Western Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

7-28-2022 11:00 AM

Reputation-Based Trust Assessment of Transacting Service Components

Konstantinos Tsiounis, *The University of Western Ontario*

Supervisor: Kontogiannis, Kostas, *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Computer Science
© Konstantinos Tsiounis 2022

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Part of the Artificial Intelligence and Robotics Commons, Databases and Information Systems Commons, and the Software Engineering Commons

Recommended Citation
Tsiounis, Konstantinos, "Reputation-Based Trust Assessment of Transacting Service Components" (2022). *Electronic Thesis and Dissertation Repository*. 8675.
https://ir.lib.uwo.ca/etd/8675

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.

Abstract

As Service-Oriented Systems rely for their operation on many different, and most often, distributed software components, a key issue that emerges is how one component can trust the services offered by another component. Here, the concept of trust is considered in the context of reputation systems and is viewed as a meta-requirement, that is, the level of belief a service requestor has that a service provider will provide the service in a way that meets the requestor's expectations. We refer to the service offering components as service providers (SPs) and the service requesting components as service clients (SCs).

In this approach, we propose a technique that allows for evaluating and assigning trust to various service providers, by considering their ability to fulfill their clients' expectations or policies, and assigning and updating the reputation of other service clients, based on their capabilities as recommenders for the aforementioned service providers. Service clients request opinions from other service clients (i.e. recommenders), when looking for an appropriate service to use. Different sources of recommendation are considered and opinions are transformed into evidences to be used by our proposed ranking algorithm. After the utilization of a service, the requesting client updates the trust and distrust values for said service, as well as the reputation values for the service's recommenders. In this work, service clients and service providers are considered software applications that coordinate with each other, and may include micro-services, software agents, smart contracts or any other distributed inter-networked resource, without making any assumptions as to what a service client or a service provider component is, as long as it is a component that issues or responds to requests.

The proposed approach has been shown, through implementing a prototype and executing appropriate experiments, to be very stable in the presence of high percentages of malicious users. Even when the dishonest clients account for up to 80%, honest users are able to receive accurate recommendations and select services that are able to fulfill their requirements. The proposed framework outperforms other approaches, in scenarios where malicious users are involved, by up to 20%, especially for higher percentages of users overvaluing or undervaluing

services offered.

Our approach is, also, capable of quickly detecting deterioration of the QoS provided by a service provider, and supports the provision of incentives and compensations for allowing the selection of new and reformed services and ameliorating bad interactions caused by extenuating circumstances, respectively.

Keywords: Distributed Components, Service-oriented Computing, Trust, Reputation Systems, Multi-agent Systems, Metaverse

Lay Summary

As Service-Oriented Systems rely for their operation on many different, and most often, distributed software components, a key issue that emerges is how one component can trust the services offered by another component. Here, the concept of trust is considered in the context of reputation systems and is viewed as a meta-requirement, that is, the level of belief a service requestor has that a service provider will provide the service in a way that meets the requestor's expectations. We refer to the service offering components as service providers (SPs) and the service requesting components as service clients (SCs).

In this approach, we propose a technique that allows for evaluating and assigning trust to various service providers, by considering their ability to fulfill their clients' expectations or policies, and assigning and updating the reputation of other service clients, based on their capabilities as recommenders for the aforementioned service providers. In this work, service clients and service providers are considered software applications that coordinate with each other, and may include micro-services, software agents, smart contracts or any other distributed inter-networked resource, without making any assumptions as to what a service client or a service provider component is, as long as it is a component that issues or responds to requests.

The proposed approach has been shown, through implementing a prototype and executing appropriate experiments, to be very stable in the presence of high percentages of malicious users. Furthermore, the approach is capable of quickly detecting deterioration of the QoS provided by a service provider, and supports the provision of incentives and compensations for allowing the selection of new and reformed services and ameliorating bad interactions caused by extenuating circumstances, respectively.

Contents

Abstract ii

Lay Summary iv

List of Figures xi

1 Introduction 1

1.1 Problem Statement 4

1.2 Outline of Approach 4

1.3 Thesis Contributions 7

1.4 Thesis Outline 10

2 Related Work and Background 11

2.1 Reputation Systems 11

2.1.1 Reputation Systems Taxonomy 12

2.1.2 Commercial Reputation Systems 14

2.1.3 Academic Reputation Systems 15

2.1.4 Recommender Systems 23

2.1.5 Limitations of Related Work 25

2.2 Requirements modelling 25

2.3 Consensus in Distributed Systems 27

2.4 Reasoning Under Uncertainty 28

2.4.1 Dempster-Shaffer 28

2.4.2 Fuzzy Logic and Reasoning . . . 29
2.5 Background on Supporting Technologies . . . 30
2.5.1 Publish/Subscribe Systems . . . 30
2.5.2 Distributed Databases . . . 31

3 Modelling and Overall Process **33**
3.1 Modeling Concepts - Entities . . . 33
3.2 Modeling Concepts - Relations . . . 35
3.3 Modeling Concepts - Relation values . . . 38
3.3.1 Perceived Trust and Distrust per Interaction (OT and OD values) . . . 38
3.3.2 Cumulative Trust and Distrust (T and D values) . . . 39
3.3.3 Individual Reputation of a Recommender (R value) . . . 40
3.3.4 Overall Reputation of Recommenders (AR value) . . . 42
3.4 Sources of Recommendations . . . 43
3.5 Process Overview . . . 44
3.5.1 Process Outline . . . 44
3.5.2 Running Example . . . 46
3.6 Summary . . . 49

4 Trust and Reputation Evaluation Algorithms **51**
4.1 Evaluation of OT and OD values . . . 51
4.2 Evaluation of T and D values . . . 53
4.3 Evaluation of R value . . . 55
4.4 Timeouts . . . 61
4.5 Evaluation of AR value . . . 63
4.6 Selection of R values for AR calculation . . . 66
4.6.1 Adaptive Replacement Cache Policy . . . 67
4.6.2 ARC Adaptation . . . 70

4.7 Service Ranking 73
4.7.1 Selection of Recommenders 74
4.7.2 Aggregation of T and D values per recommender group 75
4.7.3 Dempster-Shafer 76
4.7.4 Incentives 78
4.7.5 Compensations 79
4.7.6 Running Example (Revisited) 80
4.8 Discussion on Self-Regulating Behaviour of Recommenders 88
4.9 Summary 90

5 System Architecture **92**
5.1 Centralized Architecture 92
5.1.1 Architecture Overview 92
5.1.2 Interface Specification 98
5.1.3 Process Sequence Diagrams 99
Recommendations and Ranking 100
Incentives, Utilization and Compensations 102
T/D and R values Update 104
Overall Reputation Value Update 107
Timeouts 108
5.2 Distributed Architecture 111
5.2.1 Architecture Overview 111
5.2.2 Interface Specification 113
5.2.3 Process Sequence Diagrams 114
Recommendations and Ranking 115
Incentives, Utilization and Compensations 115
T/D and R values Update 116
Overall Reputation Value Update 118

Timeouts 119
5.3 Blockchain Architecture 120
5.4 Messaging Protocols 121
5.4.1 Data Acquisition 121
5.4.2 Update values after interaction 123
5.4.3 Updating AR values 124
5.4.4 Publications of Events 125
5.4.5 Removing obsolete values 126
5.5 Summary 127

6 Implementation and Experiments 129

6.1 Overall Infrastructure 129
6.1.1 Framework implementation 129
6.1.2 Event Propagation 130
6.1.3 Database Utilization 130
6.2 Experiments 131
6.2.1 Simulator 131
6.2.2 Experimental Setup 133
Simulated Network Setup 133
Experiments Execution 134
Bootstrapping phase 134
6.2.3 Stability in the Presence of Malicious Components 135
6.2.4 Degrading Services 137
6.2.5 Connections to Recommenders and Service Providers 139
6.2.6 Sources of Recommendation 140
6.2.7 Effect of Considering Distrust Values 142
6.2.8 Incentives 144
6.2.9 Comparison 147

6.3 Summary 149

7 Conclusion and Future Work **151**

7.1 Summary of the Approach 151

7.2 Contributions 153

7.3 Discussion of Limitations 159

7.4 Future Work 161

Bibliography **163**

List of Figures

3.1 A network of service clients and providers 35
3.2 Interactions between client and provider components 37
3.3 Example network of recommenders and services 46

4.1 Example model for service evaluation 52
4.2 Offset Formula for R evaluation 58
4.3 General Structure of ARC lists 69
4.4 Example of lists for SC_3 70
4.5 Example network of recommenders and services 81

5.1 Component diagram for centralized architecture. 93
5.2 Centralized Architecture: Sequence diagram for ranking part. 100
5.3 Centralized Architecture: Sequence diagram for incentives, utilization and compensations part. 102
5.4 Centralized Architecture: Sequence diagram for updating *T/D* and *R* values after service utilization. 105
5.5 Centralized Architecture: Sequence diagram for deciding on important *R* values and updating *AR* values. 107
5.6 Centralized Architecture: Sequence diagram for removing obsolete *T/D* and *R* values. 109
5.7 Component diagram for distributed architecture. 112
5.8 Distributed Architecture: Sequence diagram for ranking part. 114

5.9 Distributed Architecture: Sequence diagram for incentives, utilization and compensations part. 116
5.10 Distributed Architecture: Sequence diagram for updating *T/D* and *R* values after service utilization. 117
5.11 Distributed Architecture: Sequence diagram for deciding on important *R* values and updating *AR* values. 118
5.12 Distributed Architecture: Sequence diagram for removing obsolete *T/D* and *R* values. 120

6.1 System stability vs Percentage of Malicious Components 135
6.2 Recognition of degrading service 137
6.3 Connections to recommenders and service providers 139
6.4 Recommendation Sources 141
6.5 Ranking Score for Different Observed Distrust 143
6.6 Services at bottom and out of ranking 146
6.7 Transaction success with respect to malicious users 149

Chapter 1

Introduction

Over the past few years, we witness an accelerated proliferation of virtual digital environments and applications that involve ad-hoc interactions between various software entities or digital models of physical entities (e.g. avatars). These applications and environments relate to a wide spectrum of domains, ranging from social commerce [94] to agent-computing [98], and metaverse [99].

In these types of applications, interacting entities can act as clients seeking access to resources and services, and/or as providers offering access to such resources and services. Examples of client entities include applications that seek an appropriate merchant in an e-commerce or a social-commerce site, applications that seek the most accurate information source to obtain reliable data from, and avatars that seek to obtain services and virtual objects from other avatars in *Massive Multiplayer Online* games (MMOs) or metaverse realities. Examples of providers include entities that generate or serve information the accuracy and provenance of which is important to their clients, applications that deliver trustworthy expert opinions, and applications that offer services required to meet their promised QoS levels.

Here, interactions between service clients and service providers most often are part of adaptive, context-aware, and elaborate processes that implement complex logic.

Furthermore, due to the high volume, velocity, and complexity of such interactions, it be-

comes evident that these cannot be coordinated manually, and applications must rely on elaborate frameworks that establish and maintain trust among the transacting parties. More specifically, a key challenge arises when it comes for a client to decide with which service provider to interact with or trust, with respect to the quality of service or provenance of data served.

The proposed framework can easily be incorporated and used in a variety of real-world scenarios. One of the most straightforward applications of our approach is e-commerce systems. Existing systems only provide functionality for providing reviews for services and products, without accounting for the reputation and nature of the reviewer. Such an approach leads to fake reviews [145] and inability to discern honest ones, as opposed to our framework that weighs opinions according to recommenders reputation. Another area where our framework would be a useful addition is virtual environments [144]. With major investment being put in different metaverse environments [146] and parabolic increase in users participating in them [147], a method for assigning trust to different entities in an automated yet efficient and reliable way is of paramount importance. Of course, virtual environments, also, include *Massively Multiplayer Online (MMO)* games, which account for a significant amount of interactions between digital entities. Finally, our proposed approach can be utilized in all cases where a network of trust is required when selecting an appropriate service provider, such as trading platforms, where the choice of recommendations from trading experts could be made on account of their assigned reputation.

It is important to note that the concept of *trust* has been considered in many different contexts, in different forms (i.e. static or dynamic) and, in different encodings (i.e. binary, or range values).

One area of *trust* is related to system security. Here *trust* may relate first, to the ability to access or deny service provision based on specific policies and authentication and authorization processes and second, to whether an application is considered safe, in the sense it is virus or malware free. In these scenarios, the concept of *trust* equates more to the ability to verify through specific processes that a component does not pose an immediate *threat*. The verifica-

tion dimension of trust can, also, revolve around the fulfillment of certain characteristics of a component, such as reliability, maintainability or dependability, as assessed by third-party entities. A noteworthy area that gained traction regarding *trust* and *security*, involves the concept of Zero-trust architectures [96] which is based on the assumption that nothing can be trusted and, thus, everything needs to be validated at every step utilizing specific processes and algorithms.

A second area of *trust* relates to consensus systems where the concept of trust is used as a mechanism between transacting nodes [97] to reach an agreement. The issue has been addressed as a potential problem in classic consensus protocols, such as the Byzantine Fault Tolerance [32] or the Paxos [31] algorithms, but trust has, also, been considered as a dynamic dimension that would allow one to specify new protocols, such as the Proof-of-Trust Consensus Protocol [100].

A third area of *trust* relates to reputation systems [19]. This thesis considers *trust* under this context and deals with its utilization as part of said systems. More specifically, in the case of reputation systems, trust of from one entity to another is established through the assessment and use of each entity's reputation. The result of that assessment can be provided in many forms and utilized by different underlying systems, usually specified by the approach's target use. In this thesis, we consider *trust* as a meta-requirement, that is, the level of belief a service client has that a service provider will provide the service in a way that meets the clients' expectations. Key requirements of reputation systems involve dealing with malicious components that infiltrate the system and provide false recommendations, as well as dynamically responding to changes in the behaviour of entities. The approach proposed in this thesis falls in this category of reputation systems and deals with the ability to assess *trust* and *distrust*, utilizing a dynamically formed and maintained network of transacting entities where the reputation of service providers and nodes who recommend them is constantly evaluated and updated.

1.1 Problem Statement

As mentioned earlier, high volume and high velocity virtual interactions are becoming increasingly common nowadays and, in most cases establishing, updating, and maintaining trust is not always possible either due to the sheer complexity of the interactions, or due to limited access to past behaviour entities have exhibited when interacted with other entities. This issue can be addressed using a trust-maintenance system that is based on the dynamic management of reputation models between clients, recommenders, and service providers.

The problem description pertaining this thesis can thus be formulated as follows:

Given a set of service providers $SP = \{SP_1, SP_2, \ldots, SP_n\}$, a set of service clients $SC = \{SC_1, SC_2, \ldots, SC_m\}$, sets of clients $R_i \subset SC$ and which act as recommenders for service provider SP_i, upon the request of a service client SC_j requesting the opinion service clients $SC_k \in R_i$ have about service provider SP_i, devise a framework where *a)* client SC_j is able to select an appropriate service provider SP_i based on the recommendations (i.e. evidences) it received from other clients $SC_k \in R_i$; *b)* upon using the service provider SP_i, client SC_j is able to assess and update the reputation of SC_k as a competent recommender; *c)* client SC_j is able to make itself available as a future recommender for service SP_i; *d)* the proposed approach is able to identify and isolate malicious recommenders within a practical period of time and; *e)* make use that the network of transacting entities is stable and its time and space performance are tractable.

1.2 Outline of Approach

In this thesis, we propose a technique that is based on a network of interacting entities acting as either service clients (SC) or service providers (SP).

Service providers (SP) are entities which act as proxies to actual services offered by different third party external systems. Therefore, each service provider node SP_i in the network is a proxy that corresponds to a single externally offered service and, if one wishes to provide

different services, multiple SP_i nodes must be modelled and registered.

Service clients (SC) correspond to entities that connect to the system to seek, select, and use services. Service clients can evaluate the positive and negative aspects of the individual interactions they had with service providers by applying and evaluating a model that denotes the expectations the specific client has on the service provider which is based on the service's QoS published characteristics. Such a model can be denoted using fuzzy rules, goal models, and i* models to name a few. Service clients can select a service not only based on their prior experience of using a service provider but also, based on recommendations they receive from other clients (i.e. the recommenders). In this respect, the clients can also evaluate and update the quality of the recommendations they have obtained from the recommenders by comparing the recommendation versus their own experience after using the recommended service. The values pertaining to the individual interactions are not valid forever. After a predetermined amount of time, they are considered stale and are excluded from consideration for the calculation of their cumulative equivalents. Compensations can be given by service providers which have failed for reasons beyond their ability to offer the expected QoS so they avoid negative evaluations, while incentives can be given by new or lower ranked service providers so that they can be selected again and enter the network.

The overall approach can be described in eight main steps.

Step 1. A Service Client SC_i seeks to select a service provider.

Step 2. The Service Client SC_i reaches out to three types of other clients who can act as recommenders of available service providers SP_k ... SP_j. The first category of recommenders, that SC_i reaches out to, includes the clients $SC_{experts}$, who have the highest overall reputation as recommenders. The second category of recommenders involves all clients $SC_{friends}$, from which SC_i has gotten good recommendations in the past. The third category of recommenders contains all clients SC_{fof} (friends-of-friends), who are known to be good recommenders by clients belonging in the previous category.

Step 3. Service client SC_i assesses the positive and negative aspects of the recommendations it receives and applies a reasoning algorithm based on the Dempster-Shaffer theory of evidence to compile a ranking of available services. Information about available incentives are, also, considered, allowing new or reformed services to be selected over established ones and actively participate in the network.

Step 4. Service client SC_i selects and uses service provider SP_j and, subsequently evaluates the positive and negative aspects, based on its experience from using SP_j, by applying an evaluation model (in our case goal models for *trust* and *distrust*).

Step 5. In case of a service provider SP_j which is highly ranked but fails to provide the expected QoS, compensations can be given to its clients so that it can avoid being ranked poorly. The service client can choose to accept the provided compensation, and update the observed *trust* and *distrust* values, or not.

Step 6. The observed *trust* and *distrust* values, updated after receiving a compensation or not, are used by service client SC_i to *a)* update its cumulative opinion (positive and negative aspects of it) about service provider SP_j and; *b)* update its opinion about the recommenders, whose recommendation it used, by comparing said *trust* and *distrust* values after using SP_j with the one proposed by said recommenders.

Step 7. The updated individual opinion of SC_i towards its recommenders in this interaction is used to update the overall reputation of those recommenders.

Step 8. Stale service provider scores and stale recommenders are removed from the network using a time-window approach, while important values for overall reputations are selected through a variation of the Adaptive Replacement Cache policy.

1.3 Thesis Contributions

The work presented in this thesis proposes several elements that contribute to the state-of-the-art in the area of reputation-based trust systems as follows:

C1: The proposed approach evaluates both trust and distrust values for service providers and reputation values for recommenders, in a dynamic way that, when combined with the proposed algorithms, leads to significant improvements in resiliency in the presence of malicious users, over existing approaches up to date. Most frameworks evaluate their performance for malicious users accounting for up to half of their respective users, in which case our proposed framework has a successful transaction rate that is 5%-15% higher than the ones we compared it to. Where our approach significantly outperforms related frameworks, however, is when higher percentages of dishonest users are involved. In such scenarios, the discrepancy in successful transaction rates can be up to 20%.

C2: The proposed approach builds upon a model of interactions that closely simulates trust behaviour between humans in social interactions, thus leading to a system that exhibits a highly stable and resilient behaviour in the presence of even radical oscillations of observed trust values. Historical values are also taken into consideration, both for positive and negative aspects of an interaction (i.e. trust and distrust values) clients have with service providers and recommenders, and novel algorithms are utilized for the update of said values, resulting in a framework that avoids sudden fluctuations resulting from interactions that are considered outliers or happenstance.

C3: The proposed approach yields an architecture which can be deployed both in a centralized or distributed manner. Existing approaches can only be deployed in one way or the other. Our approach can be utilized by an application or system operated and maintained by a central authority, thus requiring a centralized architecture, or can be part of a distributed network of agents, in which case a distributed variation can be deployed. Our approach can even be deployed as a number of smart contracts in any appropriate

blockchain, to be used in tandem with different functionalities offered in the corresponding metaverse.

C4: Another contribution pertains to dealing with the calculation of the global reputation of recommenders where we propose a novel method, based on the Adaptive Replacement Cache (ARC) protocol. In contrast to all other approaches, which either take every available node's opinion into consideration or filter only based on a specific threshold, the proposed method only considers reputation values that are deemed significant. Significance is decided on the merits of recency and frequency of opinion in question.

C5: The proposed approach allows for combination of independently evaluated positive and negative evidences (i.e. trust and distrust values) to provide a comprehensive ranking of available service providers. The vast majority of proposed reputation systems evaluate a service provider based on positive evidence or criteria, whereas a few of them focus on negative interactions. An even smaller number of approaches utilizes risk as an extra dimension to account for short term changes in behaviour. Our approach, however, provides the infrastructure for the calculation, propagation and utilization of positive and negative evidence that are considered distinct and separately evaluated. The framework is easily customizable and different algorithms for using available values for providing a service ranking can be incorporated.

C6: Even though previous approaches have dealt with the issue of data aging, a decay function or parameter, applied on the corresponding cumulative value, was utilized by most of them. Very few frameworks have handled the matter by discarding old values altogether and, in these cases a recalculation of the cumulative value is required. We propose a novel method for discarding obsolete reputation values without recalculating the corresponding cumulative values every time, thus improving the framework's performance and network impact. This allows us to get the benefits of the dynamic behaviour, offered through the aging of available recommendations, and, at the same time, being able to

accommodate a larger number of users and corresponding interactions.

C7: We propose a novel method based on *Incentives* to allow for new service providers, and service providers that have improved their performance, to be selected over already established service providers. In either case, the perceived behaviour of the incentive providing service is not consistent with the actual behaviour, either due to the service being new or because of underperforming in the past, and incentives are offered to allow said services to be selected, and eventually prove themselves, over their higher ranked counterparts. The proposed method facilitates the introduction of new services and the discovery of behaviour changes in old ones, thus, leading to system behaviour that is more dynamic.

C8: Finally, we propose the concept of *Compensations* to allow for historically well-behaving services to not be penalized for lower QoS due to extenuating circumstances. In case of a service that has performed as expected for a significant amount of time, occasional and short-lived drops in quality of service should not cause significant decreases in the level of trust put in their ability to perform up to standard, especially if the circumstances under which those bad interactions occurred are out of the provider's immediate control. To ameliorate such behaviour, we allow service providers to offer compensations to affected service clients. This approach allows for a system that remains stable when a service's behaviour hasn't actually degraded, but has only been temporarily altered due to external events.

The above lead to a framework for which evaluation results indicate that is highly stable and resilient in the presence of a high number of malicious users (i.e. malicious recommenders).

1.4 Thesis Outline

The thesis is organized as follows. Chapter 2 discusses related work on the topics of reputation systems, recommender systems and requirements and awareness modelling, as well as background information on Publish-Subscribe middleware frameworks, distributed databases, fuzzy reasoning and the Dempster-Shafer evidence theory. Chapter 3 presents the details on modelling choices, regarding entities, relations and recommendation sources, and provides an outline of the overall process followed by our approach when requesting recommendations and using an available service, as well as an example. Chapter 4 introduces and explains the algorithms created to accommodate the needs of the system, namely trust/distrust evaluation, reputation assessment, handling of obsolete values and ranking of services. The example included in the previous chapter is expanded to include the calculations required as part of the process. Chapter 5 presents the architecture of the proposed framework, providing several options in order to accommodate centralized or distributed needs. The process is revisited to indicate the utilization of the specific components and the messaging protocol used by the individual components to communicate is discussed. Details about the implementation of the prototype for the proposed approach, along with experiments executed to evaluate its performance and capabilities in the face of malicious users and dynamic service behaviour, are presented in Chapter 6. Finally, the thesis is concluded in Chapter 7, where a discussion about the behaviour of the proposed approach is included, some open issues are identified and pointers for further research are provided.

Chapter 2

Related Work and Background

In this chapter, we present the background and related work for all aspects of our presented approach, including reputation systems, modelling of requirements and incentives using goal models and consensus mechanisms in distributed systems.

