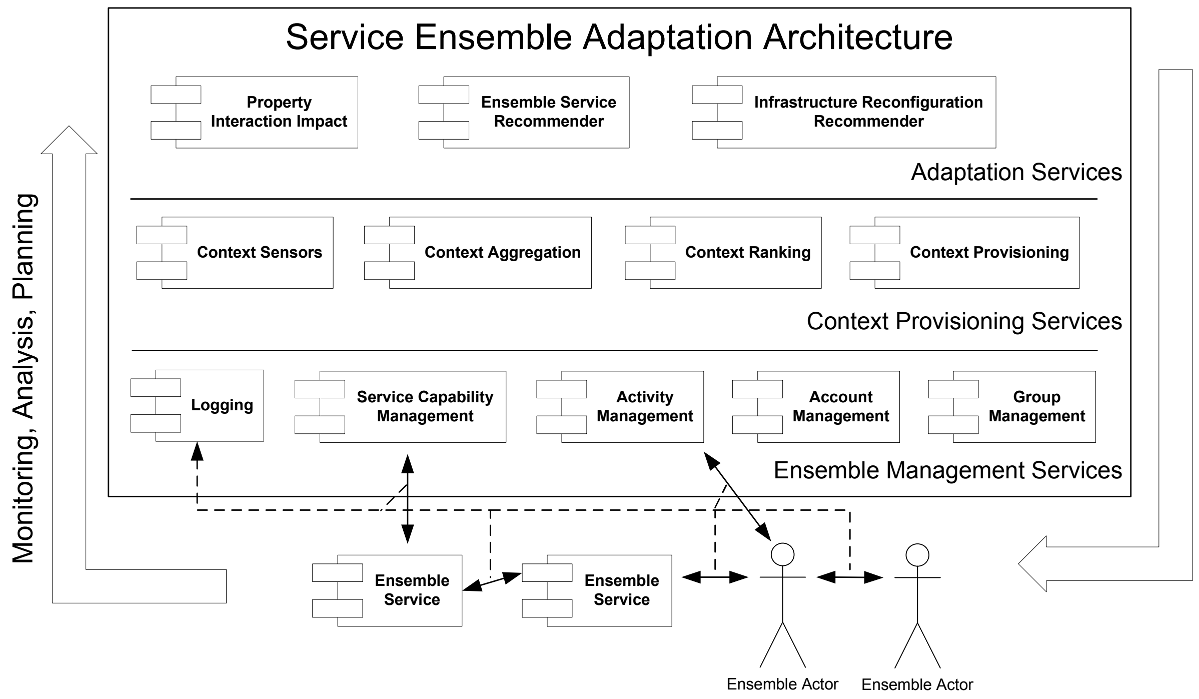

Service Ensemble Adaptation Architecture
Property Interaction Impact
Ensemble Service Recommender
Infrastructure Reconfiguration Recommender
Adaptation Services
Context Sensors
Context Aggregation
Context Ranking
Context Provisioning
Context Provisioning Services
Logging
Service Capability Management
Activity Management
Account Management
Group Management
Ensemble Management Services
Monitoring, Analysis, Planning
Execution / Management
Ensemble Service
Ensemble Service
Ensemble Actor
Ensemble Actor

Figure 7.1: Service Ensemble Adaptation Architecture overview.

7.2 Ensemble Management Services

7.2.1 Capability Management Service

The *Capability Management Service* provides operations for managing service profiles. Section 6.3 discusses the structure of service profiles. Here we present the corresponding operations and technical details. The service consists of three main components:

- Profile Registry: wraps the backend XML database storing the service capability profiles. The native XML database eXist[1] provides the necessary storage and retrieval methods. Profile inserts are executed directly via REST PUT requests. Queries and updates utilize XQuery and XUpdate statements embedded in REST PUT requests.
- Query Handler: transforms the supplied query parameters into an XQuery statement and retrieves the fitting profiles from the Profile Registry. Service clients can only provide the query identifier and required input parameters. Available queries extract

[1]http://exist.sourceforge.net/

profiles based on capability identifier and allow refinement by additionally specifying required property identifier, property values, or property value range.

- Change Handler: evaluates profile updates and publishes corresponding change events (Figure 7.2). *RepositoryChange* events inform about highlevel changes, such as new, changed, or removed profiles. The actual changes are described in the *ProfileChange* event. Notifications are made available via Atom feed, JMS, and WS-Notification. All three types contain the same event format.

The service operations include:

registerService adds a complete service profile to the *Profile Registry*.

unregisterService removes a complete service profile from the *Profile Registry*.

updateServiceCategories replaces the existing service category membership.

updateServiceProfile replaces a complete service profile.

updateServiceComponent replaces a complete component within a profile.

updateServiceCapability replaces a complete capability within a component.

queryCapabilities supplies one or more queries (containing query identifier and query parameters). The operation returns a set of matching service profiles. The service client can configure whether multiple queries within a single query request are treated disjunctive or conjunctive.

7.2.2 Activity Service

The *Activity Service* manages data instances of the activity model (introduced in Section 4.1). It supports structuring of ensemble activities (i.e., goals), and assignment of roles, resources, artifacts; allows managing of deadlines and tracking of progress. The service provides coordination primitives are inspired by (Dustdar 2004). The three major components are:

- Coordination Handler: manages the structure of activities and handles *Delegation* requests. A delegation remains in a user's *Inbox* until s/he accepts or rejects the delegation (using the respective *RespondDelegation* message). Users are also able to notify about starting or stopping work on an activity.

- Activity Store: wraps the backend XML-enabled database. We utilize JDBC to connect to the IBM DB2 database. The hybrid approach allows us to combine SQL and XQuery statements. Activities create a hierarchy using URIs as references to parent

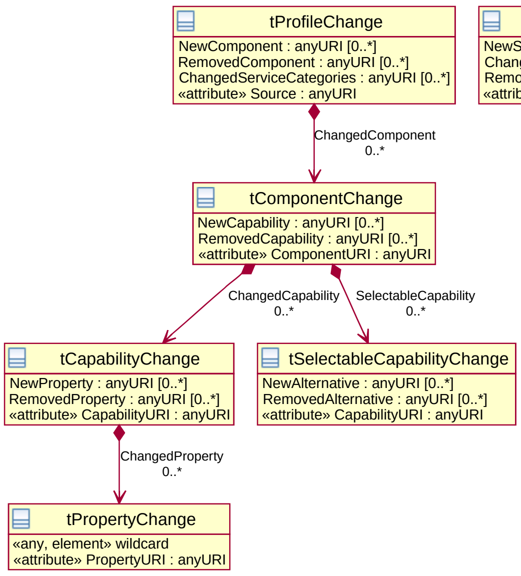

tProfileChange
NewComponent : anyURI [0..*]
RemovedComponent : anyURI [0..*]
ChangedServiceCategories : anyURI [0..*]
«attribute» Source : anyURI
tRepositoryChange
NewServiceProfile : anyURI [0..*]
ChangedServiceProfile : anyURI [0..*]
RemovedServiceProfile : anyURI [0..*]
«attribute» RepositoryURI : anyURI
ChangedComponent
0..*
tComponentChange
NewCapability : anyURI [0..*]
RemovedCapability : anyURI [0..*]
«attribute» ComponentURI : anyURI
ChangedCapability
0..*
SelectableCapability
0..*
tCapabilityChange
NewProperty : anyURI [0..*]
RemovedProperty : anyURI [0..*]
«attribute» CapabilityURI : anyURI
tSelectableCapabilityChange
NewAlternative : anyURI [0..*]
RemovedAlternative : anyURI [0..*]
«attribute» CapabilityURI : anyURI
ChangedProperty
0..*
tPropertyChange
«any, element» wildcard
«attribute» PropertyURI : anyURI

Figure 7.2: Capability Change model UML class diagram

and child activities. Subsequently, each activity instance populates a separate row in the database. SQL provides recursive iteration techniques to retrieve a complete activity subtree (see Listing 7.1). We limit XML content in the database to the actual activity data. Additional metadata for managing historical and deleted activity instances remain as regular SQL columns (e.g., Listing 7.1 line 5: ISDELETED or line 6: ISNEWEST).

- Query Handler: accepts a reference to a query and the corresponding query input encoded as XML. The Query Handler retrieves the query from the Query service (if not already cached) and replaces the placeholders in the query with the parameters from the query input. As context sensors and context clients make most use of the Query service, we provide details and design rationale in the next subsection. Three default queries enable retrieval of (i) all activities a user is involved in, (ii) all activities within an activity subtree, and (iii) all activities delegated to but not yet accepted by a particular user.

The Activity service supports following operations:

addActivities accepts multiple new activities at the same time. The client can provide different Activity subclasses within the same request. The service creates activity identifiers upon successful storage. Consequently, the set of new activities within a request can refer only to stored activities, but cannot establish a tree structure themselves. This has to be done using *setDetails* and *addDetails* operations.

setDetails manipulates all single-valued properties such as name, description, parent, or start. The service overwrites all such properties, thus a clients need to provide also properties that haven't changed.

addDetails affects only multi-valued properties such as tags, childActivities or memberInvolvements. The provided activity instance needs to include only the difference (i.e., the added details). The service adds the provided properties to the set of existing properties. Duplicate entries are removed, thus any tag, child activity, or resource occurs at maximum once.

removeDetails is the inverse operation to *addDetails*. The operation will remove all multi-valued properties provided from the activity instance in the database.

deleteActivities marks the corresponding activities as deleted in the database, thus becoming inactive. An Activity having active child activities cannot be removed.

getActivities returns all activities identified by their URI.

queryActivities retrieves the referenced query and inserts the provided query parameters. Activities marked as deleted will not turn up in the result set unless the query explicitly addresses deleted activities. The service supports only one query per request.

delegateActivity requests to change the responsible member(s) of an activity. A delegation request consist of sender, receivers, activity identifier, and delegation type. The delegation type determines if the receiver is just a normal *Delegate* replacing the sender, or a *DelegateSplit* to clone the delegated activity for each receiver, or a *DelegateJoint* to divide responsibility amongst multiple members.

respondDelegation indicates the acceptance or refusal to take over responsibility. For accepted delegations, the service updates the activity by replacing the responsible person. The formerly responsible person becomes an observer.

notifyWork informs involved members and services about an person starting or stopping work.

7.2.3 Context Coupling Mechanisms

Ensemble services require context information to adapt to the user, activity, and overall ensemble. A service client is unlikely to know which context the service provider requires. It merely invokes the service which has to retrieve the required context by itself. As service clients potentially assume multiple roles within an ensemble, retrieval of the relevant context information becomes challenging.

Context coupling mechanisms provide a correlation of service invocations to client context. The client context includes the context changes of previously invoked services. A newly invoked service receives the context correlation information to adapt to the relevant context. Figure 7.3 outlines this process. Correlation information consists of activity URI and user URI. The basic coupling steps are:

```
WITH RAL (URI, ACTIVITYXML) AS
(  SELECT ROOT.URI, ROOT.ACTIVITYXML
    FROM $SCHEMA.ACTIVITY ROOT
    WHERE
    ROOT.ISDELETED = 0
    AND ROOT.ISNEWEST = 1
    AND xmlexists('declare default element namespace
        "http://www.in-context.eu/ns/activity";
        $c/tActivity[ParentActivity="$id"]'
        passing ROOT.ACTIVITYXML as "c")
    UNION ALL
    SELECT CHILD.URI, CHILD.ACTIVITYXML
    FROM RAL PARENT, $SCHEMA.ACTIVITY CHILD
    WHERE
    CHILD.ISDELETED = 0
    AND CHILD.ISNEWEST = 1
    AND xmlexists('declare default element namespace
        "http://www.in-context.eu/ns/activity";
        $f/tActivity[ParentActivity=$d]'
        passing CHILD.ACTIVITYXML as "f",
        PARENT.URI as "d")
)
SELECT URI, ACTIVITYXML FROM RAL
```

Listing 7.1: Recursive retrieval of sub activities within a activity hierarchy, starting at activity URI "$id"

Correlation Management consists of identifying the ensemble actor, and selecting the desired activity. Composite services pass the correlation information they have initially received to their embedded service client.

Correlation Establishment occurs when the client-side context coupling handler adds user and activity identifier to the SOAP header of the outgoing service invocation. Listing 7.2 contains an example SOAP message with activity and user references in the context coupling SOAP header extension.

Access Layer acts as transparent HTTP proxy, producing a copy of every SOAP request and reply. Each copy is passed to the Logging Service.

Logging Service provides a subscription interface. Every subscriber receives a copy of SOAP messages encapsulated in an interaction event (see Table 7.1). The next section details how context sensors subsequently generate context information.

Correlation Extraction The service-side handler extracts the correlation information from the inbound SOAP message. The service utilizes the references as entry points for accessing the actual context via the Context Retrieval services.

Most of the ensemble management services are part of the Pervasive Collaboration Service Architecture (PCSA) devised in the inContext project [2]. We designed and implemented the *Activity Management Service* and the *Capability Management Service* and

[2]`http://www.in-context.eu`

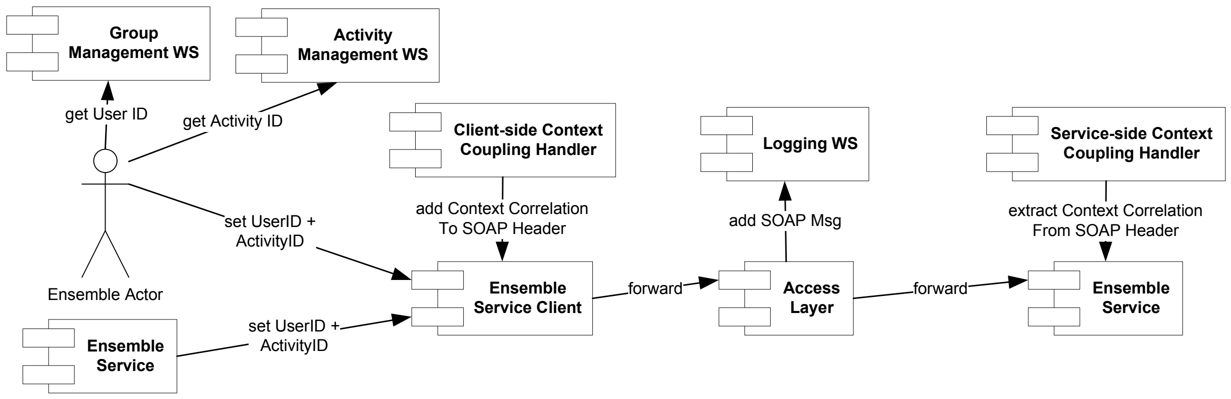


Figure 7.3: Context Coupling Mechanism.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Header>
    <ns1:activity_id
     soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
     soapenv:mustUnderstand="0" xsi:type="soapenc:string"
     xmlns:ns1="incontext"
     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
     http://www.in-context.eu/Activity/Activity#1624
    </ns1:activity_id>
    <ns2:user_id
     soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next"
     soapenv:mustUnderstand="0" xsi:type="soapenc:string"
     xmlns:ns2="incontext"
     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
     http://www.in-context.eu/User/User#7
    </ns2:user_id>
 </soapenv:Header>
 <soapenv:Body>
    <ns3:addDocument
     soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:ns3="http://localhost/Eadt/Tasks/DocService">
     <sessionId xsi:type="xsd:string">
      f625c495bff00a09eaf91b08d7b12a6e
     </sessionId>
     <profile xsi:type="xsd:string">
     </profile>
     <url xsi:type="xsd:string">
      http://localhost:80/inContext2/tmp_files_to_transfer/EC_link.txt
     </url>
    </ns3:addDocument>
 </soapenv:Body>
</soapenv:Envelope>
```

Listing 7.2: Example SOAP message with context coupling header

Property	Description
clientIP	the IP address of the service client
messageType	request or response
messageCorrelationID	for correlating request and response messages
serviceEndpoint	the service endpoint address
sourceID	the address of the Access Layer having intercepted the message
timestamp	the time at invocation
userURI	the invoking user on behalf of which a service client is acting
consumerID	the service client, which is invoking the actual service operation
message	a copy of the SOAP message

Table 7.1: Interaction Event properties.

participated in the development of the components enabling the context coupling process. Specifically, project deliverables D5.2 (inContext Consortium 2007b) and D5.3 (inContext Consortium 2007a) provide implementation details on the Account Management service, Group Management service (aka. Team Service and Team Management Service), and Access Layer. Deliverable D4.2v2 (Casella, Dorn, Polleres, and Yi 2008) provides more information on the context coupling SOAP header, while D2.2v2 (Dorn, Polleres, and Yi 2008) discusses Logging service specific details.

7.3 Context Provisioning Services

The context provisioning process (Figure 7.4) commences with incoming SOAP logs at the *Logging Subscriber Web service*, subsequently distributed to the various *Context Sensors*. The sensors apply the Ensemble Management services to reason about the service invocations before submitting context updates. The *Query and Update Store Service* provides the detailed structure of context updates and queries. *Metric Definitions* describes which ensemble metrics the *Ensemble Context Aggregation Service* should calculate, and which metrics are available for subscription at the *Metric Change Publisher Service*. The *Interaction Graph Manager* maintains distance measurements. The *Context Ranking Service* selects the most relevant context data given a set of context correlation identifiers while the *Context Retrieval Service* provides specific on-demand context information.

7.3.1 Context Sensing and Aggregation

The *Logging Subscriber Service* subscribes at the Logging Service. Upon receiving a new interaction event, it extracts the service operation from the SOAP message before passing the event to interested sensors. The main purpose of context sensors is transforming the incoming raw SOAP message to raw action events. Where required, they access the ensemble management services to obtain additional information. An email sensor, for

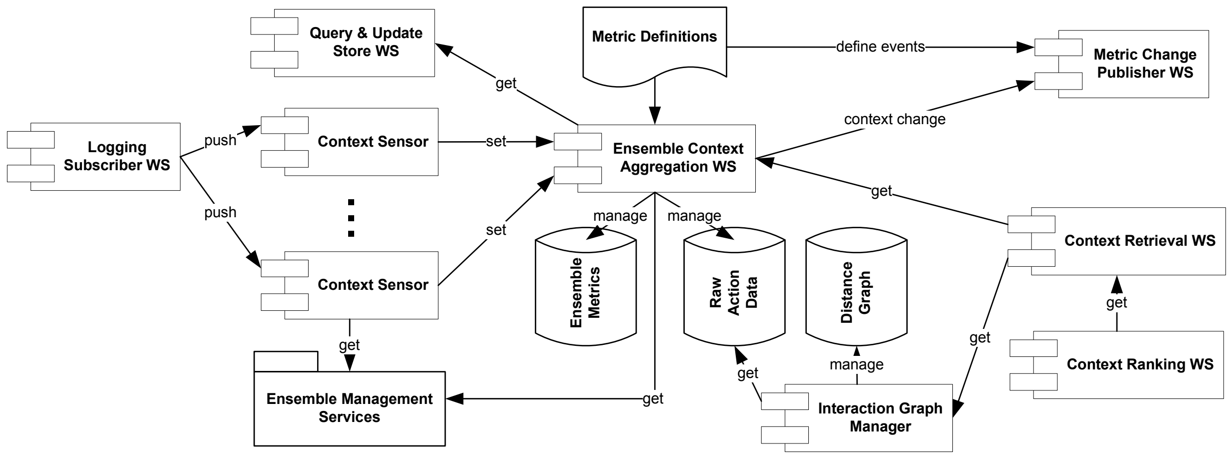


Figure 7.4: Context Provisioning Subsystem.

example, listens for successful invocations of the Email service (part of the PCSA) and resolves the receivers' email addresses at the Account Management Service. Thus it is able to create a communication action containing the involved user URIs.

A context sensors adds context data at the *Ensemble Context Aggregation Service* via two context input interfaces:

addAction accepts a new action and corresponding sensing metadata describing time, confidence, sensor id, etc. This operation captures the actions of all active elements in the ensemble.

addContext accepts updates as managed by the *Query and Update Store Service*. A sensor identifies the update statement, provides the corresponding input XML, and the sensing metadata as for addAction. This operation is intended for capturing context changes about elements in the ensemble such as location changes, or availability.

Internally, the ensemble context aggregation service stores the *Raw Action Data* but also computes *Ensemble metrics*. The *Metric Definitions* configure which metric calculation plug-ins exist and upon which update (identified by URI) they should be triggered. Similar to context sensors, a plug-in accesses ensemble management services to reason about metric changes.

The *Interaction Graph Manager* takes raw action data to generate the 4-partite action graph. It provides both context-based and interaction-based distance for activities, active entities (humans and services), resources, and artifacts.

7.3.2 Query and Update Store Service

The *Query and Update Store Service* manages authorized (context) queries and also context update. The decision to keep queries/updates separated from the actual storage services

Property	Description
identifier	the unique identifier for the query/update
humanReadableName	a short name describing the query/update
humanReadableDescription	a longer description on how the query is used and what it does
inputXSD	specifies the XML schema, which describes the format of the required XML input
inputExampleXML	a sample XML input to be supplied by the query/update invoker
statement	the query/update statement interpreted by the query/update engine together with the XML input
resultXSD	specifies the format of the returned XML result
outputExampleXML	an example output

Table 7.2: Query/Update object.

(e.g., Activity Service, Ensemble Context Aggregation Service) enables a simple, static service interface. Queries and Updates can be added and removed as required without having to change the services interface description. Service clients need not know the details of the query language which comes with following advantages:

1. Design and testing of queries is done by an expert familiar with the ensemble context model. This reduces the number of inefficient, incorrect, privacy infringing, or computationally complex queries.
2. Reuse of existing queries simplifies the process of writing new ones. Additionally, a known set of input and output XML schemas facilitate the provisioning of query input and processing of result data on the consumer side.
3. Service providers can validate the provided input XML.
4. Any changes to the context model can be evaluated against the necessary alterations of update and query statements, thus simplifying change management.
5. Context sensors do not need to know the details of the update language.

Table 7.2 provides the query/update object properties. It contains all information required during design-time to create queries/updates. During runtime the invoked service (e.g., Activity Service) retrieves only the *statement* part. The example query statement in Listing 7.3 returns all activities that have an artifact of given artifact type and given resource reference. The first passing input statement declares that the placeholder *$type* in the query template is to be replaced with the value of the XPath statement executed on the provided input XML.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TXQueryTemplate xmlns="http://www.in-context.eu/ns/contextquery">
    <QueryStmt>
        SELECT $SCHEMA.ACTIVITY.ACTIVITYXML
        FROM $SCHEMA.ACTIVITY
        WHERE $SCHEMA.ACTIVITY.ISDELETED = 0
        AND $SCHEMA.ACTIVITY.ISNEWEST = 1
        AND xmlexists('declare default element namespace
        &quot;http://www.in-context.eu/ns/activity&quot;;
        $c/tActivity/EditArtifacts[WrapsResourceURI=&quot;$ref&quot;
        and ResourceType=&quot;$type&quot;]'
        passing $SCHEMA.ACTIVITY.ACTIVITYXML as &quot;c&quot;)
    </QueryStmt>
    <PassingInput
        Placeholder="$type"
        InputXPathStatement="/ArtifactQuery[1]/ResourceType/text()"/>
    <PassingInput
        Placeholder="$ref"
        InputXPathStatement="/ArtifactQuery[1]/ResourceRef/text()"/>
</TXQueryTemplate>
```

Listing 7.3: Example Query statement for retrieving activities that have an artifact of given resource type and given resource reference

7.3.3 Context Retrieval

The *Context Retrieval Service* is the main source of on-demand context information. It provides two operations:

getAction accepts a query URI and corresponding input XML. The result is a set of actions.

getContext takes the same input parameters as getAction but returns an XML string as defined by the output schema of the corresponding query object.