2.1 Reputation Systems

Virtual interactions have been a commonplace over the last couple of years, with most of them occurring without prior real world relationship between the participating entities. Social media and the metaverse have exponentially increased the frequency of such interactions and have expanded the context and content of said interactions. Because of that, there is a pressing need for a way to evaluate an agent's credibility or trustworthiness.

Reputation systems are the solution to that problem and they have been researched for academic purposes and utilized for commercial reasons alike. The term incorporates all approaches, either algorithms or frameworks, that deal with estimating, updating, maintaining and propagating trust. Application include e-commerce, P2P file sharing networks, Web services, group decision making, e-governance etc.

2.1.1 Reputation Systems Taxonomy

Several surveys have been contacted regarding reputation systems [33, 43, 51, 42, 50, 38, 44, 39, 54], with each of them proposing their own taxonomy based on their main incentive. All of them, however, introduce several common dimensions that seem to be ubiquitous among reputation systems, whether commercial or academic.

First and foremost, reputation systems are divided into implicit and explicit. Platforms and systems that provide a reputation mechanism that is, however, not explicitly defined are considered *implicit reputation systems*. Social networks are a prime example of such systems, as a degree of trust can be inferred by observing connections to friends and assessing whether the friends are considered reputable or not. Another example would be Google's search engine, whose order of results indicates a difference in reputation. Of course, in implicit systems no mechanism for evaluating reputation and inferring trust is implemented, so further discussion about characteristics would be meaningless. *Explicit reputation systems*, on the other hand, provide a specific method of assigning reputation to different entities and are utilized in environments that rely on frequent interactions between those aforementioned entities.

In order to implement a reputation system, a number of choices have to be made regarding some of its characteristics. Some of them pertain to the evaluation of the trust and reputation values, whereas other deal with governance and deployment issues.

One of the main dimensions of reputation systems involves the type of historical reputation values maintained. Some frameworks choose to utilize *global* values for every entity in the system. Others prefer *personal or subjective* opinions that are pairwise values and correspond to the personal opinion an entity has about another. Both of those approaches have their advantages and disadvantages. *Global reputation values* allow for a more consistent view of the current state of affairs, but can be detrimental in case of malicious users. This approach also requires a central authority or additional processing power to be disseminated. *Personal reputation values* are much faster to compute, can be implemented in a distributed or decentralized environment and allow for a more personalized view of trustworthiness, but they are not always

consistent with the current state and usually require a longer time before identifying changes in behaviour of entities. The majority of reputation systems use global values when it comes to reputation of entities, which, although it introduces vulnerabilities to malicious attacks, scales better for larger systems.

The utilization of multiple contextual information or not is another question that comes up when creating a reputation system. Specific contextual attributes can provide additional meaning to occurring transactions. A small number of available systems have opted to take extra information into consideration when evaluating a user's reputation, such as ability to provide resources to the network [41] or after sales service and delivery time in case of buying goods [52]. Most of proposed systems, however, operate in a specific domain and maintain a single context throughout the system.

Another very important aspect of reputation systems involves the collection of information regarding previous interactions between entities. The most straightforward way of obtaining said information would be through *direct* observation. This includes both personal interactions and interactions that can be directly observed, as is the case in wireless networks for example. *Indirect* information could also be obtained by inquiring other entities and asking for the experience they have acquired through previous interactions. Lastly, a reputation system can use information that are *derived*, meaning that they weren't originally intended to be used by said system as a reputation source. GRAft [45, 46] is one of the few frameworks that utilize *derived* information though. Most of the systems use a combination of *direct* and *indirect* experience, with one complementing the other.

Representation of reputation values is another choice that reputation systems have to make. Several formats have been proposed over the years but some are more common than others. If an interaction or reputation of an entity is represented using boolean values, the representation is considered *binary*. Other options include *discrete* and *continuous* representation where values are discrete integers or floating point numbers respectively. Those are the three most commonly used formats, depending mostly on the domain the approach is applied on. Some

frameworks utilize some more obscure format that include a *string* or a *vector*, if the reputation value needs to stay decomposed into values coming from multiple sources.

Aggregation of said reputation values is also required in the context of reputation systems. Most of the available frameworks use some form of *counting* computation method, which includes summation of positive and negative feedbacks and averaging, weighted or not, depending on the format of reputation values. A few of the proposed approaches use *discrete* [53](involves converting discrete values into ratings using look-up tables), *probabilistic* [58, 59] (uses probability models to predict likelihood), *fuzzy* [11] (utilizes fuzzy logic), or *flow* [3, 5] (examines the flow of transitive trust) computation. The option of not aggregating is also explored [45, 46].

As far as implementation of the frameworks goes, a distinction needs to be drawn between different levels of reputation *presence*. In fully centralized systems, the underlying authority needs to be *online* for the reputation to be available, while in distributed or fully decentralized systems, authority presence can be *partial* [55, 56] or reputation values can be distributed even if authority is *offline*. Tying into the implementation of an approach and its governance authority, more specifically, options include *centralized* or *distributed* control.

Last but not least, reputation systems can be further categorized based on some characteristics of the values maintained. Reputation information can be either *atomistic* or *holistic*, depending on whether information is provided per transaction in a detailed manner or as a single, overall value respectively, and can be filtered or not. Furthermore, different approaches deal with data aging in different ways. Some provide *none*, while other *decay* older values as time passes [40] or allow for the *death* of old and obsolete reputation information [57].

2.1.2 Commercial Reputation Systems

One of the most well-known commercial reputation systems is provided by *Amazon* [47]. After completing the purchase of a product, a user can provide a review consisting of a numeric rank (5 stars or less) and a feedback message. Other users can rate those reviews as *helpful* or not

not helpful and they can, then, be ordered, based on those ratings. Review authors' reputation depends and fluctuates based on those ratings.

eBay is another example of commercial reputation systems and has been very well researched [35, 37, 34, 48, 49]. What sets this reputation system apart is its choice to request feedback from both parties participating in a transaction. After each transaction, each participant provides an overall discreet rating, some numerical ratings for different aspects of the transaction and a comment.

Stackoverflow also utilizes a reputation system to ensure the quality of questions and answers provided by different users. Asking and answering questions is encouraged, since those are the actions that earn reputation points. Different abilities are unlocked after a certain reputation level is achieved and points can be deducted for certain reasons (e.g. voting down). Reputation score is represented through a discrete value that corresponds to the reputation points the user has accumulated.

2.1.3 Academic Reputation Systems

One of the earliest approaches is presented in [3]. The *EigenTrust* system is based on the idea originally put forward by Google's *PageRank* algorithm [4]. In that system, trust is global and depends on the experiences of every other user involved. A global reputation value is computed and provided by the system for every node that participates in it. The algorithm assumes the presence of already trusted users and considers the ability to provide recommendations and the ability to provide a service to be one and the same. In our approach, recommenders and service providers are treated as separate entities with different reputation scores and different ways of updating them over time. Furthermore, when it comes to ranking services, only the opinions of recommenders with the highest scores are taken into consideration, thus allowing for faster computation and disregard of opinions coming from users that might be malicious. Finally, since the *EigenTrust* system depends on a set of pre-trusted users, proper execution relies on those users remaining honest. In our framework, however, all reputation values are subject to

change and noone is considered trustworthy forever.

A distributed approach based on *EigenTrust* is proposed in [5]. The framework is called *PowerTrust*, and it uses the realization that most feedbacks derive from a subset of the available users, consisting of a few "power" nodes. Local trust values are initially computed, and random walks are subsequently performed to aggregate and provide a global value. Upon identification of said power node, Markov chains and look-ahead random walks are utilized to update global values. Those global values are the only ones considered when requiring a rating of nodes, which contradicts our approach, where personal opinion of both recommender nodes and service providers is considered, as well as opinions of expert nodes.

The *XRep* system [6] proposes the utilization of both user-provided ratings and resource-based reputation to evaluate a user's trustworthiness. Cluster computing is performed to weigh different ratings and marginalize malicious users. *XRep* can only be utilized in types of networks and services where specific resources (i.e. files) are provided and assessment of both the resource and offerer is binary. Our framework allows for more granular rating of service providers and recommenders alike. Recommenders' reputation is, moreover, utilized to weigh opinions of recommenders regarding both offered services and other recommenders. Last but not least, *XRep* only deals with malicious users regarding their capacity to create multiple accounts in an attempt to game the polling system, whereas in our approach weighing of opinions is used to identify and isolate users that are maliciously underrating or overrating specific service providers.

Another approach to the issue of evaluating trust based on reputation is proposed in [8] through the *P-Grid* system. The premise, in which the system is based on, is that the majority of users are non-malicious.This assumption hinders it from accommodating higher percentages of malicious users, in contrast to our approach. Complaints are the only feedback taken into consideration when accounting for reputation within the system and trust is binary (i.e. user is trustworthy or not trustworthy) in this approach,as well. Again, complaints are all considered of equal importance since reputation is binary and opinions are not weighed based on the indi-

vidual user's granular reputation. Furthermore, complaint generation is not incentivized, under the proposed aggregation formula, since they harm one's trustworthiness within the system, whereas in our framework abundance of ratings is encouraged and allows for better assessment of users.

REGRET [7] employs a different approach compared to previous work. Reputation is partially inherited through the groups a user participates in. Personal opinions are also taken into consideration to compliment the reputation aspect of the framework. However, for group-inherited reputation to be used, *REGRET* assumes a sociogram (i.e. graph with social relations) is available to the user and a minimum number of interactions of the user in question has to have occurred within that group. There is, also, no distinction between recommenders and service providers and global reputation for the user proposing a target service is calculated every time. Furthermore, relations are non-directed, assuming that trust is identical for both parties, and they can be either cooperative or competitive. Finally, different reputation types are allowed based on the nature of the transaction, although they have to be specified system-wide, and individuals store separate values for each of those types. In our approach, we have opted to use goal models to evaluate different aspects of different transactions, thus allowing users to customize their preferences even further.

R^2Trust [9] is a reputation and risk based trust management framework that is fully distributed and is also applicable in the context of P2P networks. In this approach, both reputation and risk are taken into consideration when evaluating a user's trustworthiness. Transactions can have a number of distinct outcomes and direct opinions are expressed by counting the different outcomes of those transactions. All available recommendations from other peers are, also, taken into consideration and are weighed based on the recommender's reputation. Said reputation is adjusted based on the result of the interaction. The idea of time-dependent decay of values is also introduced in this approach, but no value is eventually considered obsolete, as opposed to what happens in our proposed framework. Moreover, recommender and provider reputation are considered one and the same, as is the case in most P2P systems. We, however,

opt for a distinction between the two as it allows for better assessment of opinions depending on the role we wish to evaluate.

PET [10] is another model based on trust and risk alike. While reputation is formed based on long-term behaviours, risk is utilized to mitigate short-term behaviours that might indicate error or maliciousness. The approach is specifically geared towards file sharing P2P systems, so interactions are scored using distinct values, and personal opinions are produced by classifying resources based on their category and affect both reputation and risk values. Recommendations are supplied by every peer in the network and only affect the reputation aspect and not the risk component of the system. Every available recommendations is considered and all of them bear the same significance, regardless of the recommender's reputation, thus failing to utilize trust to address the potential for malicious users in the system, which is a possibility we consider in our approach. Risk values are the ones used to detect malicious users,but are only based on direct observations and consider a smaller time window.

In *FuzzyTrust* [11], users maintain local trust values for providers with whom they have previously transacted. Global reputation is calculated by aggregating those local trust values using different weights based on a set of parameters. Fuzzy trust models are utilized to generate both the local trust values and the weights, corresponding to each recommender's opinion, and separate fuzzy rules have to be defined based on the specific domain. A threshold can be set regarding weights to specify which peers will be consulted, thus avoiding heavy network traffic in hot spots. Unlike our approach, there is no distinction between recommender and provider reputation. Moreover, the system has to run for multiple iterations, similar to *EigenTrust*[5], before the reputations converge to their final values. Lastly, the framework seems to detect the majority of malicious users in a few iterations, but has only been tested for low percentages of dishonest peers(i.e. 30%).

ARRep [12] is mainly geared towards fending off malicious attacks. It combines direct and recommender proposed trust and utilizes a transaction decay function to prioritize the requester's direct experience. Direct trust comes from counting the satisfactory and unsatisfac-

tory transaction the user has had with a particular provider in the past. Note that transaction result is binary, which is feasible as the system is geared towards P2P file sharing networks. Recommended trust occurs by considering all available values other peers have to offer. Each opinion is weighed based on the similarity and size of common set of opinions between requesting user and recommender. No historical reputation value for recommenders is maintained by the users, in contrast to our approach, and similarity is based on the opinions of recommenders for other users. Transaction existence is, also, verified, preventing recommenders from fraudulently rating users, with whom no transaction has occurred. Extra file nodes are required for the implementation of the framework and a Distributed Hash Table, similar to the one used in [8], to store past interactions.

Reputation frameworks have been utilized in the network domain too. The *CONFIDANT* protocol [13] requires all participating nodes to maintain a reputation value for each one of their neighbours in the network. Behaviour of neighbours is monitored, and if an event is deemed suspicious, the system updates the rating of the event's producer. The reputation value accounts for all previous experience and no interaction is ever deemed obsolete, thus not allowing for reformation of previously malicious nodes. If a node is considered malicious, nearby friend nodes are notified and the path containing said node are excluded from consideration for good. A prepopulated set of friends is assumed by the protocol, which hinders the adaptability of the protocol but is reasonable given the type of networks that are addressed. Incoming accusations are weighed based on the reputation of the submitting node and are only taken into account if they exceed a certain threshold, so as to avoid coincidences (i.e. network collisions).

Another approach dealing with selfish nodes in overlay networks is proposed in [14]. A reputation is maintained for each node based on willingness to fulfill network requests. Personal experience and peer testimonials are taken into consideration. Testimonials are weighed based on the peer's reputation. Like in other approaches, this one treats rating and fulfilling requests as one and the same when it comes to reputation, which hinders its ability to evaluate different type of behaviours in a more granular way.

Even though there is an abundance of approaches concerning reputation in peer-to-peer networks, very little research has been conducted on reputation of web services. In [15], web services interfaces are published in a centralized registry and users have the option of choosing an implementation for a particular interface based on its reputation. A web service's reputation is derived from historic values supplied by previous users of the service and ratings are provided as a set of attribute values related to the offered service. Different application users can provide thresholds and weights for certain attributes or risk tolerance, in which case service selection will differ from person to person. Attributes can even have different weights based on the domain the belong to, thus allowing for more personalized ratings. Every recommender, however, is considered to be honest and all opinions are of equal importance. In our framework, malicious users are considered a possibility, which is the reason why recommendations are weighed according to each recommender's reputation and only the best of the recommenders are consulted. Furthermore, even though an algorithm for aggregating ratings and damping old values is mentioned, the authors of [15] mostly focus on the conceptual model of the attributes that comprise available ratings.

In [16], authors propose an ontology model to discover the most trustworthy service depending on the consumer's preferences. Service providers register their implementations of service interfaces and service agents are created by the framework for each available interface. Providers advertise policies for their services and consumers specify QoS needs using a 3-tiered ontology proposed. Services are ranked and matched based on the policy, provided by the service as part of registering for the particular ontology they belong to, and the preferences provided by the consumer. Ranking is based solely on the values offered by the providers and no historic ratings by other users are considered.

The framework is extended in [17, 18] and sharing of ratings is introduced. Service implementations are still selected based on the providers' advertisements and the consumers' QoS requirements but the trust model has been extended to allow for reputation, deriving from opinions of other users, to be taken into consideration. Agents are connected to ontology-specific

agencies, where information about previous interactions are stored. Service quality reputation can either be a simple aggregation or capture relationships between different attributes. Furthermore, it is assumed that all participating entities act in an honest way, which is not always the case in real-world applications. Because of this assumption, everyone's opinion bears equal weight in the calculation of the service's reputation and there is no way for the proposed framework to identify dishonest users.

RATEWeb [19] is a framework proposed to establish trust among web services by assessing reputation. In this approach, a central authority is tasked with maintaining a list of the available services and the users that have previously interacted with each of them,which hinders the frameworks ability to be used in a distributed environment. When a user requires a specific service, a query is initially issued to discover the list of available services and ratings on those services are subsequently requested by users participating in that particular community. The requesting user then calculates the services' reputations, taking into account all available ratings and submitting users' reputation. The service ranking is performed using clustering of provided ratings. The service with the highest score is selected and the user rates and stores the rating after using it. That particular approach performs well as long as the majority of users are honest, as opposed to our approach, where higher percentages of malicious users can be accommodated since a subset of all recommendations is considered.

Trust and reputation have recently been considered as an important factor influencing decision making and consensus reaching in *group decision making* scenarios [132]. Large scale social networks can be used to acquire a social graph and trust can be utilized to spread experts opinions and provide recommendations, thus facilitating the negotiation between agents leading to mutually acceptable agreements [133, 134, 136]. In those systems, agents provide their opinions, said opinions are aggregated, general consensus is calculated and, then, feedback is provided to all or some members of the network to consider in the negotiation process, thus facilitating the reaching of consensus within the network.

In [133] and subsequently in [136], Wu et al. propose a pair of operators that allow the

propagation of trust between users of a social network, pointing, however, that those operators have inherent issues that may be responsible for the introduction of a specific vulnerability to the system.

When it comes to the aggregation of available opinions, fuzzy reasoning is the most commonly utilized approach. Yager et al. [137] proposed an operator called *Order Weighted Averaging(OWA)*, which allows weighting based on importance, as specified after ranking considering certain characteristics of the opinions. Wu et al. [134], on the other hand, propose that expert opinions are requested. Experts reputation is calculated based on the in-degree centrality of the node representing them in the social graph and their opinion is weighed based on that.

Other frameworks tackling the issue of group decision in social networks include the approach proposed by Wu et al. [135] and *DeciTrustNET* [138]. The former models trust relationships with linguistic information. A formalism for linguistic distribution is provided and a way to assign distributed linguistic trust, based on the set of actors, their attributes and the corresponding relations, is defined. Said trust values are aggregated using a weighted average operator and an attempt for reaching consensus is made. Feedback is provided to inconsistent users, who are forced to implement recommendation advices based on the cost they can afford, in order to reach the threshold value of group consensus degree.

DeciTrustNET [138], on the other hand, is a framework that takes into consideration users relationships that are part of the underlying social network and evaluates trust and resulting reputation based on social interaction characteristics and quality, user-provided feedback and evolution of each user's behaviour. Similarity of user profiles in the social network is considered, as well, to provide further insight on the trust a specific user puts in any other participant of the network.

Several theoretical approaches have, also, been proposed in the literature regarding the propagation of trust within an existing network of users. The main goal is to calculate trust values between a pair of agents that have no prior interaction with each other. Trust propagation models work under the premise that a user is more likely to trust the opinion of someone they

consider trustworthy. In Guha's et al. [20] model, trust may be propagated in one of the following four ways: a) if user i trusts user j and j trusts k, then i trusts k, b) if i trusts j, then j at least partially trusts i, c) if i_1 trusts j_1 and j_2 and i_2 trusts j_1, then i_2 may also trust j_2, d) if i trusts j, then i may trust k if j and k share trust in common agents. Those propagations can be combined in a single matrix and can be weighed differently according to preferences. The model can, also, be applied to propagation of distrust and the user can choose whether they want to utilize both or not.

In Bonacich and Lloyd [21], it is investigated how centrality is considered a status indicator in networks. Centrality can be calculated using eigenvector-like measures and can be utilized to discover nodes with high reputation within a network of users. This, however, requires knowledge of all available values in the system and is computationally expensive, which is why experts in our approach are chosen using a different algorithm that takes a subset of the recommendations into consideration.

2.1.4 Recommender Systems

Due to the fact that our approach is utilized to recommend appropriate services to requesting users, it can easily be misconstrued to be a recommender system. Recommender systems, however, are a subclass of information filtering systems and are mainly concerned with predicting the user's rating or choice of a service or product, based on past experience.

Since in a lot of platforms, and accompanying systems, the amount of information is too large for a user to perceive and assimilate, specific methods and approaches are required to filter said information and provide only the items that could be to the user's liking. The aforementioned filtering is performed by taking into consideration each item's and user's characteristics, as well as the relations between them. Several different methods have been proposed to accomplish the task of providing personalized services [60]. Most of them do not require the utilization of *trust* related values between users, with only a few taking into account information resulting from use of social networks or similarity metrics between different users.

One of the most commonly used methods is called *content-based* recommendation [61], where the description of different items is analyzed and the degree of similarity between items is considered in order to recommend them to the user. Another similar approach pertains to *knowledge-based* recommendations [62, 67, 68, 69], which involves maintaining a functional knowledge base that allows the system to infer relationships between a user's specific needs and the item to be recommended.

When it comes to consulting other users of the recommender system, three main ways of approaching the issue have been proposed. *Collaborative filtering-based* recommendations [63, 64, 65, 66] are based on calculating a similarity metric between users and recommending items that are liked by users with similar interests. Note that, no trust value or reputation is maintained or updated regarding other users. Similarity is calculated on a per need basis and the process ends with the recommendations of specific items. Social networks have, also, been utilized for *social network-based* recommendations[75, 76, 77], where trust to another user is actually considered when receiving recommendations. By exploiting the correlation between trust and user similarity, said systems provide recommendations by utilizing reputation as the weight in the rating prediction process. Those systems, however, use the underlying social network to discover assigned trust, but do not maintain or update the values based on recommendation result. The last technique involving other users is called *group* recommendation [78, 79, 80] and is utilized to produce a group of user suggestions when the participants cannot meet for negotiation, or their preferences are not entirely clear. These systems are more closely related to the *Group Decision Making (GDM)* systems discussed earlier.

Finally, some *computational intelligence-based* recommendation techniques have, also, been proposed, including Bayesian [71] and artificial neural networks [70], clustering [72], genetic algorithms [73] and fuzzy sets [74].

As it is evident, even though recommender systems propose a number of interesting techniques for acquiring and utilizing information, they make minimal use of *trust* in other users, and in the few where this dimension is explored, no adjustment or updates are performed on

the underlying social network.

2.1.5 Limitations of Related Work

First, the vast majority of the related work deals with P2P networks. In that set of scenarios, the result of an interaction is either successful or not (i.e. *binary*). Furthermore, in most approaches, *global* history is utilized and a few use *personal* history. Almost none, however, tries to combine the two in an attempt to gain access to the advantages offered by each of those views. Another issue is that all of the frameworks presented, especially the few concerned with trust in web services and other types of distributed components, utilize the totality of available opinions in the network and fail to accommodate larger percentages of malicious users. So, even if a user is identified as malicious, their recommendations are still taken into consideration, even though they are weighted accordingly. Another important issue to consider is the lack of approaches dealing with data aging by discarding values rather than using decay functions Finally, the frameworks that are general purpose, regarding the types of services they can consider (i.e. are not constrained to file exchange networks or telecommunication networks), require a *centralized* authority and repository for the aggregation of recommendations. Very few approaches offer a *distributed* alternative and most of them address a very specific subset of service type, usually file sharing P2P networks.

2.2 Requirements modelling

Goal-driven modelling has been heavily researched over the last couple of years to accommodate several aspects of the Requirements Engineering process in all kinds of systems. Several modelling notations, aiming at different RE activities and different focal points in the specification of goals, have been proposed, such as i* [83], Tropos [81] and KAOS [82].

A number of approach have, also, identified extensions to the basic goal models, allowing for the definition of tasks and associated actions [86] or awareness requirements.

More specifically, awareness requirements are requirements that refer to other requirements or domain assumptions and their success or failure at runtime. Those requirements can be represented using a formal language and can be monitored at runtime. Souza et al. [85] have proposed a framework where awareness requirements can be defined along regular requirements using goal models. A separate component is tasked with running feedback loops that monitor the state of different requirements based on produced events and evaluate success or failure of said goals and associated awareness requirements.

The model of monitoring at runtime and reconciling behaviour through goal modelling has also been researched by Dalpiaz et al. [29], where requirements for socio-technical systems are modelled using the Tropos modelling notation mentioned earlier. Events are monitored and certain plans are evaluated based on specified preconditions and postconditions to identify success or failure. In case of failure, alternative plans and goals are explored to compensate for the inability to complete original plan of action. This modelling approach can be utilized to discover failures within a process and propose compensations in other contexts as well.

The ability to reason on goal models, in order to verify requirements at runtime, is also of interest in the Requirements Engineering domain.

Most of the approaches utilize probabilistic reasoning with Liaskos et al. [84] further proposing the utilization of probabilistic effects of task outcomes. Instead of assuming that completion of a task brings the desired result with certainty, they propose that tasks have multiple intended and unintended outcomes, each with different likelihood. To accommodate for that probabilistic way of thinking, traditional goals were extended to include probability of success. They, also, put forward the idea of utility, which needs to be maximized as well. Reasoning takes into consideration both minimum success rate for each goal and maximum provided utility.

Chatzikonstantinou et al. [24], on the other hand, propose a framework that performs reasoning on fuzzy goal models. Truth values corresponding to the leaf nodes of the goal models are represented as fuzzy values and the propagation to the root nodes is performed in parallel

utilizing fuzzy logic, which allows more expressiveness and produces results that are easier to interpret.

2.3 Consensus in Distributed Systems

The problem of consensus is multi-faceted and is important to a variety of different applications and systems. Even though consensus is mostly related to distributed applications and their ability to agree on a specific value or set of values, it can, also, be viewed as an issue of resisting malicious behavior from a subset of the users participating in a specific service.

As described in [89], traditional consensus problems are divided into one of three categories and deal with algorithmic ways of ensuring agreement between different parties.

In the original consensus problem, each process proposes a single value, coming from a set of already specified values. All processes then communicate with one another and at the end three requirements must be met. Every process must have set its decision variable (*Termination*), all processes must bear the same decision value (*Agreement*), and if all correct processes proposed the same value, then all correct processes have chosen that value (*Integrity*). Variations of the last requirement may occur, based on the application.

A variation of the consensus problem is the Byzantine generals problem. This problem is almost the same as the previous one, with the only difference being that only one value is proposed by the leader. The same requirements must be met, again with the correct value being the one proposed by the leader. A minor difference, also, exist in the *Integrity* requirement, which can be reached only if the leader is correct.

Another variant is called *interactive consistency*, where processes agree on a vector of values, rather than a single value. Requirements are virtually the same, with *integrity* meaning that if a process p_i is correct, all correct processes decide on v_i as the value corresponding to p_i, in their vector.

Consensus problems can be regarded as specific to arbitrary process failures, they can be

useful for crash failures and other problems that require opinion agreement. Solutions to those problems can be applied to synchronous systems, but no guarantees can be provided for asynchronous ones [87]. Other methods, like failure detectors [88] or randomization, have been used to approximate consensus in such environments. The idea of soft consensus has also been put forward for non-critical systems, where reputation of agents can be used to decide on correctness of opinion.

2.4 Reasoning Under Uncertainty

Certain parts of the process performed by the proposed framework require reasoning under uncertainty. Ranking of services has to be able to take into account positive and negative evidence and produce a belief interval based on all the available information. Interactions with specific service must, also, be evaluated, but the information provided is not always distinct and precise. Sometimes the result of an interaction is not binary and a more elaborate representation and way of evaluating is required.

2.4.1 Dempster-Shaffer

The *Dempster-Shafer evidence theory* [22] is a general framework that allows its users to reason in cases where uncertainty is involved. This theory is a generalization of the *Bayesian theory of subjective probability* and allows for the combination of evidence from different sources and calculation of a degree of belief, taking all said evidence into consideration.

Two steps are involved in the calculation of the degree of belief in one or more propositions. Subjective probabilities are assigned to propositions that are considered answers to a specific question, and, if multiple evidence are available, they are combined to provide a single degree of belief.

Note here that, contrary to other probabilistic theories, probability values are assigned to sets of possible answers, rather than single ones. In case of evidence that point to a single

proposition, a unit set is created. Since the main objective of the theory is to be able to reason with uncertainty, degrees of belief are assigned to each member of the power set of the propositions and said belief is represented as an interval. The bound of this interval correspond to *belief* and *plausibility*. The first one takes into consideration the evidence in favour of a proposition, whereas the second one accounts for the probability remaining after accounting for evidence that are against a proposition (i.e. 1 minus belief to all subsets not containing proposition in question).

The theory, also, provides different ways of combining beliefs from different sources, based on assumptions made by the specific domain. Conflict between independent sources can be detected using the probability masses and a degree of ignorance can be specified, meaning that one does not have to provide probabilities that add up to 1.

The *Dempster-Shafer evidence theory*, as well as the available combination rules, have been researched [104, 105] and utilized in various domains, such as neural networks [106], classification algorithms [107], ad hoc networks [108] and geographical information systems [109].

2.4.2 Fuzzy Logic and Reasoning

In traditional *boolean logic* , the values are binary truth values (i.e. true or false) and the corresponding operations and rules are applied to such type of values and produce results of the same nature. Some times, however, this approach does not suffice when considering scenarios where uncertainty is involved. So, based on the observation that decisions in the real world are made in a non-binary way, *fuzzy logic* was introduced to deal with the need to represent vagueness. In that form of logic, the variables can bear a range of values (i.e. many-valued logic), usually a real number between 0 and 1, in an attempt to accommodate the presence of *partial truth*.

Several systems have been proposed for fuzzy logic, but the most well-known is the Mamdani rule-based approach [110]. According to that approach, a numerical input is assigned to a

set of predefined fuzzy sets with a degree of membership, fuzzy rules are, then, applied based on those degrees and the final output is transformed from a fuzzy truth value to a continuous variable, using any of the available defuzzification algorithms.

Another commonly used system is the one proposed by Tagano-Sugeno-Kang [111], where the final step of the process is integrated into the execution of the fuzzy rules.

Fuzzy reasoning has been researched in many different contexts, such as PID controllers [112], petri nets [115], goal models [24], neural networks [113], classification [114] and security [116].

2.5 Background on Supporting Technologies

To support the proposed architecture and corresponding implementation of our framework, utilization of some additional technologies and corresponding frameworks is required. More specifically, use of a Publish/Subscribe system is paramount for the dissemination of events regarding updates in trust, distrust and reputation values, and employment of a distributed database for saving the relation and corresponding values will allow for replication of certain parts of the proposed framework, thus improving its performance and potential throughput.

2.5.1 Publish/Subscribe Systems

Message brokers or *Publish/Subscribe middleware systems* are components used to facilitate communication between different applications or services. They are based on the *publish-subscribe pattern*, according to which publishers are not sending messages directly to interested parties, but rather categorize them, using any of the available methods. Subscribers, on the other hand, explicitly express interest in specific categories of messages and are notified when they become available. Neither party is aware of the other one, thus avoiding coupling between different applications.

Scalability is another advantage of such systems, since there is no need for connection

between publisher and subscriber and the brokers can be replicated and forward messages in a much more efficient way.

Several variations of *publish/subscribe* systems have been proposed, based on the subscription scheme, or message filtering. The two main methods utilized are *topic-based* and *content-based*. According to the first approach, messages are categorized based on *topics*. Those topics are logical abstractions resembling channels and they are, basically keywords that are easily understandable and enforceable across multiple platforms. To work around the static nature of this approach, however, the *content-based* method was proposed. In this approach, the subscription is based on the contents of the considered events. The properties of said events are taken into consideration when deciding whether a message is to be forwarded to a subscriber, thus allowing for a much more dynamic behaviour. Certain constrains can be specified on the event's properties and they can, also, be logically combined to provide additional power to the prospective subscriber. A third more obscure method has been proposed, as well, called *type-based publish/subscribe*. According to this approach, events are filtered based on their specific type and closer integration with the utilized language is achievable, since type safety can be ensured at compile-time.

The algorithms involved in Publish/Subscribe systems, as well as other methods to improve their performance, have been researched [117, 118, 121, 119]. Contexts under which the technology has been utilized include cyber-physical systems [120], IoT [122] and fog computing [123].

2.5.2 Distributed Databases

In case of distributed systems where information need to be shared among the different components, distributed databases can be used to solve that problem. A distributed database is, practically a database that maintains data in different physical locations. Of course, parts of a distributed database can be deployed in a single server, or computer stationed in the same location, but can, also, be dispersed over computers that are loosely coupled through any kind of

network. Note, also, that said computers can either be owned and operated by a single organization or can be independent and completely decentralized. Two processes that are distinctive in distributed databases are *replication* and *duplication*. *Replication* involves monitoring all parts for changes and updating each replica, once one is identified. *Duplication*, on the other hand, keeps all parts consistent with one predetermined *master* database. Further nuances are involved in the implementation of a distributed database, pertaining to required data security, consistency and integrity.

Distributed databases can be classified as homogeneous or heterogeneous, based on whether all parts use the same environment to run their database. Further categorization can, also, occur if one considers the *independence*, or *autonomy*, of each database, examining whether they function on their own or a master component is required for coordination.

Data fragmentation is offered by some distributed databases, allowing for faster data inquiries. In this scenarios, different parts of the data are stored in different sites and queries are performed in parallel. Different formats are, also, supported by distributed databases, with options varying from relational schemas to non-relational data models, such as *key-value*, *graph* and *wide-column*.

Distributed databases are harder to maintain due to their complexity, but offer a number of advantages compared to centralized databases. First of all, they are much easier to expand, since the data is already saved in multiple physical locations. Moreover, they are easily accessible from different networks, and they are more secure, since data are replicated and potential loss of a part doesn't mean loss of information.

Due to those merits, distributed databases have been extensively used and researched [124, 125, 126] for different contexts and scenarios. Those include fog computing [127], big data [129] medical databases [128], search engines [130] and, lately, blockchain distributed ledgers [131].

Chapter 3

Modelling and Overall Process

As is the case in all reputation systems, in our approach we have to model both the participating entities and the relations that occur between them. In this thesis we propose the concept of a *social graph* between entities. The *social graph* models the participating entities as nodes and the elapsed interactions as edges annotated by corresponding values. These model entities and relations between them are discussed in more detail below.

3.1 Modeling Concepts - Entities

Every entity in the model represents a software component in the proposed framework and is used to form the *social graph* of these interacting entities. These entities fall in one of the following categories:

- ***Service Provider:*** These are model entities that act as *service providers(SP)*. In practice, since manual logging and analysis of interactions clients have with actual services has been an issue in reputation systems, especially commercial ones, each SP is considered as a *proxy* to a corresponding *actual* service. These proxies (i.e. SPs) can also gather information about the client-actual service interaction, and allow for the automatic evaluation of interactions clients have by using the corresponding to this proxy *actual* service. In

this respect, the proposed model can be utilized without the need to add any specialised logging or monitoring capabilities to the back-end actual services. Note that even though in reality a service provider may offer multiple services, in the context of this model, each *SP* entity represents a single actual service offered by a provider. Different proxy nodes would need to be registered for each separate service.

- ***Service Client:*** These are model entities that act as *service clients(SC)*. Each of the nodes corresponds to a client participating in the framework and their functionality includes requesting information about the QoS of available service providers and ultimately utilizing one of those services.

- ***Recommender:*** These are model entities that act as *recommenders*. Recommenders are, also, *service clients(SC)* whose main responsibility is to provide recommendations at the request of their peers. They bear a score when their peers request them. They, also, bear a reputation score based on the accuracy of the recommendations thay have previously provided.

Note that, *service clients* and *recommenders* are represented by a single node in the social graph constructed as part of the proposed model (i.e. the act as clients or recommenders). Their behavior and functionality is based on the running scenario. If a node initiates a search for available services, it is considered a *service client*, whereas every other client, that is not a *service provider*, is a *recommender*.

All user components are assigned a unique identifier upon registering with the system and all relations and values pertaining to them use that identifier.

A user might occupy both the roles of a *service client* and a *service provider* in real life, but in the context of the proposed method those roles are distinct and separate nodes need to be created for each of them. Moreover, as mentioned earlier, multiple services need to be represented by multiple proxy nodes, even if they are offered by the same actual service provider.

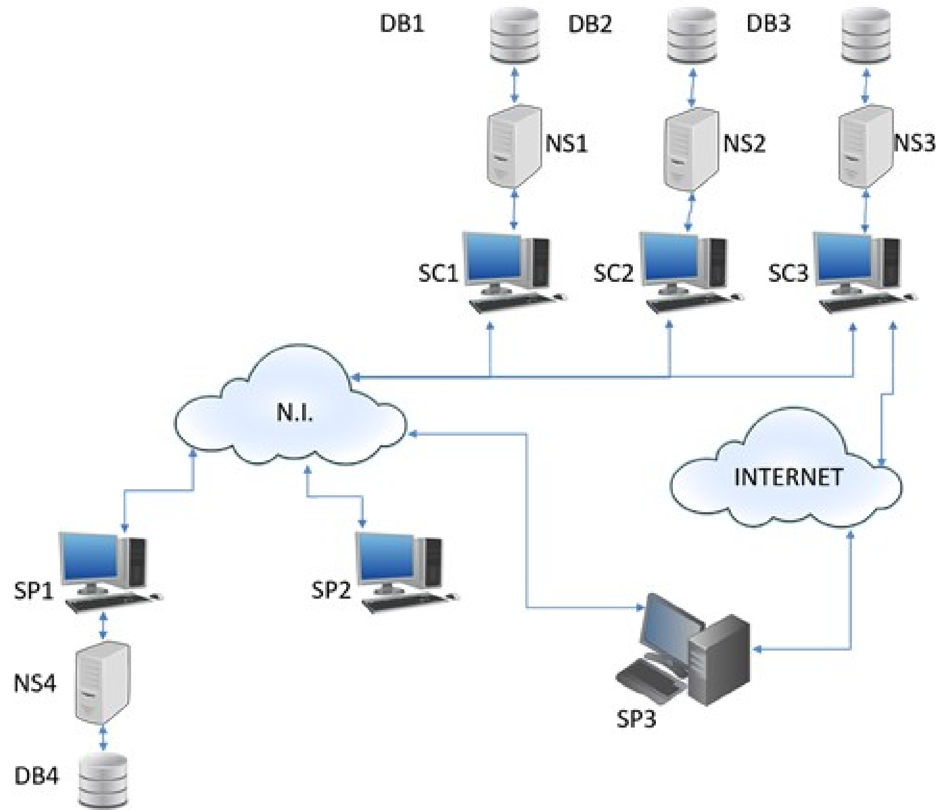

Figure 3.1: A network of service clients and providers

3.2 Modeling Concepts - Relations

Let us consider a deployment as the one depicted in Fig. 3.1. In this deployment, service clients SC_1, SC_2, and SC_3 require services offered by service providers SP_1, SP_2, and SP_3. The service client applications may execute their own programs NS1-NS3, and may have access to their own data stores DB1-DB3, in case of a decentralized approach. The same holds for the service providers SP_1-SP_3 which may execute their own programs NS4, and may have access to internal data stores DB4. Service clients and service providers can be connected via a private network infrastructure (e.g. N.I) or a public network infrastructure (e.g. the internet). In case of a centralized approach, the relations remain the same, with the only difference being that the programs are executed by a central authority and all information are stored in a server owned by said authority.

The proposed method is based on *a)* a *service client* (say SC_4) issuing a request to other

service clients, who are considered *recommenders* in this iteration (say SC_2, and SC_3), in order to obtain recommendations for *services providers* offering a given type of service *b)* once the service client SC_4 uses a service (say SP_1) as a result of such recommendations, *i)* assigns a metric value indicating the measure of belief (trust) of how well the *service provider* was perceived to have met the client's expectations (i.e. in this case how well the service SP_1 met the SC_4's expectations); ii) assigns a metric value indicating the measure of belief of how the *service provider* was perceived to *have not met* the client's expectations (i.e. SC_4's disbelief on SP_1); iii) assigns a metric value indicating how good the recommenders SC_2 and SC_3 were given that SC_4 now has a first-hand experience using SP_1. We assume that service clients SC_2 and SC_3 have already used the service SP_1 and therefore are able to provide their recommendation. The aforementioned interactions between SC_4, SC_2, SC_3 (i.e. requests for recommendations, responses to requests for recommendations), and between SC_4 and SP_1 (i.e. service invocations, service responses) create the social graph previously mentioned, where nodes are either clients (and recommenders) or service providers, and edges are these types of interactions. An example of such a graph can be seen in Fig. 3.2. In this graph it is also assumed that clients SC_1 and SC_5 have already used the service provider SP_1 and already have their opinions (denoted as T) for it, the client SC_3 has alteady an opinion about SC_2 to be a good recommender, and that client SC_2 has already provided in the past recommendations to SC_2 and SC_3.

In this respect, a level of belief/disbelief a service client has that a service provider will indeed deliver the QoS the client expects, is assigned to each service, and a level of reputation is assigned to each client for its ability to provide good (i.e. trustworthy) recommendations. The proposed method utilizes the following modelling relations and corresponding values:

***OT* Relation**: This relation OT_p^s denotes the existence of a trust (i.e belief that the service met expectations) opinion a client p has on a service s, after a specific interaction. Its value $OT(p, s)$ represents the level to which a client p perceives a service s to be trustworthy, as it met its QoS expectations and criteria (i.e. how satisfied p is by the services s has provided),

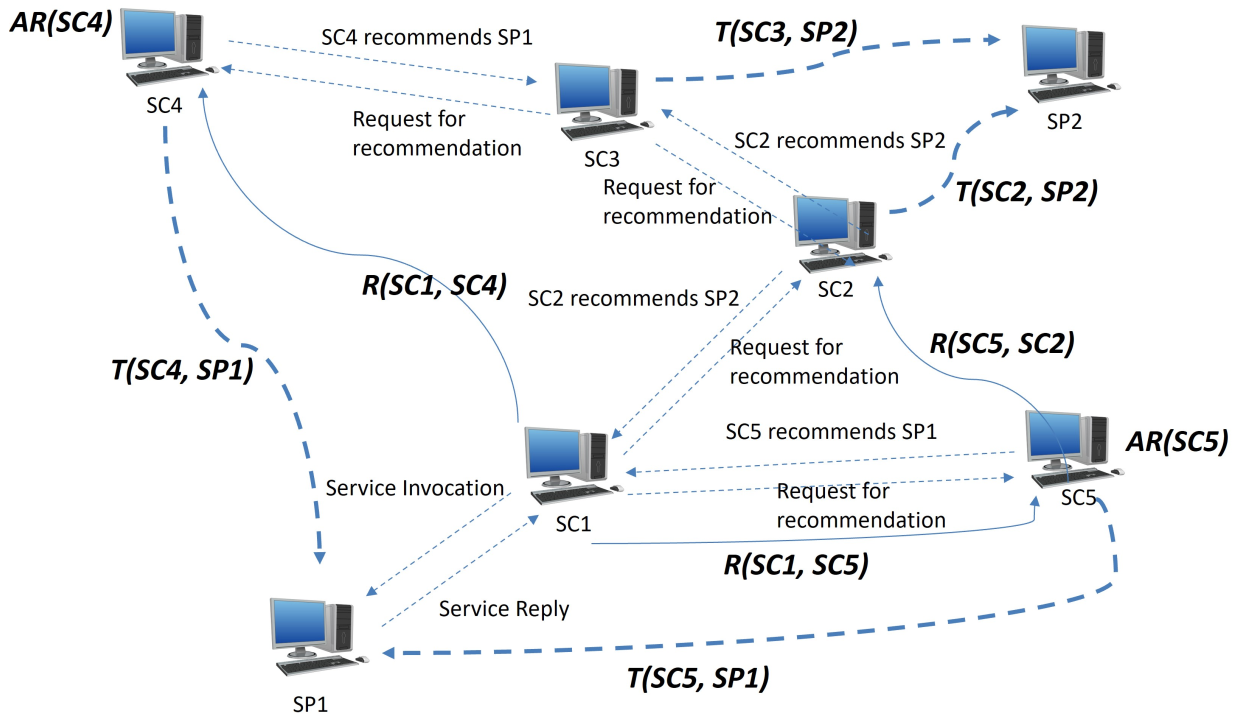

Figure 3.2: Interactions between client and provider components

after each single interaction. This kind of relation is not visible in the social graph, but its value is used to compute the corresponding value of the T relation (see below).

***OD* Relation**: This relation OD_p^s denotes the existence of a distrust (i.e belief that the service did not meet expectations) opinion a client p has about a service s, after an interaction with it. Its value $OD(p,s)$ represents the level a client p perceives that a service s may not be trustworthy (e.g. s obtains its data from an unknown source) after each single interaction. This kind of relation is not visible in the social graph, but its value is used to compute the corresponding value of the D relation (see below).

***T* Relation**: This relation T_p^s denotes the existence of an overall trust opinion a client p has for a service provider s and indicates one or more interactions with this service provider. Its value $T(p,s)$ represents the level of overall trust assigned by a client p on a service provider s, based on the history of interactions p has had with that s (see OT relations above).

***D* Relation**: This relation D_p^s denotes the existence of an overall distrust opinion a client p

has for a service provider s and indicates one or more interactions with this service provider. Its value $D(p, s)$ represents the level of overall distrust assigned by a client p on a service provider s, based on the history of interactions p has had with that s (see OD relations above).

***R* Relation**: This relation R_p^r denotes that a client p has an opinion about another client r, regarding whether r is a good recommender. This relation indicates that a client p has used services in the past, following recommendations provided by client r, and has formulated an opinion about it. The relation's corresponding value $R(p, r)$ represents the reputation of r according to p (i.e. the belief client p has that client r is historically a good recommender).

***AR* Relation**: In contrast to the other binary relations presented above, the relation AR^r is unary. It denotes the existence of a consolidated opinion by all other clients, regarding whether client r is a good recommender or not. Its value $AR(r)$ represents the level to which r is considered to be a good recommender, or not, and is computed by taking into account the quality and accuracy of all the recommendations r has given to each of the other clients, so far.

3.3 Modeling Concepts - Relation values

In this section, we discuss the different relations and corresponding values that are used for modelling *a)* the perceived QoS a service client (or recommender) experiences after using a service provider (see *T(p,s)* and *D(p,s)* values), and *b)* the reputation of a service client acting as a recommender, either globally (see *AR(r)* values) or as far as another service client is concerned (see *R(p,r)* values).

3.3.1 Perceived Trust and Distrust per Interaction (OT and OD values)

The *OT(p,s)* and *OD(p,s)* values denote how a service client assesses its level of satisfaction it experiences after a specific interaction with a particular service. More specifically, the *OT(p,s)* value denotes the measure of how much the service client p believes the service s met its expectations (e.g. QoS, constraints, requirements etc.). Similarly, the *OD(p,s)* value denotes

the measure of how much the client *p* believes that the service *s* engaged in behaviours that may signify distrust towards the service. For example, a client experiencing that a service provider is using data from a non-authorized source would increase the *OD(p,s)* value the client assigns to the service for this particular interaction. Clients can set specific requirements models pertaining both to trust and distrust of a service. The use of these pair-values (trust *OT(p,s)* and, distrust *OD(p,s)*) allows for a more flexible model where a client can provide at the same time the positive and negative sentiment related to its observations when using a service.

The value of *OT(p,s)* ranges from 0.0 to 1.0 (with 0.0 meaning that the service *s* did not meet its expectations, and 1.0 meaning that the service fully met its expectations). For example, an *OT(p,s)* value of 0.8 indicates the level to which client *p* believes that service *s* met its expectations in a satisfactory degree.

The value of *OD(p,s)* also ranges from 0.0 to 1.0 (with 0.0 meaning that client *p* agrees that service *s* did not engage in any behaviour increasing distrust, and 1.0 meaning that the client observed service behaviour indicating full distrust). For example, an *OD(p,s)* value of 0.2 indicates the level to which client *p* believes that service *s* engaged in behaviour that slightly increases distrust.

The *OT(p,s)* and *OD(p,s)* values do not need to sum up to 1 and can be evaluated based on provided goal models, as proposed in Section 4.1, or any other way the framework's user prefers.