Clients should use the *Context Ranking Service* when multiple context information instances (e.g., multiple Storage Services) occur in the ensemble. The context ranking service determines the most relevant ones given the user and activity identifier. Specifically, it provides following operations:

getRankedContext requires user and activity URI, a reference to a context query, the corresponding XML input, the maximum number of top ranked context elements, and whether to include the actions in which the desired entity was involved in. The operations returns a list of context resources (each as defined by the query) and the corresponding rank.

getRankedContextURIs accepts the same input parameters as getRankedContext, differing only in the format of the response message. This operation provides only the URI of the context object, rather than the complete representation.

The *Metric Change Publisher Service* offers a subscription endpoint for ensemble metric updates. The example subscription in Listing 7.4 expects notifications when the Property Distribution Entropy for location drops below 0.33 in ensemble Test1. The publisher services utilizes WS-Notification for delivering the metric events.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<MetricSubscription xmlns="http://www.in-context.eu/ns/contextquery">
    <ThresholdType>
        http://www.in-context.eu/ns/EnsembleMetrics/threshold_lowerbound
    </ThresholdType>
    <EnsembleURI>
        http://www.vitalab.tuwien.ac.at/projects/taaf/Ensemble#Test1
    </EnsembleURI>
    <MetricURI>
        http://www.in-context.eu/ns/EnsembleMetrics/PDE#Location
    </MetricURI>
    <Threshold>
        0.33
    </Threshold>
    <NotificationEndpoint>
        ... [WS-Addressing Endpoint Reference] ...
    </NotificationEndpoint>
</MetricSubscription>
```

Listing 7.4: Ensemble metric subscription example

7.3.4 Mobile Context Provisioning

The ensemble management service and other context provisioning service run on standard server hardware. The mobil context provisioning components target mobile devices such as PDAs, smartphones, and laptops. The OSGi [3] specification describes a JAVA based container environment for devices with limited memory and processing power.

OSGi exhibits service-like properties. Components are deployed as bundles that are dynamically updated, found, bound, and invoked. Knopflerfish[4] is the OSGi implementation of our choice. It comes with Web service support, thereby exposing specific bundles as Web services.

The mobile context provisioning subsystem (Figure 7.5) consists of several OSGi bundles. The *Context Event Publisher* and *Context Event Subscriber* are exposed as Web services. The former provides context events to interested remote clients. The latter subscribes for context events at remote context provisioning subsystems. Multiple mobile context provisioning subsystems thus form a peer-to-peer network.

Internally, the *Event Manager* takes context events from local *Context Sensor Bundles* and updates the *Mobile Context Store* as well as the event publishing bundle. The event manager also accepts incoming remote context events and merges them with the local context database.

[3] Open Service Gateway Initiative: `http://www.osgi.org`

[4] `http://www.knopflerfish.org/index.html`

The *Subscription Manager* triggers remote subscriptions when the local context store does not contain the data requested by the local *Context Query Bundle*. This bundle serves context to the remaining bundles within the local OSGi container.

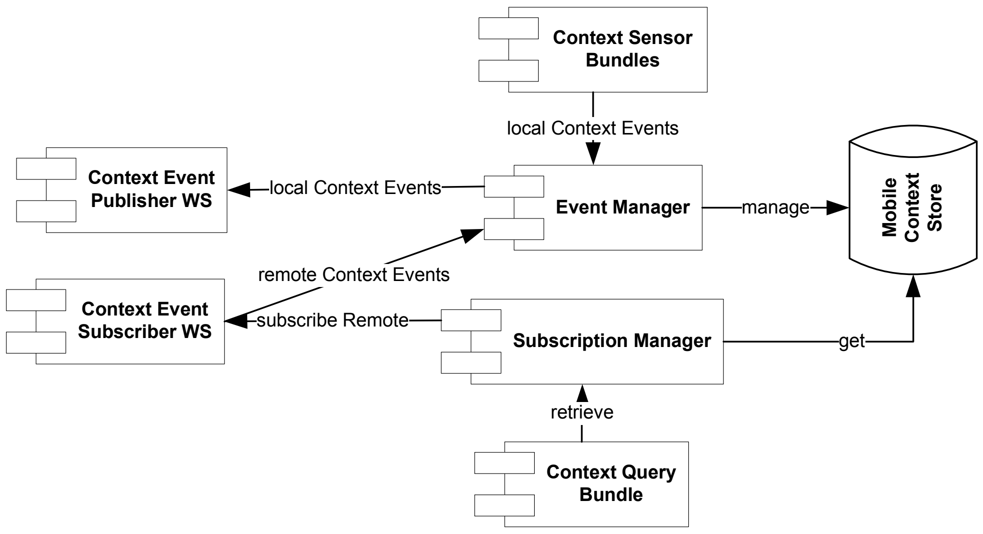


Figure 7.5: Mobile Context Provisioning subsystem.

7.4 Adaptation Services

Adaptation services rely upon context change events, especially ensemble metric updates. Figure 7.6 outlines the *Property Impact Evaluation* subsystem, and Figure 7.7 displays the *Infrastructure Adaptation* subsystem.

7.4.1 Property Impact Evaluation

The *Metric Change Subscriber* observes property-specific metric updates. The *Metric Definitions* specify which properties get monitored in the ensemble. The *Property Impact Potential* component calculates the possible impact of the changed property distribution. It subsequently configures the *Property Impact Evaluation* component with respect to the properties it should focus its interaction analysis on. In regular intervals, the *Property Impact Evaluation* component retrieves aggregated interaction data from the ensemble context services. Subsequently, it stores the actual interaction impact in the *Property Impact Trends* database.

When an ensemble service client invokes the *Ensemble Service Recommender Service*, it accesses the database to establish the best ranking for the invoking client. The ensemble management services provide the required data on the client's and neighboring services' property values. The *Ensemble Service Recommender Service* supports two operations:

getRankedServices takes the identifiers of the invoking client and the set of services that should be ranked. The identifiers enable extraction of the respective service properties from the ensemble management services. Finally, the operation returns a ranking score for each service.

getRankedPersons applies the same techniques as *getRankedService* but works with interactions of and properties about human ensemble entities.

Property impact trends are also directly available via the *Property Impact Provisioning Service*. It provides two operations:

getImpact accepts a property identifier and returns the corresponding impact matrix listing the impact between any two property values.

getImpactForValue limits the returned impact result to a particular row in the impact matrix (defined by property value identifier).

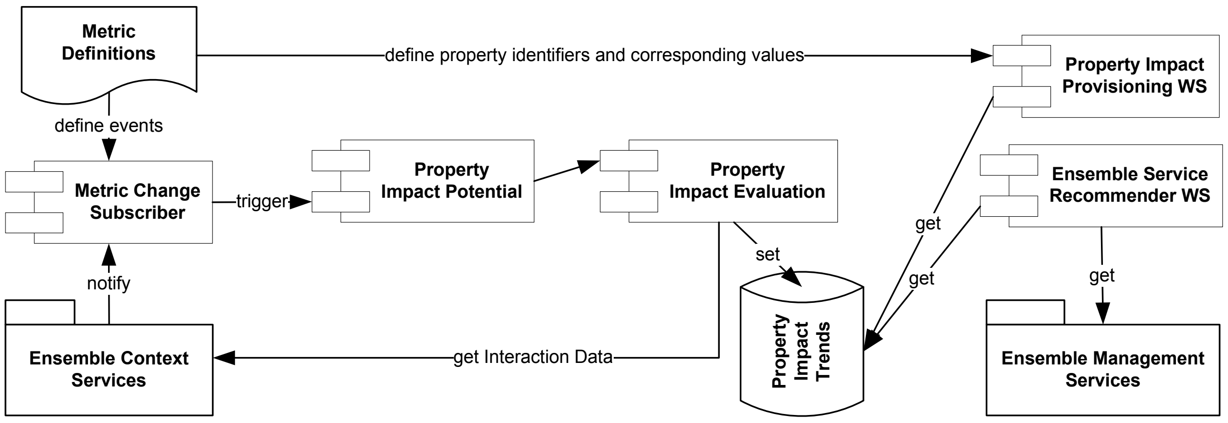

Metric Definitions
define property identifiers and corresponding values
Property Impact Provisioning WS
define events
Metric Change Subscriber
trigger
Property Impact Potential
Property Impact Evaluation
Ensemble Service Recommender WS
get
set
get
get
notify
Property Impact Trends
Ensemble Context Services
get Interaction Data
Ensemble Management Services

Figure 7.6: Property Impact Evaluation Subsystem.

7.4.2 Infrastructure Adaptation

Subscriptions to metric changes and capability changes provide the underlying data for infrastructure adaptation. Based on these events, the *Requirements Tracking* component applies the JBoss rule system (DROOLS [5]) to generate requirements. Listing 6.1 contains an example rule triggered upon changes in the location-centric property distribution entropy (ensemble location entropy: ELE).

Requirements that remain unfulfilled by the current ensemble configuration trigger a new instance of requirements matching. When requirements clustering provides sufficient

[5]`http://www.jboss.org/drools/`

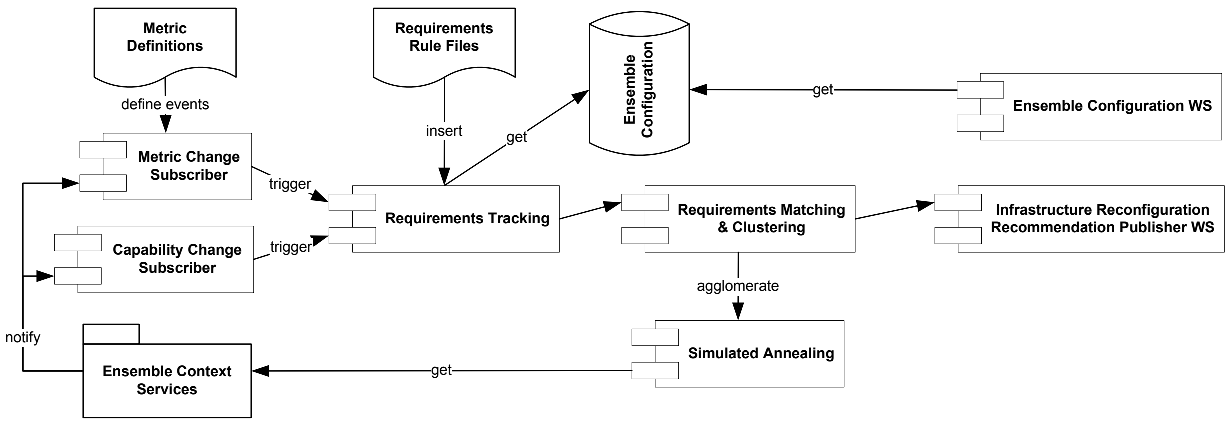


Figure 7.7: Infrastructure Adaptation Subsystem.

benefit, *Simulated Annealing* retrieves the interaction network from the ensemble context services. The *Infrastructure Reconfiguration Recommendation Publisher Service* provides a subscription interface for reconfiguration events. These events (Figure 7.8) contain a list of unfulfilled requirements (*RequirementRef*) and a set of recommended services, respectively persons. For each service, the event specifies the requirement fulfillment score (*EntityMembership*) in each cluster. For multi-cluster requirements, the event also includes the top composition (*Agglomerations*) with the best trade-off between interaction distance and joint requirements fulfillment (*Tradeoff*). For single-cluster results, all requirements remain in the same cluster (*Scope*).

Details on all current requirements and service configurations remain in the Ensemble Configuration database, accessible via the *Ensemble Configuration Service.* Figure 7.9 visualizes the ensemble configuration UML class diagram.

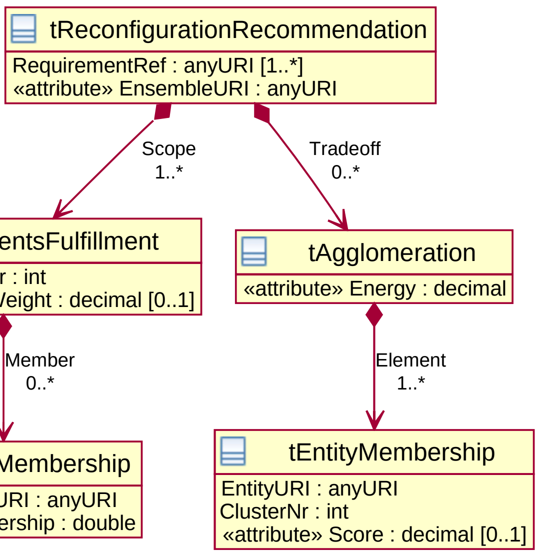

tReconfigurationRecommendation
RequirementRef : anyURI [1..*]
«attribute» EnsembleURI : anyURI
Scope
1..*
Tradeoff
0..*
tRequirementsFulfillment
«attribute» ClusterNr : int
«attribute» ClusterWeight : decimal [0..1]
tAgglomeration
«attribute» Energy : decimal
Member
0..*
Element
1..*
tClusterMembership
RequirementURI : anyURI
ClusterMembership : double
tEntityMembership
EntityURI : anyURI
ClusterNr : int
«attribute» Score : decimal [0..1]

Figure 7.8: Ensemble Reconfiguration Recommendation model UML class diagram.

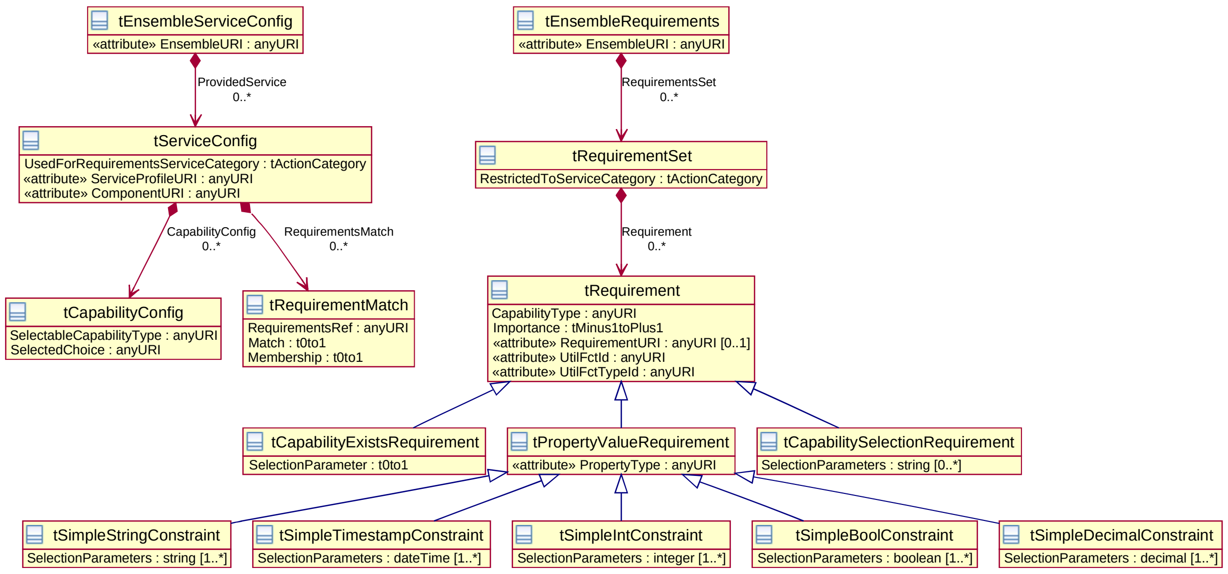

tEnsembleServiceConfig
«attribute» EnsembleURI : anyURI
ProvidedService
0..*
tServiceConfig
UsedForRequirementsServiceCategory : tActionCategory
«attribute» ServiceProfileURI : anyURI
«attribute» ComponentURI : anyURI
CapabilityConfig
0..*
RequirementsMatch
0..*
tCapabilityConfig
SelectableCapabilityType : anyURI
SelectedChoice : anyURI
tRequirementMatch
RequirementsRef : anyURI
Match : t0to1
Membership : t0to1
tEnsembleRequirements
«attribute» EnsembleURI : anyURI
RequirementsSet
0..*
tRequirementSet
RestrictedToServiceCategory : tActionCategory
Requirement
0..*
tRequirement
CapabilityType : anyURI
Importance : tMinus1toPlus1
«attribute» RequirementURI : anyURI [0..1]
«attribute» UtilFctId : anyURI
«attribute» UtilFctTypeId : anyURI
tCapabilityExistsRequirement
SelectionParameter : t0to1
tPropertyValueRequirement
«attribute» PropertyType : anyURI
tCapabilitySelectionRequirement
SelectionParameters : string [0..*]
tSimpleStringConstraint
SelectionParameters : string [1..*]
tSimpleTimestampConstraint
SelectionParameters : dateTime [1..*]
tSimpleIntConstraint
SelectionParameters : integer [1..*]
tSimpleBoolConstraint
SelectionParameters : boolean [1..*]
tSimpleDecimalConstraint
SelectionParameters : decimal [1..*]

Figure 7.9: Ensemble configuration model UML class diagram

Chapter 8

Conclusions

In this thesis we have investigated adaptation techniques for large-scale service ensembles. We highlighted in the problem statement that adaptation has to address the requirements of the overall ensemble, not merely the needs of individual humans or services. Our fundamental findings in this thesis are:

1. Ensemble adaptation combines suitable techniques at the level of service composition, service selection, and service behavior.
2. Adaptation techniques at the infrastructure level apply ensemble metrics to determine ensemble requirements. Matching requirements against deployed service capabilities reveals the demand for adaptation.
3. Efficient and effective composition trades off requirements fulfillment and composition costs. Composition costs derive from the interaction structure of ensemble entities.
4. Adaptation at the service selection level exploits service interaction patterns to determine influential service properties. These service properties determine suitable neighboring services to collaborate with.
5. Service behavior adaptation techniques rely on the most relevant context information. Context ranking requires distance metrics that describe the similarity between ensemble entities. These distance metrics must take into account the underlying ensemble structure.

These ensemble adaptation principles have been applied to our main contributions:

1. We developed a framework for ensemble infrastructure adaptation. Ensemble metrics, rules, requirements, and capabilities are the fundamental building blocks for the adaptation process.

2. Our biased fuzzy clustering algorithm groups requirements according to available service capabilities. More important requirements and better suited services yield more impact on the clustering result than less important requirements and less suited services.

3. We applied Simulated Annealing to achieve a trade-off between maximal requirements fulfillment and minimal composition costs. Experiments on real-world interaction and rating data from an online discussion forum demonstrate the benefit of requirements clustering and achieve a successful trade-off between fulfillment and costs.

4. The self-adjusting service recommendation algorithm analyzes the impact of service properties on interaction patterns. Properties that correspond to significant trends of service invocations enable service newcomers to communicate with the most suitable services. Experiments on a simulated service ensemble proof scalability and adaptiveness.

5. The Property Distribution Entropy measures the property distribution across services in the ensemble and highlights those of potential high impact.

6. Context distance metrics establish the relevant context for a given situation. We introduced a context-centric distance metric, and an interaction-centric distance metric. Experiments on real-world interaction data from an online discussion forum show that the interaction-centric metric is more sensitive to changes than the context-centric metric.

7. The context model describes persons, services, activities, artifacts, resources, and their interactions in a service ensemble.

This work has investigated the first set of adaptation techniques. Many new research questions arise from this thesis. Specifically, following research aspects provide interesting future work:
- The current ensemble context model lacks the explicit notion of temporal flows. Although actions and activities include timestamps and deadlines, the current algorithms do not exploit the temporal order of events beyond the simple aging function applied by the distance metric.
- The underlying models and algorithms analyze the interdependencies between directly interacting ensemble elements. We expect interesting results from extending this analysis to short chains of interacting ensemble entities.
- The service selection algorithm considers interaction data from all services as equally important. We envision the integration of additional aspects such as reputation and local policies.
- From an engineering point of view, interesting future work includes investigations on how to realize autonomic, decentralized ensemble composition enactment. Such work could pick up at the ensemble reconfiguration recommendations and advance the current framework. This would provide novel protocols to coordinate the adaptation plans.

References

Alava, M. J. and S. N. Dorogovtsev (2005). Complex networks created by aggregation. *Physical Review E 71*, 036107.

Albert, R. and A.-L. Barabasi (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics 74*, 47.

Albert, R., H. Jeong, and A.-L. Barabási (1999). The diameter of the world wide web. *CoRR cond-mat/9907038.*

Amundsen, S. L. and F. Eliassen (2008). A resource and context model for mobile middleware. *Personal Ubiquitous Comput. 12*(2), 143–153.

Anagnostopoulos, C., P. Mpougiouris, and S. Hadjiefthymiades (2005). Prediction intelligence in context-aware applications. In *MEM '05: Proceedings of the 6th international conference on Mobile data management*, New York, NY, USA, pp. 137–141. ACM Press.

Andreolini, M., S. Casolari, and M. Colajanni (2008, Oct.). Autonomic request management algorithms for geographically distributed internet-based systems. In *Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference on*, pp. 171–180.

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

Babaoglu, O., G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes (2006). Design patterns from biology for distributed computing. *ACM Trans. Auton. Adapt. Syst. 1*(1), 26–66.

Babaoglu, Ö., M. Jelasity, and A. Montresor (2004). Grassroots approach to self-management in large-scale distributed systems. In *UPP*, pp. 286–296.

Baldauf, M., S. Dustdar, and F. Rosenberg (2007). A Survey on Context-Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing 2*(4), 263–277.

Barabasi, A. and R. Albert (1999). Emergence of scaling in random networks. *Science 286*, 509–512.

Barabasi, A.-L. (2005). SOCIOLOGY: Network Theory-the Emergence of the Creative Enterprise. *Science 308*(5722), 639–641.

Bardram, J. E. (2005, July). Activity-Based Computing: Support for Mobility and Collaboration in Ubiquitous Computing. *Personal and Ubiquitous Computing 9*(5), 312–322.

Baresi, L., D. Bianchini, V. D. Antonellis, M. G. Fugini, B. Pernici, and P. Plebani (2003, September). Context-aware composition of e-services. In *TES*, pp. 28–41.

Bazire, M. and P. Brézillon (2005, July). Understanding context before using it. In *Modeling and Using Context: 5th International and Interdisciplinary Conference CONTEXT 2005*, pp. 29–41.

Belotti, R., C. Decurtins, M. Grossniklaus, M. C. Norrie, and A. Palinginis (2004, June). Modelling context for information environments. In *Ubiquitous Mobile Information and Collaboration Systems: Second CAiSE Workshop, UMICS 2004*, pp. 43–56.

Bezdek, J. C. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms.* Norwell, MA, USA: Kluwer Academic Publishers.

Biegel, G. and V. Cahill (2004, March). A framework for developing mobile, context-aware applications. In *Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. PerCom 2004*, pp. 361–365.

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

Bird, C., A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan (2006). Mining email social networks. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, New York, NY, USA, pp. 137–143. ACM Press.

Birukou, A., E. Blanzieri, V. D'Andrea, P. Giorgini, and N. Kokash (2007, Nov.-Dec.). Improving web service discovery with usage data. *Software, IEEE 24*(6), 47–54.

Bollobas, B. (2001). *Random Graphs.* Cambridge University Press.

Bottaro, A. and R. Hall (2007). *Dynamic Contextual Service Ranking*, Chapter Software Composition, pp. 129–143. Lecture Notes in Computer Science. Springer.

Brin, S. and L. Page (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems 30*(1-7), 107–117. Proceedings of the Seventh International World Wide Web Conference.

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

Buetow, K. H. (2005). Cyberinfrastructure: Empowering a "Third Way" in Biomedical Research. *Science 308*(5723), 821–824.

Casati, F., M. Castellanos, U. Dayal, and M.-C. Shan (2004). Probabilistic, context-sensitive, and goal-oriented service selection. In *ICSOC '04: Proceedings of the 2nd* *international conference on Service oriented computing*, New York, NY, USA, pp. 316–321. ACM.

Casella, G., C. Dorn, A. Polleres, and H. Yi (2008). Design and implementation of a context tunnelling extension - version 2. Technical report, inContext Consortium.

Chen, G. and D. Kotz (2002, June). Solar: An open platform for context-aware mobile applications. In *First International Conference on Pervasive Computing (Short Paper)*, pp. 41–47.

Chen, H., T. Finin, and A. Joshi (2003). An ontology for context-aware pervasive computing environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review 18*(3), 197–207.

Chen, Y., D. Bindel, H. H. Song, and R. H. Katz (2007). Algebra-based scalable overlay network monitoring: algorithms, evaluation, and applications. *IEEE/ACM Trans. Netw. 15*(5), 1084–1097.

Chintalapudi, K. and M. Kam (1998, May). A noise-resistant fuzzy c means algorithm for clustering. *Fuzzy Systems Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on 2*, 1458–1463.

Colman, A. (2007). Exogeneous management in autonomic service compositions. In *ICAS '07: Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, Washington, DC, USA, pp. 25. IEEE Computer Society.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2001). Dijkstra's algorithm. In *Introduction to Algorithms 2nd edition*, Chapter 24. MIT Press.

Costa, P. D., L. F. Pires, M. van Sinderen, and J. P. Filho (2004, April). Towards a service platform for mobile context-aware applications. In *1st International Workshop on Ubiquitous Computing - IWUC 2004*, pp. 48–61.

da Rocha, R. C. A. and M. Endler (2006). Context management in heterogeneous, evolving ubiquitous environments. *IEEE Distributed Systems Online 7*(4).

de Freitas and da Graca (2005). Toward a domain-independent semantic model for context-aware computing. pp. 10 pp.+.

Desai, N., P. Mazzoleni, and S. Tai (2007, Feb.). Service communities: A structuring mechanism for service-oriented business ecosystems. pp. 122–127.

Dey, A. and G. Abowd (2000, April). Towards a better understanding of context and context-awareness. In *Workshop on the What, Who, Where, When, and How of Context-Awareness at CHI 2000.*