3.3.2 Cumulative Trust and Distrust (T and D values)

These relations and their corresponding values are indicative of the service's historical performance as perceived by a specific service client. Furthermore, the relation's value is a metric that represents the client's first-hand assessment, after using a service provider. Said assessment takes into consideration the extend to which the service provider has or has not met the client's expectations in all of the so far elapsed interactions with this particular service provider.

As mentioned in Section 3.3.1, every time a service client interacts with a service, a pair of values, denoting observed trust and distrust (namely *OT(p, s) and OD(p, s)*), are calculated. These two values are then used to respectively update the overall, or cumulative, trust and distrust a service client *p* has on the specific service provider *s*, while also considering the service provider's previous performance. We denote by ***T(p, s)*** the value that indicates the belief the service client *p* has that the service provider *s* is a trustworthy provider, while by ***D(p,s)*** the belief the service client *p* has that the service provider *s* is not trustworthy. Same as their observed per interaction counterparts (i.e. *OT(p,s)* and *OD(p,s)* values), the cumulative trust (*T(p,s)*) and distrust (*D(p,s)*) values range from 0.0 to 1.0. The *T(p,s)* and *D(p,s)* values are set to a default value, if the user has no prior experience with the particular service, and are increasing or decreasing based on the discrepancy between values observed and calculated cumulative values, up until that point. The algorithm for the evaluation of *T(p,s)* and *D(p,s)* values is presented in Section 4.2. A high *T(p,s)* value indicates that the service *s* has historically demonstrated QoS that corresponds to the *p*'s expectations, whereas the opposite holds for a low *T(p,s)* value. Respectively, a high *D(p,s)* value is a clear indication that the service *s* has a propensity for participating in activities that the service client *p* deems suspicious, whereas a low *D(p,s)* value is indicative of lack of engaging in such behaviours.

Bear in mind that, said metrics are indicative of a service's performance, as perceived by a specific *service client*. Different clients may have different requirements and, thus, different opinions about services.

3.3.3 Individual Reputation of a Recommender (R value)

As it has already been mentioned, any node that has used a *service provider* can serve as a *recommender* to every other *service client*, who is requesting a recommendation regarding available services. After a recommendation that leads to the utilization of a service, a relation is created between recommender and recommendee. This relation denoted as R_p^r is indicative of a service client *p* interacting with another service client *r* to obtain recommendations.

Said relation, also, bears a corresponding value $R(p,r)$, which is a measure of trust that the r's recommendations are accurate. This measure is updated every time p uses a service, recommended by the aforementioned recommender r. The service client p compares its own experience regarding the service's QoS (i.e. after using the service) against the one advertised by the recommender r, and, based on any observed discrepancy, increases or decreases their perceived reputation for that recommender (i.e. $R(p,r)$). The value, therefore, is indicative of how much the particular service client p requesting the service, believes that this particular recommender r is trustworthy and reputable.

More specifically, this reputation value $R(p,r)$ is lowered every time the recommendation by r of a service s is inconsistent with the service client's p experience when using that particular service s. Similarly, the reputation value $R(p,r)$ is increased, whenever a service s is used, as a result of a recommendation, and the interaction meets the p's expectations, as those were formed due to said recommendation. More specifically, we denote the belief a service client p has that another service client r is a good recommender by $R(p,r)$. This reputation value corresponds only to a single service client's belief and ranges from 0.0 to 1.0. Again, a high $R(p,r)$ value is indicative of a recommender r that provides accurate and trustworthy recommendations, as far as a specific client p is concerned, whereas a low value is a testament to the opposite.

Rating recommenders and maintaining a distinct reputation value for each and every one of them provides the system with a number of advantages. First of all, a requesting user can choose the service clients from which they wish to receive a recommendation. Filtering is possible, thus allowing the requesting service client to receive opinions from a subset of recommenders whose recommendations are considered more trustworthy or at the very least more compatible to their own views. Moreover, the introduction of reputation values for reviewers of services enables the weighing of reviews or recommendations, based on said values. Multiple opinions can be considered, but their importance may vary depending on who provided them.

Existing commercial systems do not provide functionality for individual reputation values

of recommenders and, as a result, there is no way of filtering available reviews, considering them based on different levels of importance, or disregarding maliciously produced ones. Each requesting user has to manually investigate available opinions, which reduces the system's applicability in scenarios where the transacting components and corresponding business processes need to be automated.

3.3.4 Overall Reputation of Recommenders (AR value)

Newly introduced *service clients* have no previous interactions, and thus no relations to other clients of the framework. Because of that, they have no way of receiving recommendations or knowing about available services. Furthermore, *service clients* that are already participating in the network may form extremely tight-knit communities and fail to receive recommendations from other sources. To address both of those issues, we propose the use of another metric that indicates the overall or global reputation of a recommender r (i.e. $AR(r)$).

Based on the $R(k,r)$ values different service clients k have after receiving r's recommendations, we compute a metric indicating said r's overall reputation as a recommender. This overall reputation value $AR(r)$ is a function of the most important $R(k,r)$ values, where k are nodes that have an $R(k,r)$ value for r as a *recommender* (that is nodes k that have obtained and acted on recommendations from r in the past). Therefore, the overall recommendation ability (i.e. reputation as a *recommender*) of r denotes the collective belief that clients k, who have already used a service recommended by r and acquired a first-hand opinion of said service's QoS, have regarding r's ability to provide trustworthy recommendations. The collective belief that r is a good recommender is denoted by $AR(r)$. Note that, since the $AR(r)$ value depends on incoming $R(k,r)$ values (for all k nodes that have obtained a recommendation from r), it may need to be recalculated every time a new $R(k,r)$ value aimed at r is added or an existing one is updated.

A service client's $AR(r)$ value is indicative of the actual reputation of r within the network, since the opinions deemed most important (i.e. providing the most amount of information)

are the ones utilized. The choice to not utilize all available $R(k, r)$ values for r was made to improve the framework's throughput, decrease its network fingerprint and improve accuracy by considering only relevant opinions. The algorithm presented in Section 4.6, however, allows us to calculate a value that is indicative of the actual reputation of a service client within the network, while performing significantly less calculations and exchanging fewer messages.

3.4 Sources of Recommendations

When acquiring recommendations, a *service client* has to consult a variety of different sources. In the proposed approach, a client node p asks a set of recommenders $S = r_1, r_2, \ldots r_n$ for recommendations regarding available services. This set S includes recommenders selected for different reasons and is composed of:

a) *Expert recommenders*. These are nodes r_i in the social graph with the highest AR values. The selection threshold can be set as a parameter (e.g. the nodes at the top 10 percentile of AR values, or just the top 20 nodes with the highest AR values). The threshold does not affect the overall behaviour of the framework, since it merely allows for more (or less) recommenders to participate in any given recommendation request.

b) *Friends*. These are nodes r_i in the social graph from which node p has obtained recommendations in the past, and acted on them, thus creating an $R_p^{r_i}$ relation and a corresponding $R(p, r_i)$ value. In Fig. 3.2 node SC_4 is a *friend* to SC_1 as SC_4 has provided recommendations to SC_1 in the past. Again, the client can select to consult only the ones with the highest reputation values.

c) *Friends of friends*, that is nodes from which friend nodes have obtained recommendations. In Fig. 3.2 node SC_2 is a *friend of friend* of SC_1, as SC_5 has provided recommendations to SC_1 and SC_2 has provided recommendations to SC_5 in the past.

Note that we have chosen to include only paths of length up to two. One could explore all

available paths, but this would result in an approach utilizing the *Flow* method of value aggregation, as demonstrated in *Eigentrust* [3]. This method is very computationally intensive and requires a set of very specific preconditions, such as a number of pre-trusted users. Furthermore, following all available paths would reveal more available services, but since we do not consider trust to be entirely transitional (i.e. trust should be weighed based on each node's reputation within a path and the overall weight should account for the product of all weights, as is the case when a t-norm operator is used [30]), recommendations would be diluted and rendered effectively meaningless after more than two hops in the social graph that includes recommenders and service providers.

3.5 Process Overview

3.5.1 Process Outline

In this section, we outline the process the system follows during a session where a client wishes to use a service. The process can be broken down into fourteen steps as follows:

Step 1: The client expresses interest in using a service and request a list of available services ranked based on trustworthiness.

Step 2: Recommenders from each of the sources of recommendation explained in Section 3.4 (i.e. *Experts, Friends, Friends of Friends*) are selected.

Step 3: Service recommendations are collected from each of the recommenders selected in the previous step.

Step 4: The client's opinions, pertaining to previous interactions with services, are added to the pool of values obtained from the recommenders.

Step 5: All provided recommendations are transformed as sets of positive (i.e. in case a recommendation is in favour of a service) and negative evidences (i.e. in case a recommendation

is against the use of a service), in the format required by the utilized by the ranking algorithm (in our case the Demster-Schafer algorithm).

Step 6: The ranking algorithm is applied, and a ranking of the services is provided to the user. Note that for the prototype we have utilized a variation of the Dempster-Shafer algorithm [22] (see Section 4.7.3), but the approach supports any ranking algorithm.

Step 7: Incentives (if any) provided by service providers are taken into consideration and an supplementary information regarding services is, also, provided to the user (see Section 4.7.4).

Step 8: The client chooses their preferred service and proceeds with the utilization of said service.

Step 9: Data are collected from the interaction, either automatically or through user feedback, and the observed *trust* and *distrust* values (i.e. the **OT(p, s)** and **OD(p, s)** values, see Section 3.3.1) are evaluated.

Step 10: The utilized service provider can opt to provide compensations to the client, if extenuating circumstances led the service provider to offer a QoS below what was expected by the client.

Step 11: The client can choose to accept the compensation and adjust their observed **OT(p, s)** and **OD(p, s)** values or retain their observed ones.

Step 12: The client's opinion about the utilized service is updated as specified in **Algorithm T-D** presented in Section 4.2, using the observed **OT(p, s)** and **OD(p, s)** values, adjusted or not.

Step 13: For every recommender r that endorsed the utilized service to a service client s, their $R(p,r)$ value is updated based on observed values, adjusted after compensation or not, and source of recommendations through which they were selected, as specified in **Algorithm R** presented in Section 4.3.

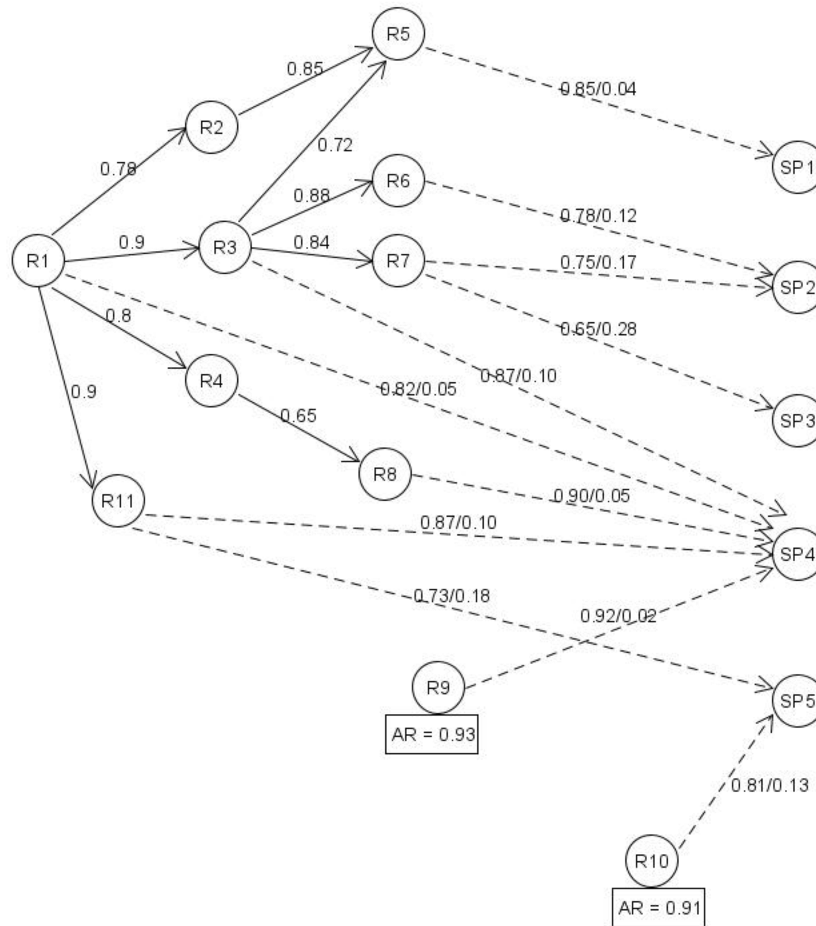

Figure 3.3: Example network of recommenders and services

Step 14: Finally, the recalculation of the corresponding $AR(r)$ value is triggered if the updated $R(p,r)$ value is relevant to the evaluation of said $AR(r)$ value (see Section 4.6). The **Algorithm AR**, presented in Section 4.5, is used.

3.5.2 Running Example

To clarify the process followed in our approach to provide a service ranking and update the reputation of recommender and service nodes after utilization of a service, a running example will be provided in this Section. Said example will demonstrate the steps, but will not provide the detailed calculations at this point. We will revisit the exact same example when we define

and explain the algorithms in Section 4. The subset of the network involved in this scenario is portrayed in Fig. 3.3. We assume the percentage of recommenders chosen per source of recommendations corresponds to 2 recommenders per source, to simplify the provided example. Following the steps provided in Section 3.5.1:

Step 1: User **R1** initiates process.

Step 2: Different sources of recommendations are consulted and recommender groups are created as follows:

- **R1** investigates the **R** values of his friends and finds that among **R2, R3, R4, R11**, the ones with the highest values are **R3** and **R11**, both having ***R*** = *0.9*.
- All available 2-step paths are then explored looking for the 2 with the highest value. The paths with the highest value are the ones going to **R6** and **R7** through **R3** with corresponding values of *0.792* and *0.756*.
- Finally, the system inspects the **AR** values of all available recommenders. Let's assume that **R9** and **R10** have the highest values of *0.93* and *0.91* respectively.
- At the end of this step, we have :
  - *Friends* = {**R3**, **R11**}
  - *FriendsOfFriends* = {**R3-R6**, **R3-R7**}
  - *Experts* = {**R9**, **R10**}.

  Recommendations about services **SP2**, **SP3**, **SP4** and **SP5** are provided by them.

Note that, even though **SP1** is part of the system, it was not recommended by any of the chosen recommenders, so it is not included in the following steps.

Step 3: For each service, we acquire all available values from the recommenders selected. The recommendations for each service are as follows:

- ***SP2***: Only recommendations from the *FriendsOfFriends* group are available. More specifically, the following values are available:

  → **T(R6,SP2)** = *0.78* and **D(R6,SP2)** = *0.12*

  → **T(R7,SP2)** = *0.75* and **D(R7,SP2)** = *0.17*

- ***SP3***: Only recommendations from the *FriendsOfFriends* group are available. More specifically, the following values are available:

  → **T(R7,SP3)** = *0.65* and **D(R7,SP3)** = *0.28*

- ***SP4***: Recommendations from *Friends* and *Experts* groups are available. More specifically, the following values are available:

  → **T(R3,SP4)** = *0.87* and **D(R3,SP4)** = *0.10*

  → **T(R11,SP4)** = *0.87* and **D(R11,SP4)** = *0.10*

  → **T(R9,SP4)** = *0.92* and **D(R9,SP4)** = *0.02*

- ***SP5***: Recommendations are available from *Friends* and *Experts* groups, so:

  → **T(R11,SP5)** = *0.73* and **D(R11,SP5)** = *0.18*

  → **T(R10,SP5)** = *0.81* and **D(R10,SP5)** = *0.13*

Step 4: The requesting user **R1** has a recommendation for **SP4**.

- **T(R1,SP4)** = *0.82* and **D(R1,SP4)** = *0.05*

Step 5: All **T** values are considered evidences in favor of that service and all **D** values are considered against, meaning that the user would rather use any service other than the one being evaluated.

Step 6: After running the chosen ranking algorithm, a list with the services is returned. For that particular example let's assume that the ordered list returned by the algorithm looks like that: **[SP4, SP5, SP3, SP2]**. This ordering means that service **SP4** is considered the most trustworthy one.

Step 7: Supplementary information is provided to the user, pertaining to incentives offered by services that are not at the top of the ranking.

Step 8: Let's assume the user does not utilize any of the offered incentives and uses the service with the highest ranking, i.e. **SP4**.

Step 9: The chosen service is used, and its performance is evaluated using the chosen method. In this example let's assume **OT(R1,SP4)** = *0.89*, **OD(R1,SP4)** = *0.07*.

Step 10: The service performed as expected, so no compensations are offered this time.

Step 11: There is no choice involved this time around in this step.

Step 12: Since the requesting user already had an opinion about **SP4**, the **T(R1,SP4)** and **D(R1,SP4)** values need to be updated according to the algorithm presented in Section 4.2.

Step 13: The recommenders that endorsed **SP4** are **R3, R11, R9**. Their new **R(R1,Ri)** values are computed using the algorithm described in Section 4.3.

Step 14: The **AR(Ri)** values are recalculated for **R3, R11, R9** according to the algorithm described in Section 4.5, if the new *R(R1,Ri)* values are considered important based on the algorithm presented in Section 4.6.

3.6 Summary

Summarizing, each user of the proposed approach can participate as either a service client p or a service provider s. Service providers need to register multiple services as different nodes, and if a user wishes to participate with both roles, separate entities need to be registered, as well.

Relations are created in the context of the framework between different entities. Said relations have corresponding values and are differentiated based on the receiving party of it. More specifically, all relations have a service client p as the source and the target is either another service client r, in which case we have a relation and corresponding value that indicates the

historically perceived reputation of said client and are denoted by R_p^r and $R(p, r)$ respectively, or a service provider s, in which case we have a pair of relations and corresponding pair of values, that indicate the historically perceived trust and distrust on said service and are denoted by $< T_p^s, D_p^s >$ and $< T(p, s,), D(p, s) >$ for relations and values respectively. A unary relation and derived value are, also, proposed to represent the overall reputation of a service client r within the system. Relation and corresponding value are denoted by AR^r and $AR(r)$. The value depends on $R(p, r)$ values that are aimed at the service client in question.

When a service client is requesting recommendations for available services, a number of sources are consulted, namely the system's *Experts* and the client's *Friends*, as well as their friends (i.e. *Friends of Friends*). The service client's personal opinions are, also, considered and a ranking of services is calculated. Incentives can be offered by lower ranked services, and the user can choose to consider them or not. After an interaction occurs, the client's personal opinion, as well as the reputation of those who recommended the utilized service, is adjusted, based on discrepancy between observed and provided values.

Chapter 4

Trust and Reputation Evaluation Algorithms

In this chapter, we presenting the algorithms to evaluate first, the perceived quality of service experienced in individual interactions by a client *p* when uses a service provider *s* (*OT(p, s)* and *OD(p, s)* values) second, the cumulative trust and distrust values clients have that a service provider can meet the client's expectations (*T(p, s)* and *D(p, s)* values) third, the reputation of a client *p* service assigns to a client r acting as recommender (*R(p, r)* value) and fourth, a recommender's *r* overall reputation (*AR(r)* value). In addition to the above, we present algorithms to deal with obsolete (i.e. stale) values using (*timeouts* and to identify the most important *R* values to be considered for each *AR(r)* value using a cache management policy (i.e. the (*ARC* policy) which we have adapted to meet the requirements of the problem associated with this thesis.

4.1 Evaluation of OT and OD values

The proposed approach does not place any constraints on the method to be used by a client *p* to assign *OT(p, s* and *OD(p, s)* values after its interaction with a service *s*. These may include fuzzy logic, statistical analysis, probabilistic reasoning etc. For the prototype implementation,

a client sets its expectations from the service in the form of models, such as goal models [23]. The user-provided goal models specify the client's requirements, as far as this particular type of service type is concerned. Since the service provider nodes are proxies of actual services, monitoring components that autimatically evaluate the QoS offered by a service provider and as this is assessed by a client's perspective. Based on the type of service provided, a proxy can attain and process information regarding a multitude of service quality characteristics such as the security protocols used, the speed of the transaction, the provenance of the data used by the service, or even the location of the server hosting the service. The data can be used as input to a goal model denoting the client's expectations from the service. The level of satisfaction of this goal model will yield the *OT(p, s)* and *OD(p, s)* for client p using the service s. The evaluation of the goal models may take various forms. If we choose to utilize the approach proposed in [24], goal model are transformed into fuzzy rules, which are, in turn, evaluated using a fuzzy reasoner. The result of the evaluation would indicate how well the service performed based on the client's requirements. One example of such a goal model denoting a client's expectations from a service provider can be seen in Figure 4.1.

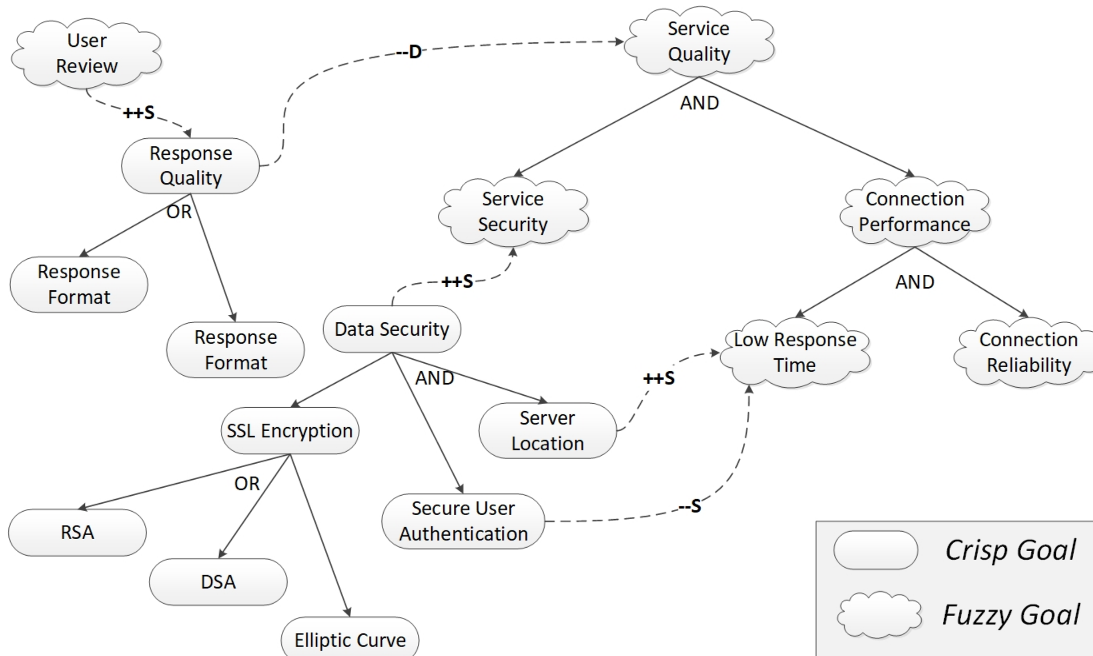

Figure 4.1: Example model for service evaluation

4.2 Evaluation of T and D values

Apart from the observed trust (i.e. *OT(p, s)*) and distrust values (i.e. *OD(p, s)*) obtained through each individual interaction between a user p and a service provider s , another set of cumulative values is maintained for each pair of client and provider that have had at least one interaction, as mentioned in Section 3.3.2.

Cumulative trust and distrust are denoted by ***T(p, s)*** and ***D(p, s)***. They are updated every time a) a new set of observed values, corresponding to the users in question, occurs or b) a set of observed values are deemed obsolete and should no longer participate in the calculation of the cumulative values. Of those two processes, the first one is described below and is triggered by a client utilizing a service, whereas the second one is explained in Section 4.4 and is time dependent.

As specified in the previous section, every time a client p utilizes a service s, two values are computed upon completion of said interaction. Those observed values are denoted by $OT(p,s)$ and $OD(p,s)$ and are used to update the $T(p,s)$ and $D(p,s)$ values respectively. A specific algorithm is utilized for that update and its purpose is to recalculate the $T(p,s)$ value, which denotes how satisfied the service client p is with service s, and the $D(p,s)$ value, which denotes the extend to which client p is dissatisfied with aspects of service s overall, based on the history of observed values pertaining to utilizations of s by p (i.e. the cumulative values of trust and distrust placed on service s by p).