Di Nitto, E., C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl (2008). A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engg. 15*(3-4), 313–341.

Dobson, S., S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli (2006). A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst. 1*(2), 223–259.

Dorn, C. and S. Dustdar (2007). Sharing hierarchical context for mobile web services. *Distributed and Parallel Databases 21*, 85–111.

Dorn, C., S. Dustdar, G. Giuliani, R. Gombotz, K. Ning, S. Perray, D. Schall, and M. Tilly (2007). *Encyclopedia of E-Collaboration (Edited by Ned Kock)*, Chapter Interaction and Context in Service-Oriented E-Collaboration Environments. Idea Group Reference.

Dorn, C., A. Polleres, and H. Yi (2007). Design and proof-of-concept implementation of the incontext context model version 1. Technical report, inContext Consortium.

Dorn, C., A. Polleres, and H. Yi (2008). Design and proof-of-concept implementation of the incontext context model version 2. Technical report, inContext Consortium.

Dorn, C., D. Schall, and S. Dustdar (2006, October). Granular context in collaborative mobile environments. In *OTM Workshops 2006, LNCS 4278*.

Dorn, C., D. Schall, and S. Dustdar (2008, October). Achieving team-awareness in scientific grid environments. In *7th International Conference on Grid and Cooperative Computing (GCC)*. IEEE Computer Society.

Dorn, C., D. Schall, and S. Dustdar (2009a). Context-aware adaptive service mashups. Submitted to IEEE Asia-Pacific Services Computing Conference (APSCC).

Dorn, C., D. Schall, and S. Dustdar (2009b, November). A model and algorithm for self-adaptation in service-oriented systems. In *IEEE European Conference on Web Services (ECOWS)*.

Dorn, C., D. Schall, R. Gombotz, and S. Dustdar (2007, June). A view-based analysis of distributed and mobile teams. In *5th International Workshop on Distributed and Mobile Collaboration (DMC 2007) at WETICE*. IEEE Computer Society.

Dorn, C., H.-L. Truong, and S. Dustdar (2008, June). Measuring and analyzing emerging properties for autonomic collaboration service adaptation. In *5th International Conference on Autonomic and Trusted Computing (ATC)*. Springer LNCS.

Dourish, P. (2004). What we talk about when we talk about context. *Personal Ubiquitous Computing 8*(1), 19–30.

Dujmovic, J. J. (2007). Continuous preference logic for system evaluation. In *IEEE Transactions on Fuzzy Systems*, Volume 15, pp. 1082–1099. IEEE Computer Society.

Dustdar, S. (2004). "Caramba Process-Aware Collaboration System Supporting Ad hoc and Collaborative Processes in Virtual Teams". *Distributed Parallel Databases 15*(1), 45–66.

Dustdar, S. and W. Schreiner (2005). A survey on web services composition. *Int. J. Web Grid Serv. 1*(1), 1–30.

Endres, C., A. Butz, and A. MacWilliams (2005, Jan). A survey of software infrastructures and frameworks for ubiquitous computing. *Mobile Information Systems 1*(1), 41–80.

Foster, I. (2005, May). Service-oriented science. *Science 208*(5723), 814–817.

Garlan, D., V. Poladian, B. R. Schmerl, and J. P. Sousa (2004). Task-based self-adaptation. In *WOSS*, pp. 54–57.

Gombotz, R., D. Schall, C. Dorn, and S. Dustdar (2006, November). Relevance-based context sharing through interaction patterns. In *2nd International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaborate-Com).*

Gómez, V., A. Kaltenbrunner, and V. López (2008). Statistical analysis of the social network and discussion threads in slashdot. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, New York, NY, USA, pp. 645–654. ACM.

Greenwood, D. and G. Rimassa (2007). Autonomic goal-oriented business process management. In *ICAS '07: Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, Washington, DC, USA, pp. 43. IEEE Computer Society.

Gu, T., H. K. Pung, and D. Q. Zhang (2004, May). A middleware for building context-aware mobile services. In *59th Vehicular Technology Conference, 2004. VTC 2004*, pp. 2656–2660.

Gu, T., H. K. Pung, and D. Q. Zhang (2005). A service-oriented middleware for building context-aware services. *J. Netw. Comput. Appl. 28*(1), 1–18.

Guimera, R., B. Uzzi, J. Spiro, and L. A. N. Amaral (2005). Team Assembly Mechanisms Determine Collaboration Network Structure and Team Performance. *Science 308*(5722), 697–702.

Hariri, S., B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu (2006). The autonomic computing paradigm. *Cluster Computing 9*(1), 5–17.

Haveliwala, T. (2003, July-Aug.). Topic-sensitive pagerank: a context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering 15*(4), 784–796.

He, J., A.-H. Tan, C.-L. Tan, and S.-Y. Sung (2003). *On Quantitative Evaluation of Clustering Systems.* Kluwer Academic Publishers.

Henricksen, K., J. Indulska, and A. Rakotonirainy (2001). Infrastructure for pervasive computing: Challenges. In *Workshop on Pervasive Computing INFORMATIK 01, Viena*, pp. 214–222.

Hey, T. and A. E. Trefethen (2005, May). Cyberinfrastructure for e-science. *Science 308*(5723), 817–821.

Hinze, A., R. Malik, and P. Malik (2005, August). Towards a tip 3.0 service-oriented architecture: Interaction design. Technical report, Department of Computer Science, University of Waikato.

Horn, P. (2001, October). Autonomic computing: Ibm's perspective on the state of information technology. Technical report, IBM Corporation.

Huebscher, M. C. and J. A. Mccann (2008, August). A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv. 40*(3), 1–28.

Hull, R. and J. Su (2005). Tools for composite web services: a short overview. *SIGMOD Rec. 34*(2), 86–95.

IBM (2004). Autonomic computing toolkit: Developer's guide. http://www-128.ibm.com/developerworks/autonomic/books/fpy0mst.htm.

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

inContext Consortium (2007a). Design and implementation of the pcsa - intermediary prototype. Technical report, inContext Consortium.

inContext Consortium (2007b). Software specification of the pcsa. Technical report, inContext Consortium.

inContext Consortium (2008). Design and implementation of the pcsa - final prototype. Technical report, inContext Consortium.

Jennings, B., S. van der Meer, S. Balasubramaniam, D. Botvich, M. Foghlu, W. Donnelly, and J. Strassner (2007, October). Towards autonomic management of communications networks. *Communications Magazine, IEEE 45*(10), 112–121.

Jones, B. F., S. Wuchty, and B. Uzzi (2008). Multi-University Research Teams: Shifting Impact, Geography, and Stratification in Science. *Science 322*(5905), 1259–1262.

Josang, A., R. Ismail, and C. Boyd (2007). A survey of trust and reputation systems for online service provision. *Decis. Support Syst. 43*(2), 618–644.

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

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). Optimization by simulated annealing. *Science, Number 4598, 13 May 1983 220, 4598*, 671–680.

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

Lee, S. H., P.-J. Kim, and H. Jeong (2006, January). Statistical properties of sampled networks. *Physical Review E 73*, 102–109.

Leski, J. (2003). Towards a robust fuzzy clustering. *Fuzzy Sets Syst. 137*(2), 215–233.

Lieberman, E., C. Hauert, and M. A. Nowak (2005, January). Evolutionary dynamics on graphs. *Nature 433*(7023), 312–316.

Little, M., E. Newcomer, and G. Pavlik (2004, November). *Web Service Context Specification (WS-Context).* OASIS.

Liu, N., J. Li, and N. Li (2008). A graph-segment-based unsupervised classification for multispectral remote sensing images. *WSEAS Trans. Info. Sci. and App. 5*(6), 929–938.

Maamar, Z., B. Benatallah, and W. Mansoor (2003, May). Service chart diagrams - description & application.

Maamar, Z., D. Benslimane, P. Thiran, C. Ghedira, S. Dustdar, and S. Sattanathan (2007). Towards a context-based multi-type policy approach for web services composition. *Data Knowl. Eng. 62*(2), 327–351.

Maamar, Z., S. Kouadri, and H. Yahyaoui (2004). A web services composition approach based on software agents and context. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, New York, NY, USA, pp. 1619–1623. ACM Press.

Macqueen, J. B. (1967). Some methods of classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297.

Manikrao, U. S. and T. V. Prabhakar (2005). Dynamic selection of web services with recommendation system. In *NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices*, Washington, DC, USA, pp. 117. IEEE Computer Society.

Marinescu, D., J. Morrison, C. Yu, C. Norvik, and H. Siegel (2008, Oct.). A self-organization model for complex computing and communication systems. In *Self-Adaptive and Self-Organizing Systems, 2008. SASO '08. Second IEEE International Conference on*, pp. 149–158.

Maximilien, E. and M. Singh (2005, June). Self-adjusting trust and selection for web services. pp. 385–386.

Maximilien, E. M. and M. P. Singh (2004). Toward autonomic web services trust and selection. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, New York, NY, USA, pp. 212–221. ACM.

McAuley, J. J., L. da Fontoura Costa, and T. S. Caetano (2007). Rich-club phenomenon across complex network hierarchies. *Applied Physics Letters 91*(8), 084103.

McBratney, A. and J. De Gruijter (1992). A continuum approach to soil classification by modified fuzzy k-means with extragrades. *Journal of Soil Science 43*, 159–175.

Mcculloh, I. A., J. Lospinoso, and K. Carley (2007). Social network probability mechanics. In *MATH'07: Proceedings of the 12th WSEAS International Conference on Applied Mathematics*, Stevens Point, Wisconsin, USA, pp. 319–323. World Scientific and Engineering Academy and Society (WSEAS).

Moody, P., D. Gruen, M. J. Muller, J. Tang, and T. P. Moran (2006). Business Activity Patterns: A New Model for Collaborative Business Applications.

Morse, D. R., S. Armstrong, and A. K. Dey (2000). The what, who, where, when, why and how of context-awareness. In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, New York, NY, USA, pp. 371–371. ACM Press.

Mrissa, M., C. Ghedira, D. Benslimane, Z. Maamar, F. Rosenberg, and S. Dustdar (2007). A context-based mediation approach to compose semantic web services. *ACM Trans. Internet Technol. 8*(1), 4.