More specifically, the new $T(p,s)$ and $D(p,s)$ values occur by taking into consideration: a) the observed values computed by using the service (see *OT(p, s)* and *OD(p, s)* values) and; b) the values of $T(p,s)$ and $D(p,s)$ prior to p's last utilization of service s. The specific algorithm can be observed on Alg.1.

Algorithm 1 Calculate T-D values

```
1:  - Let s be a service provider
2:  - Let p be the client using service s
3:  - Let c be a parameter indicating the importance of new observations taking values in the
    interval (0,1) with c = 0 indicating that new values have no meaning and c = 1 indicating
    that the latest value is the only one that matters.
4:
5:  procedure CALCULATETD(OT(p, s), OD(p, s), T(p, s), D(p, s))
6:      if (T(p, s) OR D(p, s) not available) then
7:          T(p, s) = initializeDefault() (e.g. 0.5)
8:          D(p, s) = initializeDefault() (e.g. 0.5)
9:          count(p, s) = 0
10:     end if
11:
12:     T_offset(p, s) = c(OT(p, s) − T(p, s))
13:     D_offset(p, s) = c(OD(p, s) − D(p, s))
14:     count(p, s) = count + 1
15:
16:     T(p, s) = T(p, s) + T_offset(p, s)
17:     D(p, s) = D(p, s) + D_offset(p, s)
18:
19:     notify(T_offset(p, s), D_offset(p, s))
20:     return < T(p, s), D(p, s) >
21: end procedure
```

As it is evident, the algorithm initially looks for previous values of *trust* (i.e, *T(p, s)*) and *distrust* (i.e., *D(p, s)*), indicating that client *p* has interacted with service *s* in the past (line 6). If none can be found, the cumulative trust and distrust values are set to a default value (lines 7-8) and the interaction counter is set to 0 (line 9).

An offset value, based on the discrepancy between observed and cumulative values, is calculated (lines 12-13). Note here that, if the observed trust or distrust value is greater than the cumulative trust or distrust value, the offset is positive. The meaning of this is different in the each case. A higher trust value is appreciated, while a higher distrust value is problematic.

In any case, though, the cumulative values slowly adjust upwards or downwards based on the observed interactions. This is done by adding the offset to the cumulative values, weighed according to the user-specified parameter *c*, which ranges between 0.0 and 1.0. The parameter is utilized to allow for customization, since its value indicates the importance of newly observed

values over historical performance of the service. Values close to the lower end produce very small offsets, thus adjusting the cumulative values in a minor way. Values closer to the upper end, however, put major emphasis on the newly observed trust and distrust, by producing larger offset and significant adjustment of the cumulative values.

Note that, even though the *T(p, s)*and *D(p, s)* values are supposed to range between 0.0 and 1.0, there is no verification part of the algorithm to ensure that they stay within the limits. This is not really an issue, though, since the offset depends on the difference between cumulative value and observed value. Observed values are within range by design, so there is no way that the difference multiplied by c, which is less than 1.0, is going to force the cumulative value off the limits.

The interaction counter is also incremented by 1 (line 14), which allows us to know to keep track of the number of interactions that have elapsed between this particular pair of service client and service provider. When the last of interactions becomes obsolete, we can totally remove the entry, thus safeguarding against potential mathematical errors that may occur when dealing with real numbers.

Last but not least, the framework is *notified* about the calculated offsets, so that they can be timestamped and reversed when the time comes (see Section 4.4). More specifically, the *notify* method (line 19) produces an event that is received by the corresponding algorithms dealing with timeouts, in order to save the offset value along with its timestamp, so that it can be retrieved and deleted when it becomes stale (see Algorithm 3 and Section 4.4).

4.3 Evaluation of R value

As mentioned in Section 3.3.3, each client p maintains a reputation value *R(p,r)* for every recommender r that has provided at least one recommendation that has resulted to utilization of recommended service.

Said value is denoted by *R(p,r)* and is updated every time a) a recommendation, leading

to the service utilization, is made or b) a previous recommendation is deemed obsolete due to time elasped, and should not be considered when calculating a recommender's reputation value anymore. Similarly to the *T(p,s)* and *D(p,s)* values, the first update case of the *R(p, s)* value is discussed below, and is triggered by a client after utilizing a service following the recommendation by one or more recommender *r*, whereas the second update case is due to the *R(p, r)* value becoming stale and is discussed in Section 4.4.

As mentioned before, a service's *s* utilization by a client *p* occurs after other nodes *r* within the framework provide a recommendation pertaining to said service *s*. Interaction of s service client *p* with the service *s* produces a set of observed values *OT(p,s)* and *OD(p,s)* values (see Section 4.1) based on the client's preferences (i.e. goal models). Those observed values can, then, be juxtaposed with the values provided by each recommender *r* , as part of the recommendation part of the process. The discrepancy between the two can, then, be used to update the recommender's *r* reputation value *R(p,r)*, as far as that specific client *p* is concerned. The type of recommender and previous (i.e. historical) reputation value are, also, taken into account when calculating the updated reputation value.

To further explain, consider the following scenario. Client *p* asks client *r* for a recommendation about services of a specific type (i.e. process monitoring services), *r* provides a recommendation for service *s* (see $T(r, s)$, and $D(r, s)$ values), among others. *p* ends up using service *s* as a result of this recommendation, and now has first-hand experience about the service's behaviour (see $OT(p, s)$ and $OD(p, s)$ values).As a result, *p* can now, finally, form an opinion of how good *r*'s recommendation was (i.e. how trustworthy *r* is as a recommender for recommending service *s*) and update the *R* value node *p* has for node *r* (i.e. $R(p, r)$), based on elapsed interaction, previous value and type of recommender (see Section 3.4).

Each reputation value $R(p, r)$ indicates how much trust a service client *p* places on the recommendations *r* provides, based on past experience, and the exact algorithm utilized for its calculation is presented in Alg.2.

Algorithm 2 Calculate R value

```
1:  - Let p be the node that is asking for the recommendation of a service
2:  - Let r be the recommender node that is recommending a service
3:  - Let error be the difference between observed and provided trust and distrust
4:  - Let offset be the offset that is to be applied to the trust in r's reputation value based on
    new data
5:
6:  procedure calculateR(OT(p, s), OD(p, s), T(r, s), D(r, s))
7:      error(p, r) = |T(r, s)–OT(p, s)| + |D(r, s) − OD(p, s)|
8:      R_offset(p, r) = calculateOffset(error)
9:
10:     if r in expert nodes then
11:         /* that is r is in the top of the list of recommenders
12:         ranked by their AR value (see Section 3.4) */
13:         R(p, r) = AR(r) + R_offset(p, r)
14:         count(p, r) = 1
15:     else if r is friend of p then
16:         /* That is r is in the top of the list of recommenders
17:         ranked by their R value provided by p (see Section
18:         3.4) */
19:         R(p, r) = R(p, r) + R_offset(p, r)
20:         count(p, r) = count + 1
21:     else if r is a friend of friend k of p then
22:         /* that is r is in the top of the list of recommenders
23:         ranked by the cross product of R values provided
24:         by p and its friends, in all two step paths emanating
25:         from p (see Section 3.4) */
26:         R(p, r) = (R(p, k) * R(k, r)) + R_offset(p, r)
27:         count(p, r) = 1
28:         R_offset(p, k) = R(k, r) * R_offset(p, r)
29:         R(p, k) = R(p, k) + R_offset(p, k)
30:         count(p, k) = count + 1
31:     end if
32:
33:     notify(R_offset(p, r), R_offset(p, k)) // For timeout purposes
34:     notify(R(p, r)) // For AR calculation purposes
35:
36:     return < R(p, r), count(p, r) >
37: end procedure
```

An error value is initially calculated (line 7), accounting for the difference between trust and distrust values provided by the recommender and values observed through the recent interaction

with the service. We use the absolute difference, since in this use case it doesn't matter whether the recommendation values were over or under the observed ones, but rather how large the discrepancy is. Overvaluing (i.e. proposing that a service behaves in a much better way than it actually is) is as bad, if not worse, as undervaluing (i.e. downplaying a service's actual performance). So, in either case, the larger the discrepancy, the greater the error value will be.

Every approach in the related literature that automatically adjusts a recommender's reputation based on elapsed interaction, proposes a unique custom formula that holds certain properties [9, 12, 19, 138]. Said formula is utilized to provide the updated reputation value. There is no specific standard for creating such formulas, other than the fact that they must account for the discrepancy between values provided by the recommender and observed values. In our approach, we propose a formula that calculates offsets (line 8), rather than directly updating the reputation value. The rationale behind this comes from our choice to account for data aging. We have opted to disregard values for obsolete (i.e. distant past) interactions, as will be further explained in Section 4.4, but in a way that allows our framework to *a)* maintain a minimum network fingerprint and *b)* have a high throughput and support as many users as possible. The offset approach allows us to reduce the amount of calculations required and messages sent.

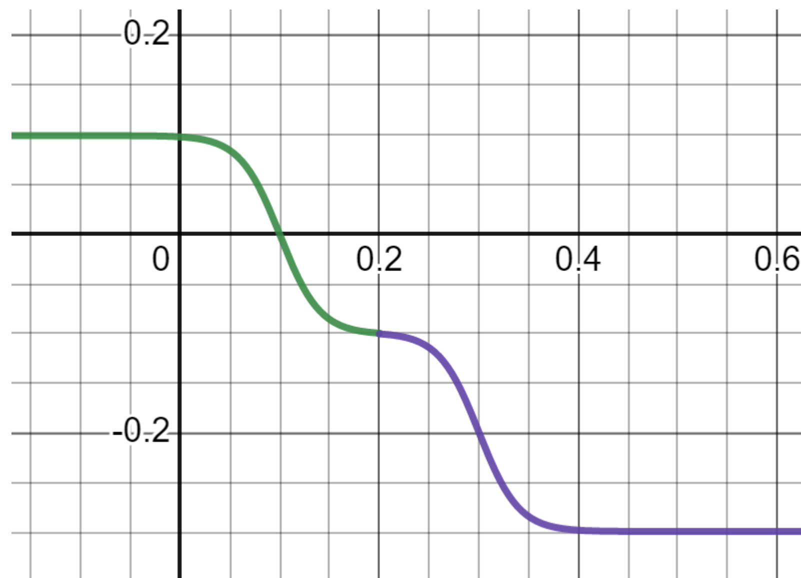

Figure 4.2: Offset Formula for R evaluation

Our custom offset formula was created with specific properties in mind. Resulting offset must *a)* depend on calculated error (i.e. discrepancy between provided values and observed values), *b)* allow for minimal values for insignificant errors, *c)* provide exponentially larger values as error values increase and *d)* reach its maximum value for a specific error value and remain stable regardless of potential further increases. The proposed formula can be seen plotted in Fig. 4.2 and in equation form in Eq. 4.1.

$$R_{offset}(p,r) = \begin{cases} 0.1 * \left[\frac{2}{1+e^{(50*(error-0.1))}} - 1.01 \right] & ,error < 0.2 \\ 0.1 * \left[\frac{2}{1+e^{(50*(error-0.3))}} - 2.99 \right] & ,error \geq 0.2 \end{cases} \tag{4.1}$$

The type of recommender is, then, inquired (lines 10,15,21) to identify the type of value that led to their inclusion in the list of recommenders (see Section 4.7.1). Since there are three sources of information consulted during the recommendation part of the process, there are three possible types of recommenders who could have recommended the utilized service. Said types and corresponding cases are as follows:

- Recommender is a *Friend* node: The requesting client has interacted with that particular recommender in the past and has a *R* relation and corresponding value towards it (see Section 3.3.3). Since the reputation value *R(p,r)* is the reason this node was selected as a recommender, the offset is added to said value (line 19). The interaction counter is incremented by one (line 20), as an additional interaction now contributes to the cumulative value.

- Recommender is an *Expert* node: The node's overall reputation (see Section 3.3.4) is utilized, since it was the reason that particular client was chosen to be a recommender. The node's *AR* value is now the baseline to which the offset will be added (line 13). Note that since this node was not selected due to an existing *R* relation with the requesting client, a new one has to be created. The interaction counter, therefore, is set to 1 (line 14).

- Recommender is a *Friend of Friend* node: In this case, the recommendation originated from a node that was in turn recommended by a node with an already existing reputation value. After the interaction with the recommended service concludes, both of those nodes' reputation value needs to be updated. The baseline for each case is different. For the immediate friend, the already existing R value is used (i.e. $R(p, friend)$, where p is the requesting client) (line 29). For the second step in the path, however, both the *friend*'s reputation and the recommendation for the *friend of friend* have to be taken into consideration (line 26). We propose the utilization of the t-norm operator, as specified in [30]. The resulting baseline reputation value for the *Friend of Friend* recommender node is the product of the two aforementioned reputation values (i.e. $R(p, k) * R(k, r)$, where k is a *Friend* and r is a *FriendOfFriend*). As far as the offset is concerned, it remains as is for the *Friend of Friend* node, but is fittingly adjusted for the *friend* node (line 28). It is only fair that this node is penalized or rewarded proportionally to the recommendation value offered for the actual recommender (i.e. *Friend of Friend*).

Different approaches have been proposed regarding propagation of trust in social graphs [20]. We chose to use the multiplication norm (i.e. t-norm operator) in the friend of friend scenario since it resonates more with the world we are trying to approximate through our approach.

In any case, the utilized value (i.e., *R(p,r)* value for *Friend*, *AR(r)* value for *Expert* and *R(p,k)*R(k,r)* value for *FriendOfFriend*) is then considered to be the previous reputation value or baseline, the offset is added to it and the result is assigned to the reputation value of client p for r (i.e. $R(p, r)$) (line 26).

The interaction counter (i.e., $count(p, r)$) allows us to keep track of the number of interactions that have elapsed between this particular pair of service clients. When an interaction becomes obsolete, we can totally remove the entry, thus safeguarding against *a)* potential mathematical errors that may occur when dealing with real numbers, and *b)* issues with message ordering in timeout scenarios. The interaction counter is either incremented by 1, if a previous

R relations led to the consultation of recommender in question (lines 20, 30), or reset to 1, in any other case (lines 14, 27).

Note that, even though the *R* values are supposed to range between 0.0 and 1.0, there is no such provision in the algorithm. This is not an omission, but is done by design. As it will become evident in Section 4.4, offsets are saved and later subtracted from the cumulative value, when the corresponding interaction has to be disregarded. To avoid issues with that process, we do not cap the reputation values, but we make sure that when utilization of reputation values that are out of range occurs, the framework treats them as if they are exactly at the corresponding end of the range (i.e. 0.0, if negative, and 1.0, if over 1.0).

Last but not least, the framework is *notified* about both the calculated offset (line 33), so that it can be timestamped and reversed when the time comes (see Section 4.4), and the new derived reputation value (line 34), which will trigger the process for selecting the most important *R* values for the calculation of *AR* values (see Section 4.6). More specifically, the first *notify* method produces an event that is received by the corresponding algorithms dealing with timeouts, in order to save the offset value along with its timestamp, so that it can be retrieved and deleted when it becomes stale (see Algorithm 3 and Section 4.4). Discarding stale values is very important as it allows to first maintain a tractable set of values observed through interactions and second consider values based on the most up-to-date behaviour of service providers in question. As for the second *notify* method, the event produced by it is consumed by the algorithms tasked with deciding the *R* values to be considered for *AR* values. The procedure is run and the corresponding lists are updated (see Algorithm 6 and Section 4.6).

4.4 Timeouts

As specified in the previous sections regarding the update process of trust and distrust values (**T(p,s) and D(p,s)**) and reputation values (**R(p,r)**), every time an adjustment is made, the framework is notified. Based on the specific context and the observed frequency of interactions,

a time window is defined and each received notification is timestamped and filed.

A specific component (see Chapter 5) with a completely separate process is executed in parallel to discover obsolete values, taking into consideration the corresponding time window and each value's timestamp. Once an applied offset is deemed to be outdated, the appropriate procedure is run to remove the effect of said value. All procedures are presented in Alg. 3 and correspond to either trust/distrust or reputation offsets.

Algorithm 3 Remove obsolete offsets

```
1:  - Let p be the node that asked for recommendation
2:  - Let r be the node that recommended a service
3:  - Let s be a utilized service provider
4:
5:  procedure deleteTDOffset(T_offset(p, s), D_offset(p, s), T(p, s), D(p, s), count(p, s))
6:      if count(p, s) <= 1 then
7:          /* This is the only set of trust/distrust values to
8:          be removed */
9:          T(p, s) = null
10:         D(p, s) = null
11:     else
12:         /* The offset needs to be removed */
13:         T(p, s) = T(p, s) − T_offset(p, s)
14:         D(p, s) = D(p, s) − D_offset(p, s)
15:         count(p, s) = count − 1
16:     end if
17: end procedure
18:
19: procedure deleteROffset(R_offset, R(p, r), count(p, r))
20:     if count(p, r) <= 1 then
21:         /* This is the last recommendation to
22:         be removed */
23:         R(p, r) = null
24:     else
25:         /* The offset needs to be removed */
26:         R(p, r) = R(p, r) − R_offset(p, r)
27:         count(p, r) = count − 1
28:     end if
29: end procedure
```

Note that the number of updates that have contributed to a value is, also, maintained (i.e $count(p,r)$). The interaction counter is checked (lines 6, 20) and when the last of the adjust-

ments has to be reverted, the values (line 9-10), or value (line 23), are invalidated and removed altogether. We opted to totally get rid of the value to signify the lack of connection in the social graph and differentiate between lack of value and a series of very bad recommendations or interactions that would lead to a zero value.

If, however, the offset(s) to be removed is not the last one, the $T_{offset}(p, s)$ and $D_{offset}(p, s)$ values, or $R_{offset}(p, r)$ value, are subtracted from the $T(p, s)$ and $D(p, s)$ cumulative values (lines 13-14), or $R(p, r)$ respectively (line 26). The interaction counter is, also, decreased by 1 (lines 15, 27), indicating that there is one less interaction contributing to the cumulative values.

Algorithm 4 Calculate AR value

```
 1: - This is the variation for calculating AR values. The second variation is in Alg. 5.
 2: - Algorithm 6 is differentiated in lines 20-23 and 36-39 to account for this variation.
 3:
 4: - Let r be the recommender node whose AR value is calculated
 5: - Let N_r be the set of nodes occurring by applying the Alg. 6 (lines 23 and 39) described
    in Section 4.6 for recommender r.
 6:
 7: procedure CALCULATEAR(N_r, R[size(N_r)])
 8:     sum(r) = 0
 9:     weightSum(r) = 0
10:     for each client node w in N_r do
11:         if AR(w) is not available then
12:             AR(w) = initializeDefault (e.g. 0.5)
13:         end if
14:         sum(r) = sum(r) + (R(w, r) * AR(w))
15:         weightSum(r) = weightSum(r) + AR(w)
16:     end for
17:     AR(r) = sum(r)/weightSum(r)
18: end procedure
```

4.5 Evaluation of AR value

Every client r participating in the network has an *AR(r)* value that represents the overall reputation of the node as a recommender. An *AR(r)* value fluctuates over time depending on the recommendations node r has provided for services and other recommenders of the network

and how accurate they have been. If the recommendations provided by *r* are consistent with the experience of other nodes who have used a service, as a result of *r*'s recommendations, *r*'s *AR(r)* value increases. If, however, discrepancies occur between provided and observed values, the value decreases. Each *AR(r)* value is a function of the collective belief demonstrated by a set N_r of other nodes that *r* is a good recommender.

This collective belief is computed by considering the *R* values all those nodes have for *r*, following the recommendations from *r* and the consequent utilization of the recommended by *r* services, thus getting first-hand experience and being able to assess *r*'s accountability. The algorithm used to compute the *AR(r)* values is presented below in Alg. 4. The main idea comes from the PageRank algorithm [4] and is based on the summation of reputation values *R(w,r)*, aimed at the client in question, with each element weighed based on their own *AR(w)* value. To allow, however, for introduction of new nodes into the system, each node that has no incoming edges in the social graph is provided with a baseline value of *AR(w)* of 0.5.

Also, instead of considering all possible reputation values, we propose Algorithm 6 that is presented in Section 4.6 to select a subset N_r of available values based on a specific logic. The choice to use a subset was made to improve the performance of the framework, since node churn could be high in reputation systems, and to allow for a more selective choice of options that is based on recentness and frequency. The rationale behind selecting the subset N_r is that the majority of important opinions comes from a select subset of users that either interact with the recommender in question frequently, or have interacted recently and their recommendation is up to date.

For each of the members of N_r, their recommendation is weighed based on their overall reputation value (i.e. *AR(w)* value) and added to the *sum(r)* (line 14), while their *AR(w)* value is added to the *weightSum(r)* (line 15). After the recommendations of all recommenders have been considered, the calculated *AR(r)* value is produced by dividing the two sums (line 17).

As mentioned earlier, if one of the contributing clients have no overall reputation value (line 11), they are assigned a default one (line 12) that can either be set to be a specific value (e.g.

0.5) or a value corresponding to the average or mean value within the system. Also, bear in mind, that the default value is discarded as soon as at least one incoming *R* relation is created.

Even though this algorithm captures the logic behind the calculation of the overall reputation values, it is not very efficient in terms of number of calculations and requests for information. It quickly becomes evident that, the summation of all applicable values is not required every time one of them changes. To account for that and allow for the event-driven approach we are proposing, a second variation of Alg. 4 is presented, where changes in the *AR(r)* values are made on a per request mode. This variation can be seen in Alg. 5.

Algorithm 5 Update AR value

1: - Let r be the recommender node whose AR value is updated
2: - Let I_r be the set of pair values $< R(w,r), AR(w) >$ that need to be included, based on Alg. 6, in the calculation of the overall reputation value.
3: - Let E_r be the set of pair values $< R(w,r), AR(w) >$ that need to be excluded, based on Alg. 6, from the calculation of the overall reputation value.
4:
5: **procedure** UPDATEAR($I_r, E_r, sum(r), weightSum(r)$)
6:     **if** $sum(r)$ is not available **then**
7:         $sum(r) = 0$
8:         $weightSum(r) = 0$
9:     **end if**
10:
11:     **for** each $< R(w,r), AR(w) >$ pair in E_r **do**
12:         $sum(r) = sum(r) - (R(w,r) * AR(w))$
13:         $weightSum(r) = weightSum(r) - AR(w)$
14:     **end for**
15:
16:     **for** each $< R(w,r), AR(w) >$ pair in I_r **do**
17:         $sum(r) = sum(r) + (R(w,r) * AR(w))$
18:         $weightSum(r) = weightSum(r) + AR(w)$
19:     **end for**
20:     ***save***$(sum(r), weightSum(r))$
21:     $AR(r) = sum(r)/weightSum(r)$
22: **end procedure**

Note that this updated algorithm receives the *sum(r)* and *weightSum(r)* as parameters (line 5). It, also, receives as parameters the pairs of values $< R(w,r), AR(w) >$ that need to be added or deleted. This approach requires fewer calculations and can easily be used to apply changes

in predetermined intervals and in a batch manner, in a way that, also, minimizes network utilization. Updating a *R(w,r)* value, which is already in N_r, is equivalent to deleting the old and adding the new one. If the particular client never had their overall reputation value calculated in the past (i.e. it is the first time values are added) (line 6), we make sure that the *sum(r)* and *weightSum(r)* values are initialized (lines 7-8), before the execution of the rest of the algorithm.

The set of values that need to be included and excluded from the calculation are discovered and provided by the proposed method for selecting only the important opinions presented in Section 4.6.

For each pair of values that are to be excluded (line 11), the reputation value $R(w, r)$ is weighted based on the recommender's overall reputation value (i.e. $AR(w)$) and is, then subtracted from $sum(r)$ (line 12). Each weight is, also, subtracted from $weightSum(r)$ (line 13). A similar process is followed for the pair of values that are to be included (line 16), only in this case the weighted reputation value (i.e. $R(w, r) * AR(w)$) and the weight (i.e. $AR(w)$) are added to $sum(r)$ (line 17) and $weightSum(r)$ (line 18) respectively.

Finally, the algorithm saves the summation values (line 20) and performs the division to provide the updated *AR(r)* value of the particular recommender (line 21).

4.6 Selection of R values for AR calculation

The calculation of the *AR(r)* value of each node *r* depends on *R(w,r)* reputation values assigned to the node in question by other nodes *w*. One option would be to consider all available values, but since the number of values could grow exponentially as more and more users participate in the network this could prove to be less than optional. Furthermore, not all opinions should bear the same significance, especially if they are older or not updated at the same frequency as other ones. Old or not frequently used values may indicate a recommender who is either not actively participating in the network or may have stale or obsolete opinions about certain service providers who may have changed their behaviour or quality of service in the time

elapsed since their last interaction with the recommender in question. For that reason, a way to come up with a subset of available values is required.

One could choose to sort the *R(w,r)* values by the time they were last updated and disregard the ones that were least recently used (LRU [92]). Another option would be to sort based on number of updates so far and exclude those with the lowest frequency of updates (LFU [93]). We feel, however, that a hybrid approach would be the most appropriate since it captures the significance of recent and thus relevant values, but also allows for some leeway when it comes to values that are usually active but for some reason have not been updated recently.

4.6.1 Adaptive Replacement Cache Policy

Since we did not want to consider all available opinions for the calculation of overall reputation values, we had to find an algorithm that could accommodate the need for selecting a subset of them, based on specific criteria. Frequency and recency seemed to be the main characteristics that could distinguish a reputation relation and corresponding value as important or not, hence our initial idea for *LRU* or *LFU* lists. Those kind of issues, however, are very common in the field of cache management, so we looked at related work to see if any of the proposed approaches could be applicable to our reputation system, especially ones that account for those characteristics or criteria mentioned earlier.

A lot of research has been done on cache management for different aspects of computer systems, namely storage systems, databases, web servers, middleware, operating systems etc. Since faster memory is significantly more expensive, its size is usually a fraction of the auxiliary memory. In case network communication is involved, the attempt is focused on minimizing network traffic. In either scenario, though, a policy for managing cache and minimizing swapping of pages is required.

Several approaches have been proposed, but one of the most prolific ones is the *Adaptive Replacement Cache (ARC)* policy [25]. This proposed policy attempts to bridge the gap between other policies, which are using lists with *Least Recently Used* or *Least Frequently Used*

pages, while maintaining low computational overhead.

The main idea is maintaining two lists to account for entries used *only once recently* and *at least twice recently*. Those lists are called $L1$ and $L2$ respectively. Whenever an entry appears that is not part of $L1 \cup L2$, it is added to the top of list $L1$. If, however, the entry already exists in $L1 \cup L2$, it is moved to the top of list $L2$.

If the cache can hold up to c number of entries, the combined size of $L1$ and $L2$ is equal to $2c$. The lists are further divided into *Top* and *Bottom* parts, with the *Top* parts having a combined size of c and representing the entries that are actually maintained by the cache memory. A visualization of that setting can be seen in Fig. 4.3 taken from [25].

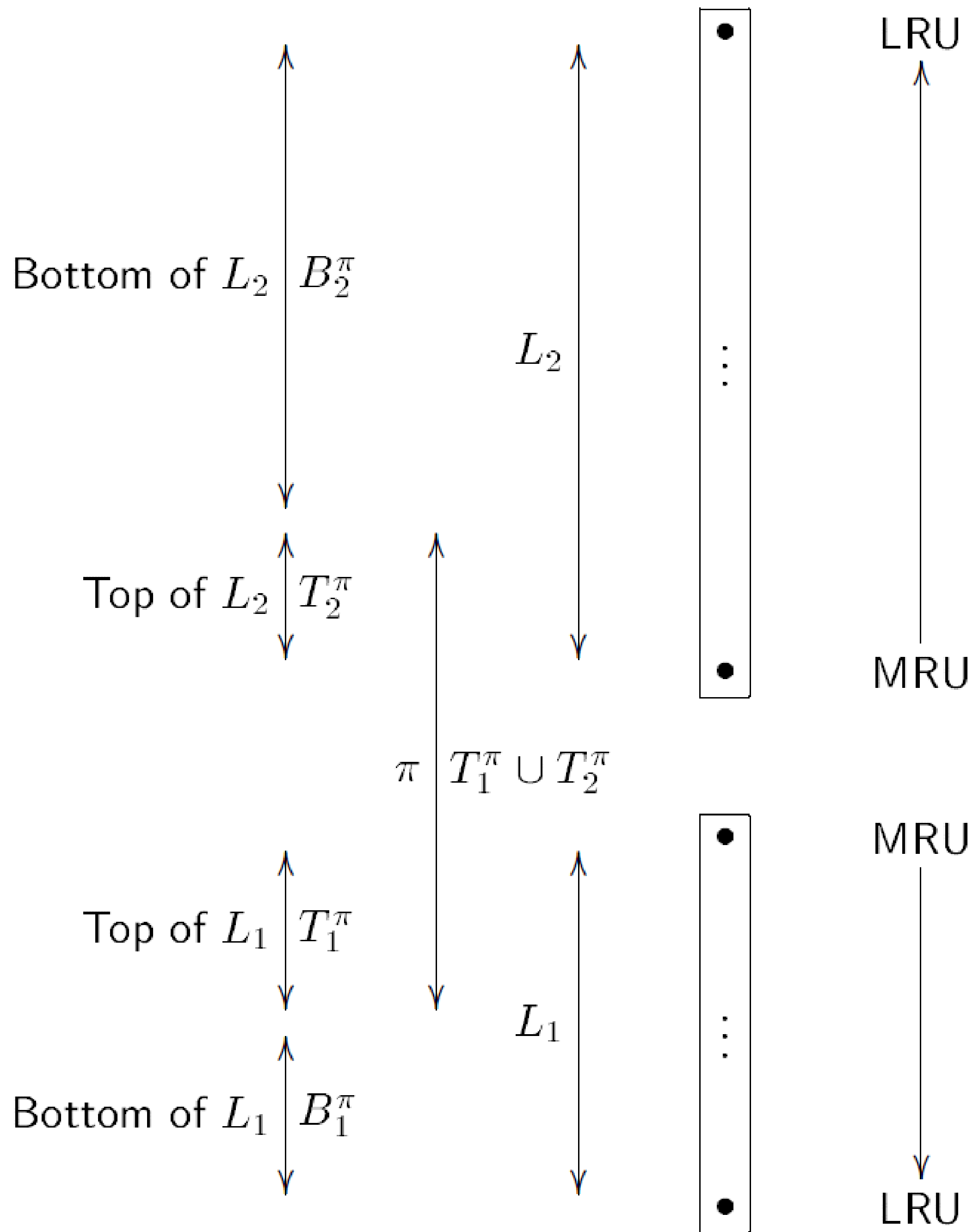

Figure 4.3: General Structure of ARC lists

Note here that, the entries in the cache are not equally divided between the *Top* parts of the two lists. The combined size always equals to *c*, but the policy adapts the partitioning based on observed workload. The main premise of the adaptation lies on ”investing” on the list that

performs better. So, a hit in $L1$ will increase the *Top* part of that list, whereas a hit in $L2$ will increase its *Top* part.

As far as the replacement policy goes, it mostly depends on whether the *Top* part of $L1$ has reached its desired size or not. Spots will be taken from $L2$, until the required size is reached.

By implementing that idea, *ARC* manages to outperform all other cache management policies, without burdening the system with unnecessary calculations.

4.6.2 ARC Adaptation

For our proposed approach, we adapt the Adaptive Replacement Cache (ARC) policy, in a way that fits the needs and requirements of our framework. Even though memory is not our main issue in this setting, discovering the R^r_w relations that are most important, instead of accounting for all available opinions, is paramount to us. Because of that, our algorithm utilizes the underlying ideas of the ARC policy and retrofits them to accommodate the functionality required by our reputation system, as discussed below.

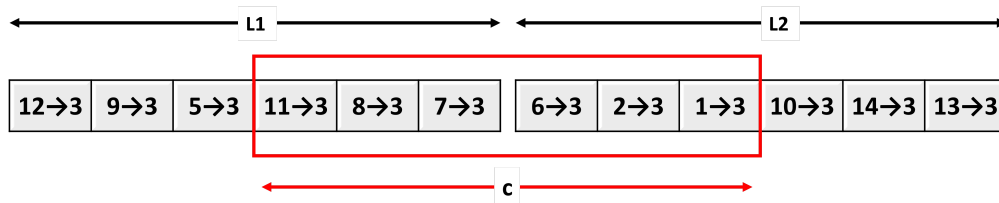

Figure 4.4: Example of lists for SC_3

Following the idea put forward by the ARC policy, two ordered lists are maintained for each recommender node r, which are used to identify the relations and corresponding reputation values (R^r_w and $R(w, r)$ respectively, where w are recommenders of r) that are to be used for the calculation of the *AR(r)* value of node r.

Algorithm 6 ARC Adaptation Algorithm

```
1: - Let r be the recommender node whose AR value is in question
2: - Let k be the recommender node who updated or deleted their recommendation for r.
3: - Let inc_r be the pair value that needs to be included in the calculation of the overall
   reputation value.
4: - Let exc_r be the pair value that needs to be excluded from the calculation of the overall
   reputation value.
5:
6: procedure addRValue(R(k,r), IMP_r, MFU_r, MRU_r)
7:     inc_r = R(k,r)
8:     if R(k,r) ∈ IMP_r then
9:         move R(k,r) to top of MFU_r
10:        exc_r = R_previous(k,r)
11:    else if R(k,r) ∈ MRU_r||MFU_r then
12:        move R(k,r) to top of MFU_r
13:        exc_r = value replaced by R(k,r)
14:        adjust IMP_r
15:    else
16:        move R(k,r) to top of MRU_r
17:        exc_r = value replaced by R(k,r)
18:        adjust IMP_r
19:    end if
20:    updateAR (inc_r, exc_r, sum(r), weightSum(r)) // Call Alg. 5.
21:
22:    // Or call Alg. 4 in case first variation is used (see comments in Alg. 4).
23:    // updateAR (IMP_r, R(k,r)|k ∈ IMP_r)
24: end procedure
25:
26: procedure removeRValue(R(k,r))
27:    if R(k,r) ∈ IMP_r then
28:        exc_r = R(k,r)
29:        if R(k,r) ∈ MRU_r then
30:            inc_r = next ∈ MRU_r
31:        else if R(k,r) ∈ MFU_r then
32:            inc_r = next ∈ MFU_r
33:        end if
34:        include inc_r in IMP_r
35:    end if
36:    updateAR (inc_r, exc_r, sum(r), weightSum(r)) // Call Alg. 5.
37:
38:    // Or call Alg. 4 in case first variation is used (see comments in Alg. 4).
39:    // updateAR (IMP_r, R(k,r)|k ∈ IMP_r)
40: end procedure
```

The first list (let's call it $\mathbf{MRU_r}$) is utilized to maintain the relations pertaining to the most recently updated reputation values put forward by other nodes, whereas the second one ($\mathbf{MFU_r}$ from now on) includes relations whose corresponding values have been updated most frequently. We, also, specify a list of size c that contains the top parts both those lists and accounts for the most important relations whose reputation values are to be considered in the calculation of the corresponding *AR(r)* value (we will call it $\mathbf{IMP_r}$). The process can be seen in Alg. 6.

As one can see, whenever a new *R(w,r)* value becomes available, the lists corresponding to the receiver of said value are checked. The available scenarios here are as follows:

- If the relation R_w^r, corresponding to the new reputation value *R(w,r)*, is in the top of either of the two lists (i.e. part of the IMP_r list), it is moved to the top of the MFU_r list. In this case, the previous value is, practically, substituted and the update consists of excluding the old and including the new value.

- If the relation R_w^r, corresponding to the new reputation value *R(w,r)* is part of either the MRU_r or the MFU_r list, but not part of the top, it is still moved to the top of the MFU_r list. However, a relation coming from another recommender is removed to make space for the new relation. The corresponding values are marked for exclusion and inclusion, respectively.

- If the relation R_w^r, corresponding to the new reputation value *R(w,r)* is not part of any of the lists, it becomes the top one in MRU_r and the associated value is deemed to be the one included in the calculation. Relation R_w^r replaces another relation R_k^r, whose corresponding reputation value *R(k,r)* is excluded from the calculation as a result.

Note that, the list with the values that are to be considered for the calculation of an *AR(r)* value (i.e. the IMP_r list) is comprised of the relations at the top of the two other lists and its size is fixed, but the amount of elements taken from each of the lists changes. You can see that the algorithm contains some pseudo-code indicating adjustment of the IMP_r list, but the actual logic is more complicated and can be found in [25]. Same thing goes for the replacement

logic, which is tightly coupled with the adjustment of the IMP_r list. The method was fully implemented for the prototype of our framework and the experimental results were acquired using that implementation.

Things are significantly more straightforward in case a *R(w,r)* relation is completely removed. The removed value is flagged to be removed from consideration for the calculation of the corresponding *AR(r)* value, if the corresponding relation R_w^r was part of the IMP_r list. Then, the next available reputation value *R(k,r)*, corresponding to the relation R_k^r that was part of the MRU_r or MFU_r list, based on where the removed relation belonged, but not of the IMP_r list, is included.

The final inc_r and exc_r values from each procedure are supplied as parameters to the algorithm presented in Alg. 5. [1] For performance purposes, several values can be accumulated over a predetermined period of time and sent as a single message to be processed by the proposed *AR* evaluation Algorithm 5. The period of time can be parameterized to account for better precision of values or improved performance.

4.7 Service Ranking

Every interaction between service client and service provider is preceded by acquisition of recommendations and compilation of a ranking of available services. The process includes selecting recommenders to be consulted, transforming recommendations into positive and negative evidence, applying ranking algorithm and, finally, inquiring for available incentives. All of the information are, then, presented to the service client, who selects a service to use, either manually or automatically through a pre-specified policy.

[1] If the first variation of the algorithm to update the *AR(r)* values is used (see Alg. 4), the alternative calls, supplying the IMP_r, are utilized.

4.7.1 Selection of Recommenders

When a user *p* requests a ranking of available services, ratings (i.e. *T(r,s)* and *D(r,s)* values) are requested by a subset of the recommenders participating in our framework. Specifically, prospective recommenders belong in one of the following 3 recommender groups:

1. Top *a%* of recommenders *r* to which *p* has a R_p^r relation and *R(p, r)* value (i.e. has received recommendations from them in the past). This is the group of *Friends* as defined in Section 3.4.

2. Top *a%* of recommenders *r* to which any recommender *k*, for who *p* has a R_p^k relation and *R(p, k)* value (i.e. has received recommendations from them in the past), has a R_k^r relation and *R(k, r)* value themselves. This is the group of *Friends of Friends* as defined in Section 3.4.

3. Top *a%* of recommenders *r* with the highest *AR(r)* value in the system. This is the group of *Experts* as defined in Section 3.4.

Note here that, there is an additional parameter introduced in the selection of recommenders. The parameter *a* can be set according to the user's preferences. A higher value of *a* will result in getting the opinion of more recommenders into account. The client can choose to get all of the opinions from one group, or a percentage of them. The proposed approach even supports setting a threshold in reputation values or number of recommenders. Any choice made, however, does not affect the semantics of the proposed approach. It merely adjusts the framework's throughput and network utilization vs plethora of available recommendations.

After selecting recommenders to consult, all available recommendations about service providers are obtained. Personal experience of *p* is, also, taken into consideration and ratings of service providers used in the past are obtained as well.

4.7.2 Aggregation of T and D values per recommender group

Even though the framework can support any ranking algorithm, we are proposing the use of a variation of the Dempster-Shafer evidence theory [22], where a final score for each recommended service, as a function of the *T(r,s)* and *D(r,s)* values that are provided by the different recommenders *r* involved (i.e. clients who have already used the service and have an opinion about the service), is calculated. The reason to consider Dempster-Shafer is because it naturally lends itself to the use of both positive and negative evidence for reaching a decision.

More specifically, each recommender *r* (i.e. clients who have already used the service and have an opinion about the service) provides a ⟨T(r,s),D(r,s)⟩ pair of values for each recommended service *s*. Even though each value supplied could be considered a distinct evidence, the Dempster-Shafer theory is heavily dependent on the plurality of evidences provided. This means that if values for a service are provided by a lot of recommenders belonging in one of the groups, but way fewer ones belonging in another, the outcome will be overwhelmingly affected by the former ones. This is especially common in cases of newer nodes, who don't yet have a lot of *Friends*. Also, the requesting user's personal experience accounts for only one set of evidences, which wouldn't really matter compared to the tens or even hundreds of opinions coming from other groups. For that reason, further preprocessing of the values is required.

For each one of the previous mentioned groups of recommenders and for each available service the *T(k,s)* and *D(k,s)* values are aggregated. The formulas are as follows:

$$T_{G \to s} = \frac{\sum_{k \in G}(Rep_k \cdot T(k,s))}{\sum_{k \in G}(Rep_k)} \tag{4.2}$$

$$D_{G \to s} = \frac{\sum_{k \in G}(Rep_k \cdot D(k,s))}{\sum_{k \in G}(Rep_k)} \tag{4.3}$$

In those formulas, Rep_k is the reputation value of every recommender of that group based on the group's nature (i.e. for *Friends* $Rep_k = R(p,k)$, for *Friends of Friends* $Rep_k = R(p,m) \cdot R(m,k)$, for *Experts* $Rep_k = AR(k)$) and ⟨T(k,s),D(k,s)⟩ are the values of each recommender

for every service available. The values produced by these formulas correspond to the rating of each service, as perceived by each group G (i.e. *Friends*, *Experts*, *FriendsOfFriends*). User's p personal opinions are used as is. This preprocessing allows for a single set of values coming from each group of recommenders for each service, with every recommender's opinion weighed differently within the group based on their corresponding reputation.

4.7.3 Dempster-Shafer

After the preprocessing, the ranking system considers every $T_{G \to s}$ and $D_{G \to s}$ value produced. Each $T_{G \to SP1}$ value, for example, is considered as an "in-favour" evidence for service *SP1*. This means that a set, containing only that service, is created (i.e. $\{SP1\}$) and is assigned the value calculated in the previous Section. The process is repeated for each of the recommender groups. Similarly, each $D_{G \to SP1}$ value is considered as an "against" evidence for service *SP1*, or complimentary as an "in-favour" evidence for the set of all available services except *SP1*. In this case, the complimentary set of the one created for $T_{G \to SP1}$ is constructed, or the one containing all available services except *SP1* (i.e. $\{SP1\}^C = \{SP2, SP3, SP4, ...\}$). The $D_{G \to SP1}$ value calculated in the previous Section is assigned to that set and the process is, again, repeated for all recommender groups. Bear in mind that, a group might not have any recommendations for that particular service, in which case no evidences are created.

These evidences are then aggregated using the Dempster-Shafer evidence theory algorithm to provide an overall belief interval for each recommended service, given the "in-favor" and "against" values for each service by each group of recommenders. If more significance needs to be assigned to the opinions of one of the groups, or the user's personal opinion, values produced by that group can be weighed accordingly, by duplicating the corresponding evidence, before they are utilized by the Dempster-Shafer algorithm.

More specifically, once the T and D values have been calculated for each available service and each group, using Formula 4.2 and Formula 4.3, certain values must be computed for each available service. The formulas for calculating those values utilized the aforementioned T and

D values available for that particular service and can be seen below:

$$p_s = \frac{(1 - \prod_{G \in \text{rec groups}}(1 - T_{G\to s})) \cdot \prod_{G \in \text{rec groups}}(1 - D_{G\to s})}{1 - (\prod_{G \in \text{rec groups}}(1 - T_{G\to s}) \cdot \prod_{G \in \text{rec groups}}(1 - D_{G\to s}))} \tag{4.4}$$

$$c_s = \frac{(1 - \prod_{G \in \text{rec groups}}(1 - D_{G\to s})) \cdot \prod_{G \in \text{rec groups}}(1 - T_{G\to s})}{1 - (\prod_{G \in \text{rec groups}}(1 - T_{G\to s}) \cdot \prod_{G \in \text{rec groups}}(1 - D_{G\to s}))} \tag{4.5}$$

$$r_s = 1 - p_s - c_s \tag{4.6}$$

$$d_s = c_s + r_s \tag{4.7}$$

Once those values have been computed for all of the recommended services, another value required by the Dempster-Shafer algorithm is computed. This K is required to calculate the belief interval and its formula is provided below:

$$K^{-1} = \prod_{sp \in \{\text{all SPs}\}} d_{sp} \cdot (1 + \sum_{sp \in \{\text{all SPs}\}} \frac{p_{sp}}{d_{sp}}) - \prod_{sp \in \{\text{all SPs}\}} c_{sp} \tag{4.8}$$

Once we have all the information, we can calculate the belief to the set containing just the service in question and the belief to the complimentary set, as well.

$$Bel(\{s\}) = K \cdot (p_s \cdot \prod_{sp \in \{s\}^C} d_{sp} + r_s \cdot \prod_{sp \in \{s\}^C} c_{sp}) \tag{4.9}$$

$$Bel(\{s\}^C) = K \cdot (\prod_{sp \in \{\text{all SPs}\}} d_{sp} \cdot \sum_{sp \in \{s\}^C} \frac{p_{sp}}{d_{sp}} + c_s \cdot \prod_{sp \in \{s\}^C} d_{sp} - \prod_{sp \in \{\text{all SPs}\}} c_{sp}) \tag{4.10}$$

Those two belief values are, finally, utilized to provide the belief interval for that particular service, as calculated using this variation of the Dempster-Shafer evidence theory.

$$s : [Bel(\{s\}), 1 - Bel(\{s\}^C)] \tag{4.11}$$

This belief interval's lower end corresponds to the result of the *Bel* function applied on the unit set of the service in question (i.e. $Bel(\{s\})$, see Formula 4.9). The result of the application of the belief function on the unit set of s gives us the total amount of belief committed to s, after all evidence bearing on it has been pooled. The upper end, on the other hand, comes from subtracting the result of the *Bel* function, applied on the complimentary of unit set of the service in question (i.e. $Bel(\{s\}^C)$, see Formula 4.9), from 1. $Bel(\{s\}^C)$ indicates the extent to which the evidence supports the negation of s (i.e. the belief that any other service is a better choice), with the result of $1 - Bel(\{s\}^C)$ expressing the plausibility of s (i.e. the extent to which evidence allows one to fail to doubt s).

After calculating the belief interval for each of the recommended services, we can sort them and provide the user with a ranking. Since this is an interval, and not a single value, one can sort them in a variety of ways. We propose ranking them based on the low end of the interval, which is the worst case for each service, with the upper end being utilized in case of a tie between two different services.

Note that, the belief values are not indicative of the recommendation values provided for each service, but rather the relative belief in a service, based on other available options, i.e. services. What this means is that, if the process has discovered three services that are all performing very well, each of them will bear a belief value of about 0.3.

Also, the Dempster-Shafer evidence theory is very unforgiving when it comes to negative evidence. This was one of the reasons that led to the utilization of this variation as the ranking algorithm for our approach. *Distrust*, in our system, accounts for participation of the service provider in behaviours that are considered unsolicited by the service client, so higher numbers should be heavily penalized, in our opinion.

4.7.4 Incentives

After each service is utilized by a number of service clients, and assuming that there are few or no malicious users, its actual QoS is available to everyone requesting it, either through

personal experience or through recommendations from other nodes in the framework. Even though that is highly desirable, it fails to address the issue of service providers improving the quality of the services they offer. Since users only pick the top service available at any point, improved services will never be utilized, and thus their score will never be updated. Furthermore, new services have no one to recommend them, since they have never been used in the past. Therefore, in this case they will never appear in the ranking of available services.