Newman, M. E. J. (2002, Oct). Assortative mixing in networks. *Phys. Rev. Lett. 89*(20), 208701.

Newman, M. E. J. (2003, Feb). Mixing patterns in networks. *Phys. Rev. E 67*(2), 026126.

Ning, K., R. Gong, S. Decker, Y. Chen, and D. O'sullivan (23-26 July 2007). A context-aware resource recommendation system for business collaboration. *Int. Conf. on E-Commerce Technology and the 4th IEEE Int. Conf. on Enterprise Computing (CEC/EEE 2007).*, 457–460.

Parashar, M. and S. Hariri (2004). Autonomic computing: An overview. In *UPP*, pp. 257–269.

Quitadamo, R., F. Zambonelli, and G. Cabri (2007, May). The service ecosystem: Dynamic self-aggregation of pervasive communication services. In *Software Engineering for Pervasive Computing Applications, Systems, and Environments, 2007. SEPCASE '07. First International Workshop on*, pp. 1–10.

Ramparany, F., J. Euzenat, T. H. F. Broens, A. Bottaro, and R. Poortinga (2006, April). Context management and semantic modelling for ambient intelligence. Technical Report TR-CTIT-06-52, Enschede.

Reiff-Marganiec, S., H.-L. Truong, G. Casella, C. Dorn, S. Dustdar, and S. Moretzki (2008, December). The incontext pervasive collaboration services architecture. In *ServiceWave*. Springer.

Rosenberg, F., P. Leitner, A. Michlmayr, P. Celikovic, and S. Dustdar (2009, 29 2009-April 2). Towards composition as a service - a quality of service driven approach. pp. 1733–1740.

Saffre, F., R. Tateson, J. Halloy, M. Shackleton, and J. L. Deneubourg (2008). Aggregation Dynamics in Overlay Networks and Their Implications for Self-Organized Distributed Applications. *The Computer Journal*, bxn017.

Salehie, M. and L. Tahvildari (2009). Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst. 4*(2), 1–42.

Satyanarayanan, M. (2001, Aug). Pervasive computing: vision and challenges. *Personal Communications, IEEE 8*(4), 10–17.

Schall, D. (2009). *Human Interactions in Mixed Systems - Architecture, Protocols, and Algorithms.* PhD Thesis in Computer Science, Information Systems Institute – Vienna University of Technology (TU Wien), Distributed Systems Group, Argentinierstrasse 8184-1, 1040 Wien, Austria.

Schall, D., C. Dorn, S. Dustdar, and I. Dadduzio (2008, September). Viecar - enabling self-adaptive collaboration services. In *34th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE Computer Society.

Schall, D., C. Dorn, H.-L. Truong, and S. Dustdar (2008, December). On supporting the design of human-provided services in soa. In *4th International Workshop on Engineering Service-Oriented Applications: Analysis and Design (WESOA'08), Co-located with International Conference on Service Oriented Computing (ICSOC) 2008*. Springer.

Schall, D., R. Gombotz, C. Dorn, and S. Dustdar (2007, July). Human interactions in dynamic environments through mobile web services. In *International Conference on Web Services (ICWS)*. IEEE Computer Society.

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

Schilit, B., N. Adams, and R. Want (1994, Dec.). Context-aware computing applications. pp. 85–90.

Schmid, S., M. Sifalakis, and D. Hutchison (2006). Towards autonomic networks. In *Autonomic Networking*, pp. 1–11.

Serugendo, G. D. M., N. Foukia, S. Hassas, A. Karageorgos, S. K. Mostéfaoui, O. F. Rana, M. Ulieru, P. Valckenaers, and C. van Aart (2003). Self-organisation: Paradigms and applications. In *Engineering Self-Organising Systems*, pp. 1–19.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal 27*.

Sheng, Q. Z., B. Benatallah, Z. Maamar, M. Dumas, and A. H. H. Ngu (2004). Enabling personalized composition and adaptive provisioning of web services. In *CAiSE*, pp. 322–337.

Silva-Lepe, I., R. Subramanian, I. Rouvellou, T. Mikalsen, J. Diament, and A. Iyengar (2008). Soalive service catalog: A simplified approach to describing, discovering and composing situational enterprise services. In *ICSOC '08: Proceedings of the 6th International Conference on Service-Oriented Computing*, Berlin, Heidelberg, pp. 422–437. Springer-Verlag.

Skopik, F., D. Schall, and S. Dustdar (2009, August). The cycle of trust in mixed service-oriented systems.

Skopik, F., H.-L. Truong, and S. Dustdar (2009, June). Trust and reputation mining in professional virtual communities. In *9th International Conference on Web Engineering (ICWE)*. Springer.

Sørensen, C.-F., M. Wu, T. Sivaharan, G. S. Blair, P. Okanda, A. Friday, and H. Duran-Limon (2004, October). Context-aware middleware for applications in mobile ad hoc environments. In *ACM/IFIP/USENIX International Middleware conference 2nd*

Workshop on Middleware for Pervasive and Ad-Hoc Computing (online proceedings), Toronto, Canada.

Sousa, J. P., V. Poladian, D. Garlan, and B. R. Schmerl (2005). Capitalizing on awareness of user tasks for guiding self-adaptation. In *CAiSE Workshops (2)*, pp. 83–96.

Sterritt, R., M. D. Mulvenna, and A. Lawrynowicz (2004). Dynamic and contextualised behavioural knowledge in autonomic communications. In *Proceedings of the 1st Interational Workshop on Autonomic Communication, WAC*, pp. 217–228.

Sterritt, R., B. Smyth, and M. Bradley (2005). Pact: personal autonomic computing tools. In *EASe Workshop at ECBS 2005*, pp. 519–527.

Tai, S., N. Desai, and P. Mazzoleni (2006, Nov.). Service communities: Applications and middleware.

Valverde, S. and R. V. Solé (2006). Self-organization and hierarchy in open source social networks. Technical report, DELIS – Dynamically Evolving, Large-Scale Information Systems.

Černý, V. (1985, January). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications 45*(1), 41–51.

Vieira, V., P. A. Tedesco, and A. C. Salgado (2005). Towards an ontology for context representation in groupware. In *Proceedings of the International Workshop on Groupware, CRIWG*, pp. 367–375.

Vu, L.-H., M. Hauswirth, and K. Aberer (2005). Qos-based service selection and ranking with trust and reputation management. In *OTM Conferences (1)*, pp. 466–483.

Wang, X., T. Vitvar, M. Kerrigan, and I. Toma (2006). A qos-aware selection model for semantic web services. In *ICSOC*, pp. 390–401.

White, S. R., J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart (2004). An architectural approach to autonomic computing. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*, Washington, DC, USA, pp. 2–9. IEEE Computer Society.

Wolf, T. D. and T. Holvoet (2004). Emergence versus self-organisation: Different concepts but promising when combined. In *Engineering Self-Organising Systems*, pp. 1–15.

Wolf, T. D. and T. Holvoet (2005). Towards a methodology for engineering self-organising emergent systems. In *SOAS*, pp. 18–34.

Yang, Y., F. Mahon, M. H. Williams, and T. Pfeifer (2006). Context-aware dynamic personalised service re-composition in a pervasive service environment. In *UIC*, pp. 724–735.

Yu, T. and K.-J. Lin (2005, April). Adaptive algorithms for finding replacement services in autonomic distributed business processes. In *Autonomous Decentralized Systems, 2005. ISADS 2005. Proceedings*, pp. 427–434.

Zhang, J. and R. Figueiredo (2006, June). Autonomic feature selection for application classification. In *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, pp. 43–52.

Zhang, J.-S. and Y.-W. Leung (2004, April). Improved possibilistic c-means clustering algorithms. *Fuzzy Systems, IEEE Transactions on 12*(2), 209–217.

Appendix A

XML Schemata

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.in-context.eu/ns/activity"
    targetNamespace="http://www.in-context.eu/ns/activity" elementFormDefault="qualified"
    version="0.5">

<xs:element name="Activity" type="tActivity"/>
<xs:complexType name="tActivity">
    <xs:sequence>
        <xs:element name="ActivityURI" type="xs:anyURI" nillable="true"/>
        <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="Tags" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Start" type="xs:dateTime" minOccurs="0" maxOccurs="1"/>
        <xs:element name="End" type="xs:dateTime" minOccurs="0" maxOccurs="1"/>
        <xs:element name="Duration" type="xs:duration" minOccurs="0" maxOccurs="1"/>
        <xs:element name="Priority" type="xs:integer" minOccurs="0" maxOccurs="1"/>
        <xs:element name="ParentActivity" type="xs:anyURI" minOccurs="0" maxOccurs="1"/>
        <xs:element name="ChildActivities" type="xs:anyURI"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="RelatedActivities" type="xs:anyURI"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="LocationRefURI" type="xs:anyURI"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="EditArtifacts" type="tArtifact"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="ApplyResourceRefURIs" type="xs:anyURI"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="MemberInvolvements" type="tMemberInvolvement"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Requirements" type="tRequirement"
            minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Name" type="xs:string" use="optional"/>
    <xs:attribute name="Progress" type="xs:integer" use="optional"/>
</xs:complexType>

<xs:complexType name="tRequirement">
    <xs:sequence>
        <xs:element name="RoleRefURI" type="xs:anyURI" minOccurs="0" maxOccurs="1"/>
        <xs:element name="SkillRefURI" type="xs:anyURI" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="Required" type="xs:boolean" default="false"/>
</xs:complexType>

<xs:complexType name="tArtifact">
    <xs:sequence>
        <xs:element name="WrapsResourceURI" type="xs:anyURI" minOccurs="0" maxOccurs="1"/>
        <xs:element name="Name" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1"/>
        <xs:element name="ResourceType" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="tMemberInvolvement">
    <xs:sequence>
        <xs:element name="FoafAgentURI" type="xs:anyURI" minOccurs="1" maxOccurs="1"/>
        <xs:element name="Role" type="tInvolvementRole" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<xs:simpleType name="tInvolvementRole">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Creator"/>
        <xs:enumeration value="Observer"/>
        <xs:enumeration value="Contributor"/>
        <xs:enumeration value="Responsible"/>
        <xs:enumeration value="Supervisor"/>
    </xs:restriction>
</xs:simpleType>
</xs:schema>
```