To mitigate those shortcomings, we propose that service providers are allowed to provide incentives to users in order to get the opportunity to demonstrate their improved or newly introduced performance. For example, a newly added to the system service may provide some ”bonuses” (i.e incentives) to the propsective clients, so that it can be chosen instead of another established service provider. As far as modelling incentives, a few approaches have been proposed. Most of them deal with incentives regarding honest behaviour by participating nodes [26], rather than incentives to prefer a service provider over another. Some, however, have proposed either specific models regarding price adaptation [27] or use of agent-based modelling to simulate behaviour and discover appropriate incentives per case [28]. Goal models could, also, be utilized to both describe conditions for providing incentives and specifying criteria for automatically accepting services and accompanying incentives by service clients. Original ranking of services will still be provided by taking recommendations and personal experience into consideration, but additional information about service-provided incentives will be available. The user can, then, choose, based on those information, whether they prefer the top rated service or any other service, whose incentives seem appealing to them.

4.7.5 Compensations

Another scenario that might occur is a service underperforming due to specific circumstances that have nothing to do with the service's regular QoS. Observed values for that interaction might be significantly lower than usual, which will lead to a decrease in the cumulative values (i.e. *T and D* values) maintained by the user. Said user will be more reluctant to use the

service in question in the future and will provide worse recommendation when inquired about it. To avoid this outcome, a service provider can opt to offer a compensation to the user. User can either accept the compensation and recalculate observed values, or decline it and proceed with the original values. For example, a temporarily underperforming service which overall is a trustworthy service but failed to meet the expected QoS, will be able to provide some compensations to the client (e.g. money, extra access to the service) so that the client will not rank the service as low. Consistent use of compensations by a service will indicate a degrading service to be reflected in the T and D values.

To model provided compensations based on difference between actual and desired behaviour, Dalpiaz et al.[29] have proposed an approach that utilizes and extends goal models and requirement engineering. Their method addresses the need for compensations in socio-technical systems, but it can be adapted and utilized for reconciliation and compensation in other service providing systems, such as the one proposed in this approach.

4.7.6 Running Example (Revisited)

In Chapter 3, we provided an example to demonstrate the steps involved in the process of getting recommendations and acting on them. However, no calculations were provided at this point, since the corresponding algorithms hadn't been presented yet.

To provide a complete and comprehensive insight into how the framework works, we revisit the exact same example and show the calculations required in some of the provided steps, using the algorithms presented in this chapter.

The same network of nodes, which can be seen again in Fig. 4.5, is utilized. The steps that do not involve calculations are included for clarity purposes, but in a summarized manner.

Once more, the process involves fourteen steps that are as follows:

Steps 1-4 The client expresses interest in using a service, selects recommenders belonging to each of the recommender groups (see Section 4.7.1) and collects their recommendations, adding its personal opinions to the pool.

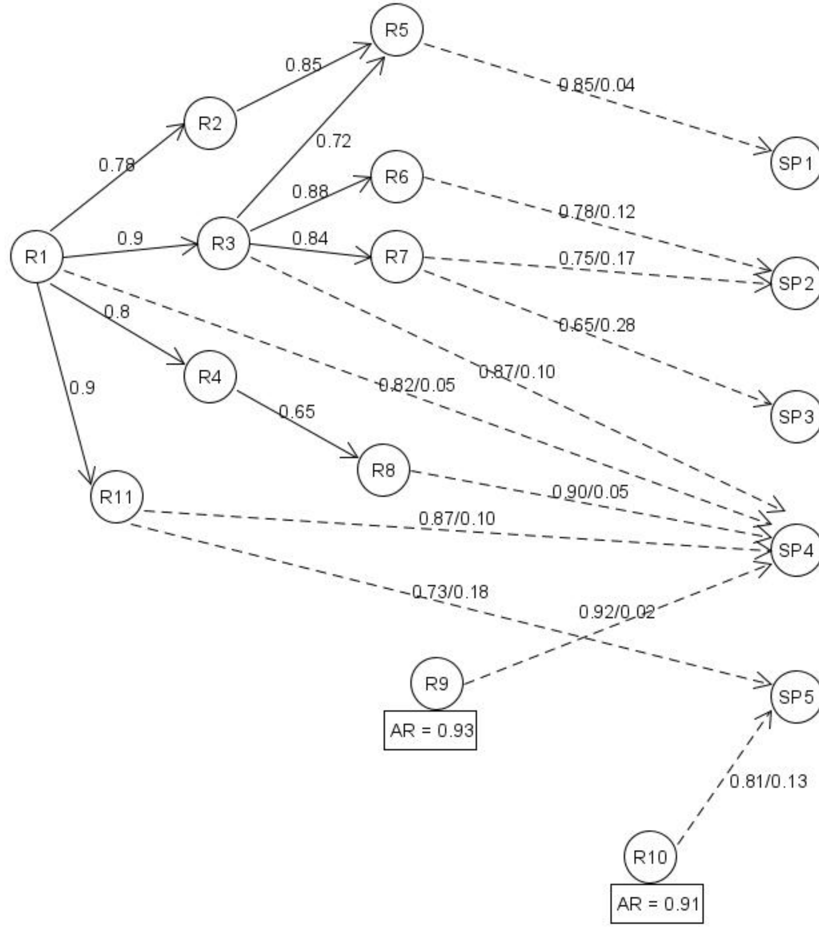

Figure 4.5: Example network of recommenders and services

Step 5 We need to calculate the aggregate values for each of the services and each group of recommenders (F → , FoF → *Friends of Friends*, E → *Experts*, P → *Personal*). So the calculations are as follows:

- ***SP2:*** For each group the T and D values are:
  - $$T_{SP2}^{FoF} = \frac{R(R1,R3)\cdot R(R3,R6)\cdot T(R6,SP2) + R(R1,R3)\cdot R(R3,R7)\cdot T(R7,SP2)}{R(R1,R3)\cdot R(R3,R6) + R(R1,R3)\cdot R(R3,R7)}$$
    $$T_{SP2}^{FoF} = \frac{0.9\cdot 0.88\cdot 0.78 + 0.9\cdot 0.84\cdot 0.75}{0.9\cdot 0.88 + 0.9\cdot 0.84} = 0.77$$
  - $$D_{SP2}^{FoF} = \frac{R(R1,R3)\cdot R(R3,R6)\cdot D(R6,SP2) + R(R1,R3)\cdot R(R3,R7)\cdot D(R7,SP2)}{R(R1,R3)\cdot R(R3,R6) + R(R1,R3)\cdot R(R3,R7)}$$
    $$D_{SP2}^{FoF} = \frac{0.9\cdot 0.88\cdot 0.12 + 0.9\cdot 0.84\cdot 0.17}{0.9\cdot 0.88 + 0.9\cdot 0.84} = 0.14$$
- ***SP3:*** For each group the T and D values are:

* $T_{SP3}^{FoF} = \frac{R(R1,R3) \cdot R(R3,R7) \cdot T(R7,SP3)}{R(R1,R3) \cdot R(R3,R7)} = \frac{0.9 \cdot 0.84 \cdot 0.65}{0.9 \cdot 0.84} = 0.65$
* $D_{SP3}^{FoF} = \frac{R(R1,R3) \cdot R(R3,R7) \cdot D(R7,SP3)}{R(R1,R3) \cdot R(R3,R7)} = \frac{0.9 \cdot 0.84 \cdot 0.26}{0.9 \cdot 0.84} = 0.26$

– ***SP4:*** For each group the T and D values are:

* $T_{SP4}^{F} = \frac{R(R1,R3) \cdot T(R3,SP4) + R(R1,R11) \cdot T(R11,SP4)}{R(R1,R3) + R(R1,R11)}$
  $T_{SP4}^{F} = \frac{0.9 \cdot 0.87 + 0.9 \cdot 0.87}{0.9 + 0.9} = 0.87$
* $D_{SP4}^{F} = \frac{R(R1,R3) \cdot D(R3,SP4) + R(R1,R11) \cdot D(R11,SP4)}{R(R1,R3) + R(R1,R11)}$
  $D_{SP4}^{F} = \frac{0.9 \cdot 0.10 + 0.9 \cdot 0.10}{0.9 + 0.9} = 0.10$
* $T_{SP4}^{E} = \frac{AR(R9) \cdot T(R9,SP4)}{AR(R9)} = \frac{0.93 \cdot 0.92}{0.93} = 0.92$
* $D_{SP4}^{E} = \frac{AR(R9) \cdot D(R9,SP4)}{AR(R9)} = \frac{0.93 \cdot 0.02}{0.93} = 0.02$
* $T_{SP4}^{P} = \frac{T(R1,SP4)}{1} = \frac{0.78}{1} = 0.82$
* $D_{SP4}^{P} = \frac{D(R1,SP4)}{1} = \frac{0.23}{1} = 0.05$

– ***SP5:*** For each group the T and D values are:

* $T_{SP5}^{F} = \frac{R(R1,R11) \cdot T(R11,SP5)}{R(R1,R11)} = \frac{0.9 \cdot 0.73}{0.9} = 0.73$
* $D_{SP5}^{F} = \frac{R(R1,R11) \cdot D(R11,SP5)}{R(R1,R11)} = \frac{0.9 \cdot 0.18}{0.9} = 0.18$
* $T_{SP5}^{E} = \frac{AR(R10) \cdot T(R10,SP5)}{AR(R10)} = \frac{0.91 \cdot 0.81}{0.91} = 0.81$
* $D_{SP5}^{E} = \frac{AR(R10) \cdot D(R10,SP5)}{AR(R10)} = \frac{0.91 \cdot 0.13}{0.91} = 0.13$

Step 6 With all the T and D values available, we can now run the Dempster-Shafer algorithm:

– The required *p,c,r,d* values have to be calculated for each recommended service.

* ***SP2:***
  · $p_{SP2} = \frac{(1-(1-T_{SP2}^{FoF})) \cdot (1-D_{SP2}^{FoF})}{1-((1-T_{SP2}^{FoF}) \cdot (1-D_{SP2}^{FoF}))} = 0.825$
  · $c_{SP2} = \frac{(1-(1-D_{SP2}^{FoF})) \cdot (1-T_{SP2}^{FoF})}{1-((1-D_{SP2}^{FoF}) \cdot (1-T_{SP2}^{FoF}))} = 0.04$
  · $r_{SP2} = 1 - p_{SP2} - c_{SP2} = 0.134$

· $d_{SP2} = c_{SP2} + r_{SP2} = 0.175$

∗ ***SP3:***

· $p_{SP3} = \frac{(1-(1-T^{FoF}_{SP3}))\cdot(1-D^{FoF}_{SP3})}{1-((1-T^{FoF}_{SP3})\cdot(1-D^{FoF}_{SP3}))} = 0.65$

· $c_{SP3} = \frac{(1-(1-D^{FoF}_{SP3}))\cdot(1-T^{FoF}_{SP3})}{1-((1-D^{FoF}_{SP3})\cdot(1-T^{FoF}_{SP3}))} = 0.12$

· $r_{SP3} = 1 - p_{SP3} - c_{SP3} = 0.23$

· $d_{SP3} = c_{SP3} + r_{SP3} = 0.35$

∗ ***SP4:***

· $p_{SP4} = \frac{(1-(1-T^{F}_{SP4})\cdot(1-T^{E}_{SP4})\cdot(1-T^{P}_{SP4}))\cdot((1-D^{F}_{SP4})\cdot(1-D^{E}_{SP4})\cdot(1-D^{P}_{SP4}))}{1-((1-T^{F}_{SP4})\cdot(1-T^{E}_{SP4})\cdot(1-T^{P}_{SP4})\cdot(1-D^{F}_{SP4})\cdot(1-D^{E}_{SP4})\cdot(1-D^{P}_{SP4}))} = 0.84$

· $c_{SP4} = \frac{(1-(1-D^{F}_{SP4})\cdot(1-D^{E}_{SP4})\cdot(1-D^{P}_{SP4}))\cdot((1-T^{F}_{SP4})\cdot(1-T^{E}_{SP4})\cdot(1-T^{P}_{SP4}))}{1-((1-D^{F}_{SP4})\cdot(1-D^{E}_{SP4})\cdot(1-D^{P}_{SP4})\cdot(1-T^{F}_{SP4})\cdot(1-T^{E}_{SP4})\cdot(1-T^{P}_{SP4}))} = 0.0003$

· $r_{SP4} = 1 - p_{SP4} - c_{SP4} = 0.162$

· $d_{SP4} = c_{SP4} + r_{SP4} = 0.16$

∗ ***SP5:***

· $p_{SP5} = \frac{(1-(1-T^{F}_{SP5})\cdot(1-T^{E}_{SP5}))\cdot((1-D^{F}_{SP5})\cdot(1-D^{E}_{SP5}))}{1-((1-T^{F}_{SP5})\cdot(1-T^{E}_{SP5})\cdot(1-D^{F}_{SP5})\cdot(1-D^{E}_{SP5}))} = 0.7$

· $c_{SP5} = \frac{(1-(1-D^{F}_{SP5})\cdot(1-D^{E}_{SP5}))\cdot((1-T^{F}_{SP5})\cdot(1-T^{E}_{SP5}))}{1-((1-D^{F}_{SP5})\cdot(1-D^{E}_{SP5})\cdot(1-T^{F}_{SP5})\cdot(1-T^{E}_{SP5}))} = 0.02$

· $r_{SP5} = 1 - p_{SP5} - c_{SP5} = 0.28$

· $d_{SP5} = c_{SP5} + r_{SP5} = 0.3$

– The K value must be calculated next.

$$K^{-1} = (d_{SP2} \cdot d_{SP3} \cdot d_{SP4} \cdot d_{SP5}) \cdot (1 + \frac{p_{SP2}}{d_{SP2}} + \frac{p_{SP3}}{d_{SP3}} + \frac{p_{SP4}}{d_{SP4}} + \frac{p_{SP5}}{d_{SP5}}) - (c_{SP2} \cdot c_{SP3} \cdot c_{SP4} \cdot c_{SP5})$$

* $$K^{-1} = (0.17 \cdot 0.35 \cdot 0.16 \cdot 0.3) \cdot (1 + \frac{0.83}{0.17} + \frac{0.65}{0.35} + \frac{0.84}{0.16} + \frac{0.7}{0.3}) - (0.04 \cdot 0.12 \cdot 0.0003 \cdot 0.02) = 0.044662$$

* $K = 1/K^{-1} = 22.39058$

– The belief intervals for the unit and complimentary sets of each service will be computed, as well.

* ***SP2:***

  · $$Bel(\{SP2\}) = K \cdot (p_{SP2} \cdot d_{SP3} \cdot d_{SP4} \cdot d_{SP5} + r_{SP2} \cdot c_{SP3} \cdot c_{SP4} \cdot c_{SP5}) = 22.39058 \cdot (0.83 \cdot 0.35 \cdot 0.16 \cdot 0.3 + 0.13 \cdot 0.12 \cdot 0.0003 \cdot 0.02) = 0.3132$$

  · $$Bel(\{SP2\}^C) = K \cdot (d_{SP2} \cdot d_{SP3} \cdot d_{SP4} \cdot d_{SP5} \cdot (\frac{p_{SP3}}{d_{SP3}} + \frac{p_{SP4}}{d_{SP4}} + \frac{p_{SP5}}{d_{SP5}}) + c_{SP2} \cdot d_{SP3} \cdot d_{SP4} \cdot d_{SP5} - c_{SP2} \cdot c_{SP3} \cdot c_{SP4} \cdot c_{SP5})$$
  $$Bel(\{SP2\}^C) = 22.39058 \cdot (0.17 \cdot 0.35 \cdot 0.16 \cdot 0.3 \cdot (\frac{0.65}{0.35} + \frac{0.84}{0.16} + \frac{0.7}{0.3}) + 0.04 \cdot 0.35 \cdot 0.16 \cdot 0.3 - 0.04 \cdot 0.12 \cdot 0.0003 \cdot 0.02) = 0.6358$$

  · Belief interval $\rightarrow [Bel(\{SP2\}), 1 - Bel(\{SP2\}^C)] = [0.3132, 0.3642]$

* ***SP3:***

  · $$Bel(\{SP3\}) = K \cdot (p_{SP3} \cdot d_{SP2} \cdot d_{SP4} \cdot d_{SP5} + r_{SP3} \cdot c_{SP2} \cdot c_{SP4} \cdot c_{SP5}) = 22.39058 \cdot (0.65 \cdot 0.17 \cdot 0.16 \cdot 0.3 + 0.23 \cdot 0.04 \cdot 0.0003 \cdot 0.02) = 0.12$$

$$Bel(\{SP3\}^C) = K \cdot (d_{SP2} \cdot d_{SP3} \cdot d_{SP4} \cdot d_{SP5} \cdot (\frac{p_{SP2}}{d_{SP2}} + \frac{p_{SP4}}{d_{SP4}} + \frac{p_{SP5}}{d_{SP5}}) + c_{SP3} \cdot d_{SP2} \cdot d_{SP4} \cdot d_{SP5} - c_{SP2} \cdot c_{SP3} \cdot c_{SP4} \cdot c_{SP5})$$

· $$Bel(\{SP3\}^C) = 22.39058 \cdot (0.17 \cdot 0.35 \cdot 0.16 \cdot 0.3 \cdot (\frac{0.83}{0.17} + \frac{0.84}{0.16} + \frac{0.7}{0.3}) + 0.12 \cdot 0.17 \cdot 0.16 \cdot 0.3 - 0.04 \cdot 0.12 \cdot 0.0003 \cdot 0.02) = 0.8113$$

· Belief interval → $[Bel(\{SP3\}), 1 - Bel(\{SP3\}^C)] = [0.12, 0.1887]$

* ***SP4:***

· $$Bel(\{SP4\}) = K \cdot (p_{SP4} \cdot d_{SP2} \cdot d_{SP3} \cdot d_{SP5} + r_{SP4} \cdot c_{SP2} \cdot c_{SP3} \cdot c_{SP5}) = 22.39058 \cdot (0.84 \cdot 0.17 \cdot 0.35 \cdot 0.3 + 0.16 \cdot 0.04 \cdot 0.12 \cdot 0.02) = 0.342$$

$$Bel(\{SP4\}^C) = K \cdot (d_{SP2} \cdot d_{SP3} \cdot d_{SP4} \cdot d_{SP5} \cdot (\frac{p_{SP2}}{d_{SP2}} + \frac{p_{SP3}}{d_{SP3}} + \frac{p_{SP5}}{d_{SP5}}) + c_{SP4} \cdot d_{SP2} \cdot d_{SP3} \cdot d_{SP5} - c_{SP2} \cdot c_{SP3} \cdot c_{SP4} \cdot c_{SP5})$$

· $$Bel(\{SP4\}^C) = 22.39058 \cdot (0.17 \cdot 0.35 \cdot 0.16 \cdot 0.3 \cdot (\frac{0.83}{0.17} + \frac{0.65}{0.35} + \frac{0.7}{0.3}) + 0.0003 \cdot 0.17 \cdot 0.35 \cdot 0.3 - 0.04 \cdot 0.12 \cdot 0.0003 \cdot 0.02) = 0.5921$$

· Belief interval → $[Bel(\{SP4\}), 1 - Bel(\{SP4\}^C)] = [0.342, 0.4079]$

* ***SP5:***

· $$Bel(\{SP5\}) = K \cdot (p_{SP5} \cdot d_{SP2} \cdot d_{SP3} \cdot d_{SP4} + r_{SP5} \cdot c_{SP2} \cdot c_{SP3} \cdot c_{SP4}) = 22.39058 \cdot (0.7 \cdot 0.17 \cdot 0.35 \cdot 0.16 + 0.28 \cdot 0.04 \cdot 0.12 \cdot 0.0003) = 0.1564$$

$$Bel(\{SP5\}^C) = K \cdot (d_{SP2} \cdot d_{SP3} \cdot d_{SP4} \cdot d_{SP5} \cdot (\frac{p_{SP2}}{d_{SP2}} + \frac{p_{SP3}}{d_{SP3}} + \frac{p_{SP4}}{d_{SP4}}) + c_{SP5} \cdot d_{SP2} \cdot d_{SP3} \cdot d_{SP4} - c_{SP2} \cdot c_{SP3} \cdot c_{SP4} \cdot c_{SP5})$$

· $$Bel(\{SP5\}^C) = 22.39058 \cdot (0.17 \cdot 0.35 \cdot 0.16 \cdot 0.3 \cdot (\frac{0.83}{0.17} + \frac{0.65}{0.35} + \frac{0.84}{0.16}) + 0.02 \cdot 0.17 \cdot 0.35 \cdot 0.16 - 0.04 \cdot 0.12 \cdot 0.0003 \cdot 0.02) = 0.7808$$

· Belief interval → $[Bel(\{SP5\}), 1 - Bel(\{SP5\}^C)] = [0.1564, 0.2192]$

– Finally, a ranking of the services will be created and supplied to the user. The ordered list is $[SP4, SP2, SP5, SP3]$

Steps 7-8 The client does not consider any incentives in this example and ends up using the top ranked service, which is *SP4*.

Step 9 After using *SP4*, the user receives information through the proxy node, representing the service within the framework, and evaluates the corresponding goal models, or whichever method of service evaluation they have selected. Let's assume this interaction provided the user with the following observed values:

– $OT(R1, SP4) = 0.89$

– $OD(R1, SP4) = 0.07$

Steps 10-11 Since the interaction proceeded according to plan, no compensations were offered to be considered by the client.

Step 12 Given the observed through the interaction values, client *R1* has to update their personal opinion about service *SP4*. This is done using Alg. 1. Since *R1* already has a value for *SP4*, there is no need for setting a default one. Let's also, assume that $c = 0.1$ and $count = 3$. So the calculations involved are:

$$T_{offset}(R1, SP4) = c(OT(R1, SP4) - T(R1, SP4) = 0.1 \cdot (0.89 - 0.81)) = 0.008$$

$$T(R1, SP4) = T(R1, SP4) + T_{offset}(R1, SP4) = 0.81 + 0.008 = 0.818$$

$$D_{offset}(R1, SP4) = c(OD(R1, SP4) - D(R1, SP4) = 0.1 \cdot (0.07 - 0.05)) = 0.002$$

$$D(R1, SP4) = D(R1, SP4) + D_{offset}(R1, SP4) = 0.05 + 0.002 = 0.052$$

$$count(R1, SP4) = count(R1, SP4) + 1 = 4$$

Step 13 Client *R1* has to, also, update the reputation values for each of the recommenders of *SP4*.

The offset calculation follows the same logic for each of them, but, based on the group they belong to, the baseline will be different. The recommenders and reputation values updates are as follows:

- Recommenders selected as *Friends*:
  - ***R3:***
    - $$\begin{aligned} error(R1,R3) &= |OT(R1,SP4) - T(R3,SP4)| + |OD(R1,SP4) - D(R3,SP4)| \\ &= 0.02 + 0.03 = 0.05 \end{aligned}$$
    - $R_{offset}(R1,R3) = 0.08$
    - $R(R1,R3) = R(R1,R3) + R_{offset}(R1,R3) = 0.9 + 0.08 = 0.98$
    - $count(R1,R3) = count(R1,R3) + 1 = 5$
  - ***R11:***
    - $$\begin{aligned} error(R1,R11) &= |OT(R1,SP4) - T(R11,SP4)| + |OD(R1,SP4) - D(R11,SP4)| \\ &= 0.02 + 0.03 = 0.05 \end{aligned}$$
    - $R_{offset}(R1,R11) = 0.08$
    - $R(R1,R11) = R(R1,R11) + R_{offset}(R1,R11) = 0.9 + 0.08 = 0.98$
    - $count(R1,R11) = count(R1,R11) + 1 = 2$
- Recommenders selected as *Experts*:
  - ***R9:***
    - $$\begin{aligned} error(R1,R9) &= |OT(R1,SP4) - T(R9,SP4)| + |OD(R1,SP4) - D(R9,SP4)| \\ &= 0.03 + 0.05 = 0.08 \end{aligned}$$
    - $R_{offset}(R1,R9) = 0.05$
    - $R(R1,R9) = AR(R9) + R_{offset}(R1,R9) = 0.9 + 0.05 = 0.95$
    - $count(R1,R9) = 1$

Step 14 Since we have new R values for three recommenders, namely $[R3,R9,R11]$, we need to first run Alg. 6 to discover which values the new ones will be replacing, if any.