Listing A.1: Activity Model XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.in-context.eu/ns/extFOAF"
    xmlns:res="http://www.in-context.eu/ns/resource"
    targetNamespace="http://www.in-context.eu/ns/extFOAF"
    elementFormDefault="qualified" version="0.1">
<xs:import namespace="http://www.in-context.eu/ns/resource"
    schemaLocation="resourcemodel.xsd"/>

<xs:element name="FoafAgent" type="tAgent"/>
<xs:complexType name="tAgent">
    <xs:sequence>
            <xs:element name="holdsAccount" type="tOnlineAccount"
                minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="mbox" type="xs:string" use="optional"/>
        <xs:attribute name="mbox_sha1sum" type="xs:string" use="optional"/>
        <xs:attribute name="gender" type="xs:string" use="optional"/>
        <xs:attribute name="URI" type="xs:anyURI" use="required"/>
</xs:complexType>

<xs:complexType name="tPerson">
    <xs:complexContent>
        <xs:extension base="tAgent">
            <xs:sequence>
                <xs:element name="knows" type="xs:anyURI"
                    minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        <xs:attribute name="firstName" type="xs:string" use="optional"/>
        <xs:attribute name="family_name" type="xs:string" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="tGroup">
    <xs:complexContent>
        <xs:extension base="tAgent">
            <xs:sequence>
                <xs:element name="member" type="tAgent"
                    minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="tOrganization">
    <xs:complexContent>
        <xs:extension base="tAgent">
            <xs:sequence/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="tService">
    <xs:complexContent>
        <xs:extension base="tAgent">
            <xs:sequence>
                <xs:element name="ProvidedBy" type="tAgent" minOccurs="0" maxOccurs="1"/>
                <xs:element name="ServesTo" type="tAgent"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="ServiceResource" type="res:tService"
                    minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="tOnlineAccount">
        <xs:attribute name="accountName" type="xs:string" use="optional"/>
        <xs:attribute name="accountServiceHomepage" type="xs:string" use="optional"/>
</xs:complexType>
</xs:schema>
```

Listing A.2: Entity Model XML Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns="http://www.in-context.eu/ns/action"
            targetNamespace="http://www.in-context.eu/ns/action"
            elementFormDefault="qualified"
            version="0.4">

<xs:element name="Action" type="tAction"/>
<xs:complexType name="tAction">
    <xs:sequence>
        <xs:element name="InvokedByServiceClient" type="xs:anyURI"
                        minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="ExecutedOnBehalfOfFoafAgent" type="xs:anyURI"
                        minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="AppliedResource" type="xs:anyURI"
                        minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="EditedArtifact" type="xs:anyURI"
                        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="ActionURI" type="xs:anyURI" use="required"/>
    <xs:attribute name="DescribesActivityURI" type="xs:anyURI" use="required"/>
    <xs:attribute name="Timestamp" type="xs:dateTime" use="required"/>
</xs:complexType>

<xs:element name="CoordinationAction" type="tCoordinationAction"/>
<xs:complexType name="tCoordinationAction">
    <xs:complexContent>
        <xs:extension base="tAction">
            <xs:sequence>
                <xs:element name="CoordinationType" type="tCoordinationType"
                        minOccurs="1" maxOccurs="1"/>
                <xs:element name="ToFoafAgent" type="xs:anyURI"
                        minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="CommunicationAction" type="tCommunicationAction"/>
<xs:complexType name="tCommunicationAction">
    <xs:complexContent>
        <xs:extension base="tAction">
            <xs:sequence>
                <xs:element name="NotificationType" type="tNotificationType"
                        minOccurs="0" maxOccurs="1"/>
                <xs:element name="ToFoafAgent" type="xs:anyURI"
                        minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="ExecutionAction" type="tExecutionAction"/>
<xs:complexType name="tExecutionAction">
    <xs:complexContent>
        <xs:extension base="tAction">
            <xs:sequence>
                <xs:element name="ServiceOperation" type="xs:anyURI"
                        minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="Description" type="xs:string"
                        minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Listing A.3: Action Model XML Schema Part 1

```
<xs:complexType name="tCoordinationType">
            <xs:choice>
                <xs:element name="ActivityChangeType" type="tActivityChangeType"
                    minOccurs="1" maxOccurs="1"/>
                <xs:element name="DelegateType" type="tDelegateType"
                    minOccurs="1" maxOccurs="1"/>
                <xs:element name="NotifyType" type="tNotifyType"
                    minOccurs="1" maxOccurs="1"/>
                <xs:element name="DelegateResponseType" type="tDelegateResponseType"
                    minOccurs="1" maxOccurs="1"/>
            </xs:choice>
</xs:complexType>

<xs:simpleType name="tActivityChangeType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Created"/>
        <xs:enumeration value="UpdatedData"/>
        <xs:enumeration value="AddedData"/>
        <xs:enumeration value="RemovedData"/>
        <xs:enumeration value="DeletedActivity"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="tDelegateType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Delegate"/>
        <xs:enumeration value="DelegateJoint"/>
        <xs:enumeration value="DelegateSplit"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="tNotifyType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="NotifyBegin"/>
        <xs:enumeration value="NotifyEnd"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="tDelegateResponseType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Deny"/>
        <xs:enumeration value="Accept"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="tNotificationType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Unknown"/>
        <xs:enumeration value="RequestTodo"/>
        <xs:enumeration value="RequestConfirmation"/>
        <xs:enumeration value="RequestDiscussion"/>
        <xs:enumeration value="RequestComment"/>
        <xs:enumeration value="RequestInformation"/>
    </xs:restriction>
</xs:simpleType>
</xs:schema>
```

Listing A.4: Action Model XML Schema Part 2

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.in-context.eu/ns/resource"
        xmlns:loc="http://www.in-context.eu/ns/location"
        targetNamespace="http://www.in-context.eu/ns/resource" elementFormDefault="qualified"
        version="0.12">
<xs:import namespace="http://www.in-context.eu/ns/location" schemaLocation="locationmodel.xsd"/>

<xs:element name="Resource" type="tResource"/>
<xs:complexType name="tResource">
    <xs:sequence minOccurs="1">
            <xs:element name="ResourceURI" type="xs:anyURI" nillable="true"/>
            <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1"/>
            <xs:element name="Tags" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="Name" type="xs:string" use="optional"/>
        <xs:attribute name="WSRCURI" type="xs:anyURI" use="optional"/>
</xs:complexType>

<xs:element name="SpatialResource" type="tSpatialResource"/>
<xs:complexType name="tSpatialResource">
    <xs:complexContent>
        <xs:extension base="tResource">
            <xs:sequence>
                <xs:element name="CurrentLocation" type="loc:tLocation"
                    minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="Host" type="tHost"/>
<xs:complexType name="tHost">
    <xs:complexContent>
        <xs:extension base="tSpatialResource">
            <xs:sequence>
                <xs:element name="IPaddress" type="xs:string"
                    minOccurs="1" maxOccurs="unbounded"/>
                <xs:element name="HostedDomain" type="xs:string"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="OpenPort" type="xs:string"
                    minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="MobileDevice" type="tMobileDevice"/>
<xs:complexType name="tMobileDevice">
    <xs:complexContent>
        <xs:extension base="tHost">
            <xs:sequence>
                <xs:element name="CommunicationChannel" type="tCommunicationChannel"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="DeviceCategory" type="tDeviceCategory"
                    minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Listing A.5: Resource Model XML Schema Part 1

```
<xs:simpleType name="tDeviceCategory">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Laptop"/>
        <xs:enumeration value="Smartphone"/>
        <xs:enumeration value="PDA"/>
        <xs:enumeration value="MobilePhone"/>
        <xs:enumeration value="Walkytalky"/>
        <xs:enumeration value="GPSNavigator"/>
        <xs:enumeration value="Walkytalky"/>
        <xs:enumeration value="Other"/>
    </xs:restriction>
</xs:simpleType>

<xs:element name="VirtualResource" type="tVirtualResource"/>
<xs:complexType name="tVirtualResource">
    <xs:complexContent>
        <xs:extension base="tResource">
            <xs:sequence>
                <xs:element name="ProvidedBy" type="tService" minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="Service" type="tService"/>
<xs:complexType name="tService">
    <xs:complexContent>
        <xs:extension base="tVirtualResource">
            <xs:sequence>
                <xs:element name="ServiceEndpoint" type="xs:string"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="WSDLDocumentURL" type="xs:string"
                    minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="DeployedOn" type="tHost"
                    minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="CommunicationChannel" type="tCommunicationChannel"/>
<xs:complexType name="tCommunicationChannel">
    <xs:complexContent>
        <xs:extension base="tVirtualResource">
            <xs:sequence>
                <xs:element name="Protocol" type="tCommProtocol"
                    minOccurs="0" maxOccurs="1"/>
                <xs:element name="OnlineStatus" type="tOnlineStatus"
                    minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="tCommProtocol">
    <xs:sequence>
            <xs:element name="Name" type="xs:string"
                minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="ProtocolURI" type="xs:anyURI" use="required"/>
</xs:complexType>
```

Listing A.6: Resource Model XML Schema Part 2

```xml
<xs:simpleType name="tOnlineStatus">
    <xs:restriction base="xs:string">
        <xs:enumeration value="ONLINE"/>
        <xs:enumeration value="OFFLINE"/>
        <xs:enumeration value="UNKNOWN"/>
        <xs:enumeration value="BUSY"/>
        <xs:enumeration value="AWAY"/>
    </xs:restriction>
</xs:simpleType>

<xs:element name="DocumentResource" type="tDocumentResource"/>
<xs:complexType name="tDocumentResource">
    <xs:complexContent>
        <xs:extension base="tVirtualResource">
            <xs:sequence>
                <xs:element name="MimeType" type="xs:string"
                    minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
</xs:schema>
```

Listing A.7: Resource Model XML Schema Part 3

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://xml.vitalab.tuwien.ac.at/ns/taaf/CapabilitiesMetaModel"
    xmlns:tns="http://xml.vitalab.tuwien.ac.at/ns/taaf/CapabilitiesMetaModel"
    elementFormDefault="qualified"
    version="0.1">

<element name="Profile" type="tns:tProfile"/>
<complexType name="tProfile">
        <sequence>
            <element name="WSDLlocation" type="anyURI" minOccurs="1" maxOccurs="1"/>
            <element name="Component" type="tns:tComponent"
                minOccurs="0" maxOccurs="unbounded"/>
            <element name="ServiceCategory" type="tns:tServiceCategory"
                minOccurs="1" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="ProfileId" type="anyURI" use="required"/>
</complexType>

<complexType name="tServiceCategory">
    <choice>
        <element name="ActionCategory" type="tns:tActionCategory" minOccurs="1" maxOccurs="1"/>
        <element name="AnyCategory" type="anyURI" minOccurs="1" maxOccurs="1"/>
    </choice>
    <attribute name="CategoryFit" type="tns:t0to1" use="required"/>
</complexType>

<complexType name="tComponent">
    <sequence>
        <element name="Capability" type="tns:tCapability"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="SelectableCapability" type="tns:tSelectableCapability"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="SupportedConfigurations" type="tns:tCombination"
            minOccurs="0" maxOccurs="1"/>
        <element name="SupportedTransitions" type="tns:tTransition"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="WSDLoperationScope" type="anyURI" minOccurs="0"
            maxOccurs="unbounded"/>
    </sequence>
    <attribute name="ComponentId" type="anyURI" use="required"/>
</complexType>

<complexType name="tCapability">
    <sequence>
        <element name="CapabilityId" type="anyURI" minOccurs="1" maxOccurs="1"/>
        <element name="Property" type="tns:tProperty"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="SubCapability" type="tns:tCapability"
            minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="FitnessLevel" type="tns:t0to1" use="required"/>
</complexType>

<complexType name="tProperty">
    <sequence></sequence>
    <attribute name="PropertyId" type="anyURI" use="required"/>
</complexType>

<element name="DefaultProperty" type="tns:tDefaultProperty"/>
<complexType name="tDefaultProperty">
    <complexContent>
        <extension base="tns:tProperty">
            <sequence>
                <element name="value" type="tns:tSimpleProperty"
                    minOccurs="1" maxOccurs="1"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

Listing A.8: Capability Model XML Schema Part 1

```
<complexType name="tSimpleProperty">
            <choice>
                <element name="intValue" type="int"
                    maxOccurs="unbounded" minOccurs="1"/>
                <element name="boolValue" type="boolean"
                    maxOccurs="unbounded" minOccurs="1"/>
                <element name="decValue" type="decimal"
                    maxOccurs="unbounded" minOccurs="1"/>
                <element name="timestampValue" type="dateTime"
                    maxOccurs="unbounded" minOccurs="1"/>
                <element name="strValue" type="string"
                    maxOccurs="unbounded" minOccurs="1"/>
            </choice>
</complexType>

<complexType name="pResourceSize">
    <complexContent>
        <extension base="tns:tProperty">
            <sequence>
                <element name="value" type="int"
                    maxOccurs="1" minOccurs="1"/>
                <element name="unit" type="tns:tUnit"
                    maxOccurs="1" minOccurs="1"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="tSelectableCapability">
    <complexContent>
        <extension base="tns:tCapability">
            <sequence>
                <element name="Alternative" type="tns:tCapability"
                    minOccurs="1" maxOccurs="unbounded"/>
            </sequence>
            <attribute name="RequiredSelection" type="boolean" use="required"/>
            <attribute name="DefaultSelection" type="anyURI" use="optional"/>
            </extension>
    </complexContent>
</complexType>

<complexType name="tCombination">
    <sequence>
        <choice minOccurs="0" maxOccurs="unbounded">
                <!-- the given strategy or strategy set -->
            <element name="Selection" type="anyURI" />
                <!-- select any combination of entries from the given strategy SETs! -->
            <element name="All" type="tns:tCombination" />
                <!-- select any entry from the given combinations of strategies -->
            <element name="OneOf" type="tns:tCombination" />
                <!--  may not select any entry from the given combinations of strategies ->
            <element name="NoneOf" type="tns:tCombination" />
        </choice>
    </sequence>
</complexType>

<complexType name="tTransition">
        <sequence>
            <element name="StartCombination" type="tns:tCombination"
                minOccurs="1" maxOccurs="unbounded"/>
            <element name="EndCombination" type="tns:tCombination"
                minOccurs="1" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="isPositive" type="boolean" use="required" />
</complexType>
```

Listing A.9: Capability Model XML Schema Part 2

```
<simpleType name="tUnit">
    <restriction base="string">
        <enumeration value="Byte"/>
        <enumeration value="kB"/>
        <enumeration value="mB"/>
        <enumeration value="gB"/>
        <enumeration value="tB"/>
    </restriction>
</simpleType>

<simpleType name="tActionCategory">
    <restriction base="string">
        <enumeration value="Communication"/>
        <enumeration value="Coordination"/>
        <enumeration value="Execution"/>
    </restriction>
</simpleType>

<simpleType name="t0to1">
    <restriction base="decimal">
        <minInclusive value="0"/>
        <maxInclusive value="1"/>
    </restriction>
</simpleType>
</schema>
```

Listing A.10: Capability Model XML Schema Part 3

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://xml.vitalab.tuwien.ac.at/ns/taaf/CapabilityChangeEvents"
    xmlns:tns="http://xml.vitalab.tuwien.ac.at/ns/taaf/CapabilityChangeEvents"
    elementFormDefault="qualified">

<element name="ServiceCapabilityChange" type="tns:tProfileChange"/>
<complexType name="tProfileChange">
    <sequence>
        <element name="NewComponent" type="anyURI" minOccurs="0" maxOccurs="unbounded"/>
        <element name="ChangedComponent" type="tns:tComponentChange"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="RemovedComponent" type="anyURI"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="ChangedServiceCategories" type="anyURI"
            minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="Source" type="anyURI" use="required"/>
</complexType>

<complexType name="tComponentChange">
    <sequence>
        <element name="NewCapability" type="anyURI" minOccurs="0" maxOccurs="unbounded"/>
        <element name="ChangedCapability" type="tns:tCapabilityChange"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="RemovedCapability" type="anyURI"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="SelectableCapability" type="tns:tSelectableCapabilityChange"
            minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="ComponentURI" type="anyURI" use="required"/>
</complexType>

<complexType name="tCapabilityChange">
    <sequence>
        <element name="NewProperty" type="anyURI" minOccurs="0" maxOccurs="unbounded"/>
        <element name="ChangedProperty" type="tns:tPropertyChange"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="RemovedProperty" type="anyURI" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="CapabilityURI" type="anyURI" use="required"/>
</complexType>

<complexType name="tSelectableCapabilityChange">
    <sequence>
        <element name="NewAlternative" type="anyURI" minOccurs="0" maxOccurs="unbounded"/>
        <element name="RemovedAlternative" type="anyURI"
            minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="CapabilityURI" type="anyURI" use="required"/>
</complexType>

<complexType name="tPropertyChange">
    <sequence>
        <any namespace="##other" processContents="lax" />
    </sequence>
    <attribute name="PropertyURI" type="anyURI" use="required"/>
</complexType>

<element name="ServiceCapabilityRepositoryChange" type="tns:tRepositoryChange"/>
<complexType name="tRepositoryChange">
    <sequence>
        <element name="NewServiceProfile" type="anyURI"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="ChangedServiceProfile" type="anyURI"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="RemovedServiceProfile" type="anyURI"
            minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="RepositoryURI" type="anyURI" use="required"/>
</complexType>
</schema>
```

Listing A.11: Capability Change Event Model XML Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://xml.vitalab.tuwien.ac.at/ns/taaf/EnsembleServiceConfig"
    xmlns:tns="http://xml.vitalab.tuwien.ac.at/ns/taaf/EnsembleServiceConfig"
    xmlns:cmm="http://xml.vitalab.tuwien.ac.at/ns/taaf/CapabilitiesMetaModel"
    elementFormDefault="qualified">

<import
    namespace="http://xml.vitalab.tuwien.ac.at/ns/taaf/CapabilitiesMetaModel"
    schemaLocation="CapabilitiesMetaModel.xsd"/>

<element name="EnsembleServiceConfig" type="tns:tEnsembleServiceConfig"></element>
<complexType name="tEnsembleServiceConfig">
    <sequence>
        <element name="ProvidedService" type="tns:tServiceConfig"
            minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="EnsembleURI" type="anyURI" use="required"/>
</complexType>

<complexType name="tServiceConfig">
    <sequence>
        <element name="CapabilityConfig" type="tns:tCapabilityConfig"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="RequirementsMatch" type="tns:tRequirementMatch"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="UsedForRequirementsServiceCategory" type="cmm:tActionCategory"
            minOccurs="1" maxOccurs="1"/>
    </sequence>
    <attribute name="ServiceProfileURI" type="anyURI" use="required"/>
    <attribute name="ComponentURI" type="anyURI" use="required"/>
</complexType>

<complexType name="tCapabilityConfig">
    <sequence>
        <element name="SelectableCapabilityType" type="anyURI"
            minOccurs="1" maxOccurs="1"/>
        <element name="SelectedChoice" type="anyURI"
            minOccurs="1" maxOccurs="1"/>
    </sequence>
</complexType>

<complexType name="tRequirementMatch">
    <sequence>
        <element name="RequirementsRef" type="anyURI" minOccurs="1" maxOccurs="1"/>
        <element name="Match" type="tns:t0to1" minOccurs="1" maxOccurs="1"/>
        <element name="Membership" type="tns:t0to1" minOccurs="1" maxOccurs="1"/>
    </sequence>
</complexType>

<element name="EnsembleRequirements" type="tns:tEnsembleRequirements"/>
<complexType name="tEnsembleRequirements">
    <sequence>
        <element name="RequirementsSet" type="tns:tRequirementSet"
            minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="EnsembleURI" type="anyURI" use="required"/>
</complexType>

<complexType name="tRequirementSet">
    <sequence>
        <element name="RestrictedToServiceCategory" type="cmm:tActionCategory"
            minOccurs="1" maxOccurs="1"/>
        <element name="Requirement" type="tns:tRequirement"
            minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

Listing A.12: Ensemble Service Config Model XML Schema Part 1

```
<complexType name="tRequirement">
    <sequence>
        <element name="CapabilityType" type="anyURI" minOccurs="1" maxOccurs="1"/>
        <element name="Importance" type="tns:tMinus1toPlus1" minOccurs="1" maxOccurs="1"/>
    </sequence>
    <attribute name="RequirementURI" type="anyURI"/>
    <attribute name="UtilFctId" type="anyURI" use="required"/>
    <attribute name="UtilFctTypeId" type="anyURI" use="required"/>
</complexType>

<complexType name="tCapabilityExistsRequirement">
    <complexContent>
        <extension base="tns:tRequirement">
            <sequence>
                <element name="SelectionParameter" type="tns:t0to1"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="tPropertyValueRequirement">
    <complexContent>
        <extension base="tns:tRequirement">
            <attribute name="PropertyType" type="anyURI" use="required"/>
        </extension>
    </complexContent>
</complexType>

<complexType name="tCapabilitySelectionRequirement">
    <complexContent>
        <extension base="tns:tRequirement">
            <sequence>
                <element name="SelectionParameters" type="string"
                    minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="tSimpleStringConstraint">
    <complexContent>
        <extension base="tns:tPropertyValueRequirement">
            <sequence>
                <element name="SelectionParameters" type="string"
                    maxOccurs="unbounded" minOccurs="1"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="tSimpleTimestampConstraint">
    <complexContent>
        <extension base="tns:tPropertyValueRequirement">
            <sequence>
                <element name="SelectionParameters" type="dateTime"
                    maxOccurs="unbounded" minOccurs="1"/>
            </sequence>
            </extension>
    </complexContent>
</complexType>
```

Listing A.13: Ensemble Service Config Model XML Schema Part 2

```
<complexType name="tSimpleIntConstraint">
    <complexContent>
        <extension base="tns:tPropertyValueRequirement">
            <sequence>
                <element name="SelectionParameters" type="integer"
                    maxOccurs="unbounded" minOccurs="1"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<complexType name="tSimpleBoolConstraint">
    <complexContent>
        <extension base="tns:tPropertyValueRequirement">
            <sequence>
                <element name="SelectionParameters" type="boolean"
                    maxOccurs="unbounded" minOccurs="1"/>
            </sequence>
            </extension>
    </complexContent>
</complexType>

<complexType name="tSimpleDecimalConstraint">
<complexContent>
        <extension base="tns:tPropertyValueRequirement">
            <sequence>
                <element name="SelectionParameters" type="decimal"
                    maxOccurs="unbounded" minOccurs="1"/>
            </sequence>
            </extension>
    </complexContent>
</complexType>


<complexType name="tFileSizeConstraint">
<complexContent>
        <extension base="tns:tPropertyValueRequirement">
            <sequence>
                <element name="SelectionParameters" type="cmm:pResourceSize"
                    maxOccurs="unbounded" minOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>

<simpleType name="tMinus1toPlus1">
    <restriction base="decimal">
        <minInclusive value="-1"/>
        <maxInclusive value="1"/>
    </restriction>
</simpleType>

<simpleType name="t0to1">
    <restriction base="decimal">
        <minInclusive value="0"/>
        <maxInclusive value="1"/>
    </restriction>
</simpleType>
</schema>
```

Listing A.14: Ensemble Service Config Model XML Schema Part 3