- Since *R1* has *R* relations with *R3* and *R11*, we can assume, for the sake of this example, that the reputation value *R1* has for them is already considered and the new one will just substitute the previous one.
- Let's also assume that *R1*'s previous overall reputation was 0.82, while the current one is 0.84
- Finally, if both *R3* and *R11* had an *AR* value of 0.87 and the sum and weightSum were 7.308 and 8.4 respectively, the calculations would be as follows:
  * $$\begin{aligned} sum(R3) &= sum(R3) - (R_{previous}(R1, R3) \cdot AR_{previous}(R1)) \\ &= 7.308 - (0.9 \cdot 0.82) = 6.57 \end{aligned}$$
  * $$\begin{aligned} weightSum(R3) &= weightSum(R3) - AR_{previous}(R1) \\ &= 8.4 - 0.82 = 7.58 \end{aligned}$$
  * $$\begin{aligned} sum(R3) &= sum(R3) + (R(R1, R3) \cdot AR(R1)) \\ &= 6.57 + (0.98 \cdot 0.84) = 7.3932 \end{aligned}$$
  * $$\begin{aligned} weightSum(R3) &= weightSum(R3) + AR(R1) \\ &= 7.58 + 0.84 = 8.42 \end{aligned}$$
  * $AR(R3) = sum(R3)/weightSum(R3) = 7.3932/8.42 = 0.878$
- The same exact calculations are performed for *R11*. As far as *R9* goes, the values subtracted for the sums will change, depending on what values are substituted, based on the algorithm presented in Section 4.6, but the process remains the same.

4.8 Discussion on Self-Regulating Behaviour of Recommenders

With respect to the behaviour of users within the system, they can be categorized in one of a number of possible scenarios. Those scenarios include honest and dishonest behaviours and some considerations regarding the system's ability to self regulate in any of these occasions can be found below.

Case 1: *Recommenders r who recommend correctly to requester p*. The *R(p,r)* value from *p* to these recommenders *r* will increase whenever *p* ends up acting upon *r*'s recommendation. This will trigger the re-calculation of *r*'s *AR* value, causing it to increase, making it an even more reputable recommender.

Case 2: *Malicious or erroneous recommenders r*. These recommenders *r* may maliciously degrade their recommendation towards a service s, to a node p1 seeking recommendation. However, other nodes *r1, r2, r3, . . .* , who are not malicious may also be providing a good recommendation for *s* to *p1*, which in this case *p1* may decide to use *s*. If *p1* decides to use *s*, then a discrepancy between *r*'s recommendation and *p1*'s observed value of *s*'s QoS will occur, thus reducing *p1*'s node-to-node recommendation strength *R* towards *r*, causing *r*'s *AR* to be reduced as well. The denser the network becomes the higher the chances that *p1* will be connected to other nodes *r1, r2, r3, . . .* , and not only *r*, thus exposing *r*'s malicious behavior. The same idea (dual) holds if a malicious recommender increases its recommendation for a non-trustworthy service.

Case 3: *Service client r who observes correctly and calculates T and D values correctly*. When these clients *r* are asked for their recommendation by a potential service client *p*, they will provide a correct recommendation, and thus the *R* value towards them by *p* will increase, causing *r*'s *AR* to increase, making *r* a reputable future recommender. Then this becomes equivalent to *Case 1* above.

Case 4: *Malicious service client r who observe and calculate T, and D values erroneously*. When these clients *r* are asked for their recommendation by a potential service client *p*, they will provide an incorrect recommendation, and thus the *R* value towards them by *p* will decrease, causing *r*'s *AR* to decrease, making *r* a non-reputable future recommender. Then this becomes equivalent to *Case 2* above.

4.9 Summary

To summarize, for a service to be selected the client receives an ordered list that represents the perceived ranking of services. For this to happen, several steps have to be taken.

First, the appropriate set of recommenders to be consulted has to be created. Several sources of information are considered, and the recommenders that are thought as the most influential within each group are selected and added to the aforementioned set (see Section 4.7.1).

The recommendations of the recommenders selected in the previous step are collected, as well as the personal opinions of the requesting user. The recommendations are separated, placed in different ”buckets”, corresponding to each recommender group, and eventually aggregated per group, as described in Section 4.7.2, to produce a pair of positive and negative evidences for each recommended service and each group that has an available recommendation for said service.

The evidences are, then, utilized to produce a ranking. Any ranking algorithm can be used, but we propose the use of a variation of the Dempster-Shafer evidence theory, since it accounts for both positive and negative evidence and computes belief intervals that are relative to the other available options within our framework (see Section 4.7.3). Specific values for each service are calculates as part of this algorithm, as well as values that correspond to all recommended services. Those values are, then, used to calculate the aforementioned belief intervals for each service and, ultimately, provide a ranking to the user.

Incentives...

After choosing and using a specific service, a pair of observed values (i.e. observed trust/distrust) is obtained either manually, as provided by the user, or automatically, utilizing the information collected through the proxy service node and evaluating a goal model or any other kind of model. Those values are used to update a) the cumulative *T/D* values the user maintains for that service and b) the individual reputation values of the recommenders who recommended said service. The algorithms described in Section 4.2 and Section 4.3 are utilized respectively.

Last but not least, a subset of the available *R* values are used to calculate a recommender's overall reputation, i.e. *AR* value. The proposed method for selecting the most important values is presented in Section 4.6 and the algorithm to update the overall reputation value of a recommender, when a *R* value belonging to the set gets an update or a new one is added to it, is explained in Section 4.5. We propose two algorithms for calculating an *AR* value. The first one captures the essence of the approach by performing all the required calculations every time the overall reputation value is requested, whereas the second one utilizes a more event-driven way of computing the value, by maintaining the corresponding sums and adding or subtracting only what is necessary.

Chapter 5

System Architecture

5.1 Centralized Architecture

The framework can be deployed in a centralized manner whenever a central authority or organization, such as an e-commerce platform, requires the reputation system to be utilized in tandem to other applications. In this scenario, the authority has full control over the data and operations and only offers the clients the ability to request certain functionality and use services through the already deployed framework. Such a deployment allows the integration of the proposed approach to other offered services.

5.1.1 Architecture Overview

The centralized version of the proposed framework consists of several components, with each of them performing a different task crucial to the system. Those components can be deployed in a single server, or as micro-services in separate machines, but they remain under the supervision of a central authority. Each service client runs just a small client program that is tasked with performing the REST calls to the main framework and requesting any of the available options. A description of each component and sub-component is provided in this section. The system is comprised of the following components:

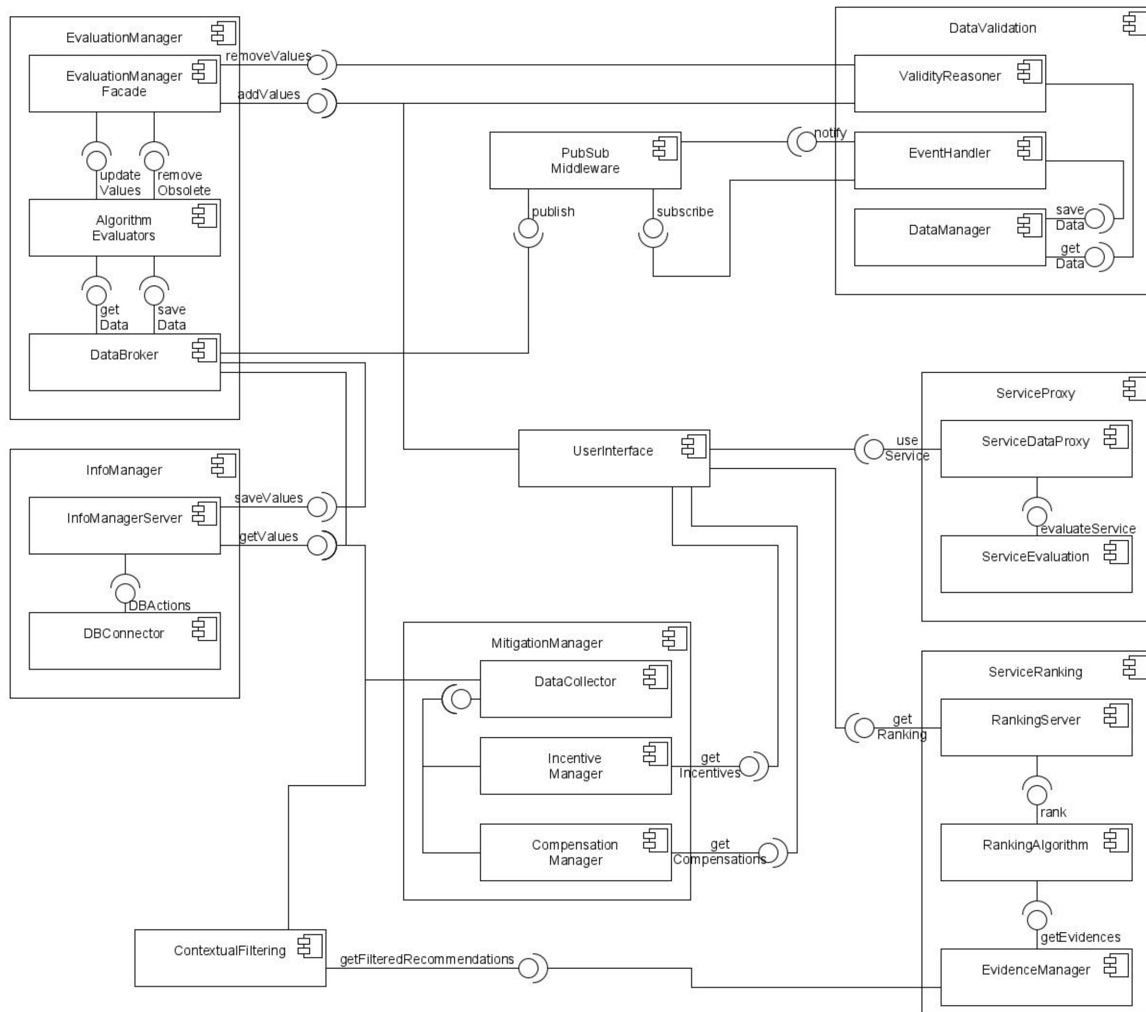

Figure 5.1: Component diagram for centralized architecture.

- **InfoManager:** This component is tasked with maintaining the information regarding all relationships between recommenders and service providers (see Sections 3.3.2, 3.3.3, 3.3.4). Every relation pertaining to the social graph maintained by the system is stored in this component.

  – ***InfoManagerServer:*** This sub-component allows for the services of the InfoManager component to be offered to other components. A standalone server is operated as part of this sub-component, including the corresponding error handling of REST calls. Unmarshalling of the messages is, also, performed by this part of the system

for the information to be utilizable by other components.

- ***DBConnector:*** The connectors and error handling for the required databases are included in this sub-component. Several low level checks regarding the incoming and outgoing values are, also, performed here. More specifically, duplicate values are discarded and entries with a count of zero are removed altogether, to signify that the connection in the social graph is no longer present.

- **EvaluationManager:** This component provides the functionality of reevaluating the values corresponding to the relationship between different entities. Any update required for *T/D*, *R* or *AR* values is handled by this component.

  - ***EvaluationManagerFacade:*** The API of the main component is offered through this sub-component. Again, a server is responsible for handling incoming REST calls and dealing with proocol errors and unmarshalling of data.
  - ***AlgorithmEvaluators:*** The actual implementations of the evaluation algorithms are contained here (see Sections 4.2, 4.3, 4.5). The calculation of each required value update is assigned to a different thread, in order to improve performance, and the produced values are, then, forwarded to other interested components for saving or further utilization.
  - ***DataBroker:*** This sub-component is utilized by the AlgorithmEvaluators sub-component to handle any needs for sending or receiving any information relevant to the evaluation algorithms. Its functionality includes communicating with the *InfoManager* component to a) obtain the information required when a evaluation of a relationship value is needed or b) save the values updated using the algorithms. This sub-component is, also, responsible with publishing the calculated *offset* and *R* values for them to be used by interested components (see Section 4.4 and Section 4.6)

- **ServiceRanking:** A ranking of available services is provided by this component to any requesting user, based on the opinions of other users participating in the framework.

Recommenders are selected, recommendations are collected and transformed into evidences and the chosen ranking algorithm is executed. Different ranking algorithms can be utilized and the choices regarding the consulted recommenders can be parameterized.

  - ***RankingServer:*** This sub-component provides the ranking service to the other components. A server is run to provide said ranking service in a RESTful way and handle any upcoming communication errors.
  - ***EvidenceManager:*** This sub-component allows for the selection of recommenders that fulfill certain criteria (see Section 3.4). *Trust* and *distrust* values are collected from selected recommenders and are then transformed into evidence to be considered by the *RankingAlgorithm* sub-component.
  - ***RankingAlgorithm:*** The ranking algorithm is implemented as part of this sub-component. All available evidence are obtained from the *EvidenceManager* sub-component and the algorithm is executed to produce the ranking. The algorithm utilized at the moment is a modified version of Dempster-Shafer (see Section 4.7.3). To further improve the system's throughput, each request for ranking is handled by a separate thread.

- **ServiceProxy:** A proxy for using a service and, subsequently, obtaining data and evaluating said service is implemented here. The service client can only use the selected service through the corresponding proxy.
  - ***ServiceDataProxy:*** The actual call to the chosen service is performed through this sub-component. Metrics of interest are automatically collected from the interaction,as well as the user's review. Different implementations regarding data gathering can be included in this sub-component, as well as GUIs to acquire the client's review immediately following the interaction.
  - ***ServiceEvaluation:*** Obtained information are provided by the *ServiceDataProxy* sub-component and the service is evaluated using the selected method or algorithm.

For our prototype implementation, we propose the use of extended goal models and fuzzy reasoning as defined in [24]. In each case, the appropriate goal model is evaluated to provide the observed *trust* and *distrust* values for the utilized service (see Section 3.3.1).

- **DataValidation:** This component is tasked with two distinct responsibilities: a) It keeps track and notifies the system of previously observed trust and distrust values and the corresponding fluctuations of recommender values that have become obsolete, and b) it maintains a list of recommender values that contribute to the overall reputation of a recommender, based on an algorithm specified in 4.6. In both cases, the *EvaluationManager* component is notified to update the corresponding values.

  – ***EventHandler:*** This sub-component is responsible with subscribing to the topics that correspond to the values of interest. It, also, provides call-back methods to be called when new data become available. Said methods receive the published data, timestamp them and transform them in a format that is appropriate for saving by the *DataManager* sub-component.

  – ***ValidityReasoner:*** The algorithm used for deciding the recommender values to be considered for the overall reputation of a recommender (see Section 4.6) is implemented here. Note that the sub-component is structured in a way that different algorithms can be utilized if required. The logic for discovering obsolete values is, also, implemented here. All of those processes are independent to the main process, so separate threads are utilized to ensure that they run uninterrupted and without delays.

  – ***DataManager:*** The data required by the ValidityReasoner are handled by this sub-component. A distributed database solution is used to allow for replication of this component. Data are inserted by the *EventHandler* sub-component or retrieved by the *ValidityReasoner* sub-component.

- **MitigationManager:** This component evaluates mitigation strategies that may be offered by specific services. The component, corresponding to the client requesting a service, communicates with this component both during the ranking process and after the service utilization to inquire for available incentives and compensations respectively.

  - ***IncentiveManager:*** Incentives may be provided by a service that is new or has performed poorly in the past. Said incentives are calculated using models specified in this sub-component. After the original ranking is provided, a request is made to this sub-component to provide them to the client. A server is included in this component, so that the information can be accessed through a REST API.

  - ***CompensationManager:*** Compensations may be given by a service in case of an interaction that didn't perform as expected. The models utilized to calculate them are part of this sub-component. This component is consulted for available compensations after the client has chosen and actually used a specific service.

  - ***DataCollector:*** If complimentary data are required for the evaluation of incentive or compensation models, the DataCollector is utilized. Data are collected from the *InfoManager* component to be provided to any models that might need them. Potential needs may include knowledge of past performance or current reputation.

- **ContextualFiltering:** Filtering of available services is performed here, based on contextual information or specific ontologies, before evidence is collected by the ServiceRanking component. No filtering is performed as part of the prototype implementation, but we are proposing an architecture that can easily be extended and enhanced.

- **PubSubMiddleware:** This a middleware framework incorporated in our system to allow decoupling communication between different components. Instances or replicas of the *EvaluationManager* component publish the calculated *offsets* and *R* values and they are received by subscribing *DataValidation* instances.

- **UserInterface:** Each user participates and interacts with the framework through this component. The service client has access to all the functionality offered by the system and is assigned a specific ID and corresponding reputation based on interactions performed using this component.Specific implementation could include a graphical UI or an API to be consumed by other applications.

5.1.2 Interface Specification

In this Section, we provide a simple and short description of the interfaces offered by each component as seen in Fig. 5.1.1. Note that each interface usually consists of multiple operations, but mentioning all of them here wouldn't really provide any further insight on the framework's functionality, so they are omitted. Said interfaces and corresponding operations are the ones used in the next Section to describe the sequence of actions between components.

Component Name	Interface Name	Description
InfoManagerServer	*getValues*	This interface is utilized to retrieve *T/D*, *R* and *AR* values.
	saveValues	This interface is utilized to save updated *T/D*, *R* and *AR* values.
DBConnector	*DBActions*	This interface is utilized to perform actions on the utilized database.
EvaluationManager Facade	*addValues*	This interface is utilized to update *T/D*, *R* and *AR* values after a new interaction.
	removeValues	This interface is utilized to remove obsolete offsets pertaining to *T/D* and *R* values.
AlgorithmEvaluators	*updateValues*	This interface is utilized to run the algorithms used for updating *T/D*, *R* and *AR* values after an interaction.
	removeObsolete	This interface is utilized to run the algorithms for removing offsets for *T/D* and *R* values.
DataBroker	*getData*	This interface is utilized to request *T/D*, *R* and *AR* values.
	saveData	This interface is utilized to save updated *T/D*, *R* and *AR* values.
EventHandler	*notify*	This interface is utilized to notify the component of new offsets and updated *R* values.

Component Name	Interface Name	Description
DataManager	*getData*	This interface is utilized to request saved offsets and *R* values.
	saveData	This interface is utilized to save offsets and information about updates in *R* values.
ServiceDataProxy	*useService*	This interface is utilized to use the chosen service through a proxy and get interaction information.
ServiceEvaluation	*evaluateService*	This interface is utilized to evaluate the services's QoS using the interaction information.
RankingServer	*getRanking*	This interface is utilized to request a ranking of available services based on recommendations.
RankingAlgorithm	*rank*	This interface is utilized to run the algorithm required to rank services.
EvidenceManager	*getEvidences*	This interface is utilized to acquire recommendations and create the corresponding evidences.
ContextualFiltering	*getFiltered Recommendations*	This interface is utilized to receive recommendations, but filtered based on specific context.
IncentiveManager	*getIncentives*	This interface is utilized to inquire about available incentives offered by services.
CompensationManager	*getCompensations*	This interface is utilized to inquire about available compensations offered by a service after an interaction.
DataCollector	*getData*	This interface is utilized to get any information required for the evaluation of incentives or compensations.
PubSubMiddleware	*publish*	This interface is utilized to publish offsets and *R* values.
	subscribe	This interface is utilized to subscribe to specific offsets and information about updates in *R* values.

5.1.3 Process Sequence Diagrams

To further explain the process required to acquire recommendations and use a service as a service client, as well as the independent processes run to discover obsolete values and decide on important values for the calculation of the overall reputation values, we provide a set of sequence diagrams with corresponding descriptions. For each of the diagrams, we specify the steps of the process, described in Section 3.5.1, it corresponds to. We, also, describe the steps in the sequence diagram to clarify the communication required between the components

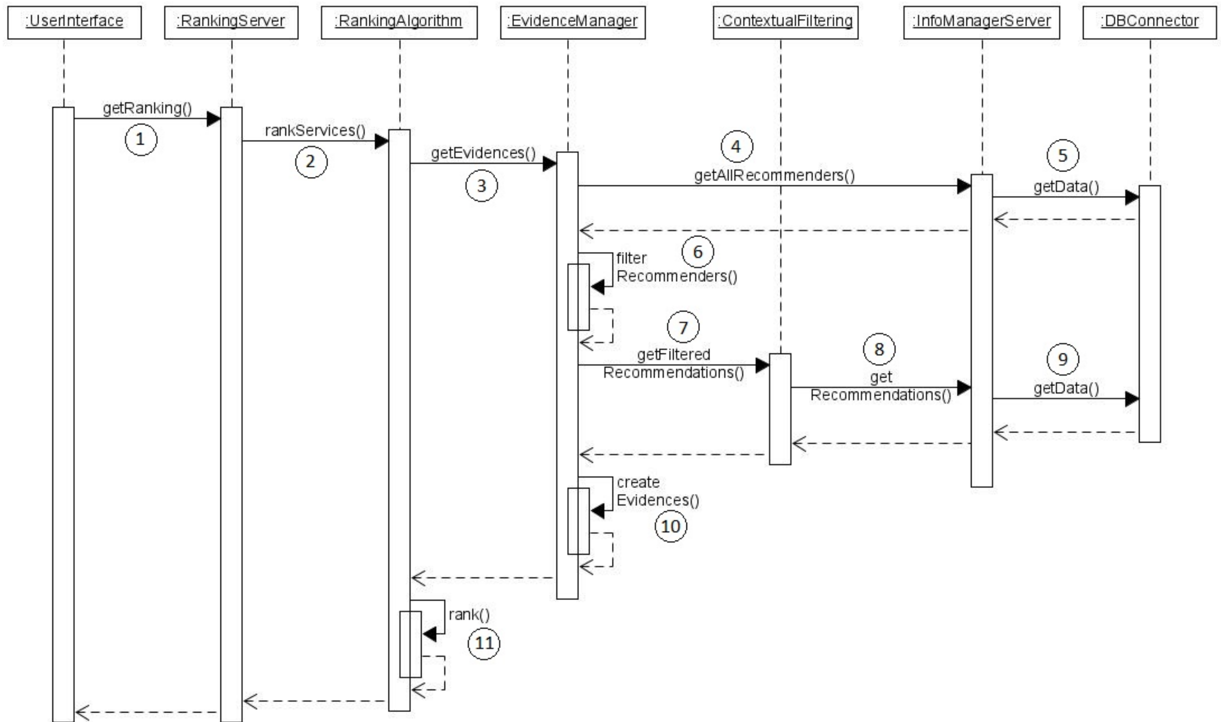

Figure 5.2: Centralized Architecture: Sequence diagram for ranking part.

specified in Section 5.1.1.

Recommendations and Ranking

The sequence described in this section corresponds to the request for recommendations by a service client, followed by the selection of recommenders to consult, acquisition of recommendations and, finally, transformation to evidences and calculation of service ranking. Said sequence accounts for Steps 1-6 of the process described in Section 3.5.1 and can be observed in Fig. 5.2.

To perform this sequence of actions the framework takes the following steps:

1. The *UserInterface* component corresponding to the requesting user asks for a ranking of available services from the *RankingServer*.

2. The *RankingServer* forwards the request to the utilized *RankingAlgorithm* component.

3. The *RankingAlgorithm* makes a request for available evidences to the *EvidenceManager*.

4. The *EvidenceManager* requests all available recommenders, belonging to each of the sources of recommendation (see Section 3.4), from the *InfoManagerServer*.

5. The *InfoManagerServer* queries the accompanying *DBConnector* component, after creating the appropriate queries.

6. After all the available information has been returned to the *EvidenceManager*, recommenders are filtered based on the logic described in Section 4.7.1.

7. Recommendations from selected recommenders are requested, filtered by specific context, such as location, service type etc. Note that, no filtering was performed in the prototype implementation, but the architecture supports the functionality.

8. All available recommendations from selected recommenders are requested from the *InfoManagerServer*.

9. Again, the *InfoManagerServer* queries the accompanying *DBConnector* component, after creating the appropriate queries.

10. Once the recommendations have been acquired by the *EvidenceManager*, they are grouped in ”buckets”, as explained in Section 4.7.2, and the evidences are created.

11. The *RankingAlgorithm* receives the produced evidences and performs the ranking using the chosen algorithm. In our prototype, the algorithm used is a variation of the Dempster-Shafer evidence theory, described in Section 4.7.3. The final ranking is returned to the requesting *UserInterface* component.

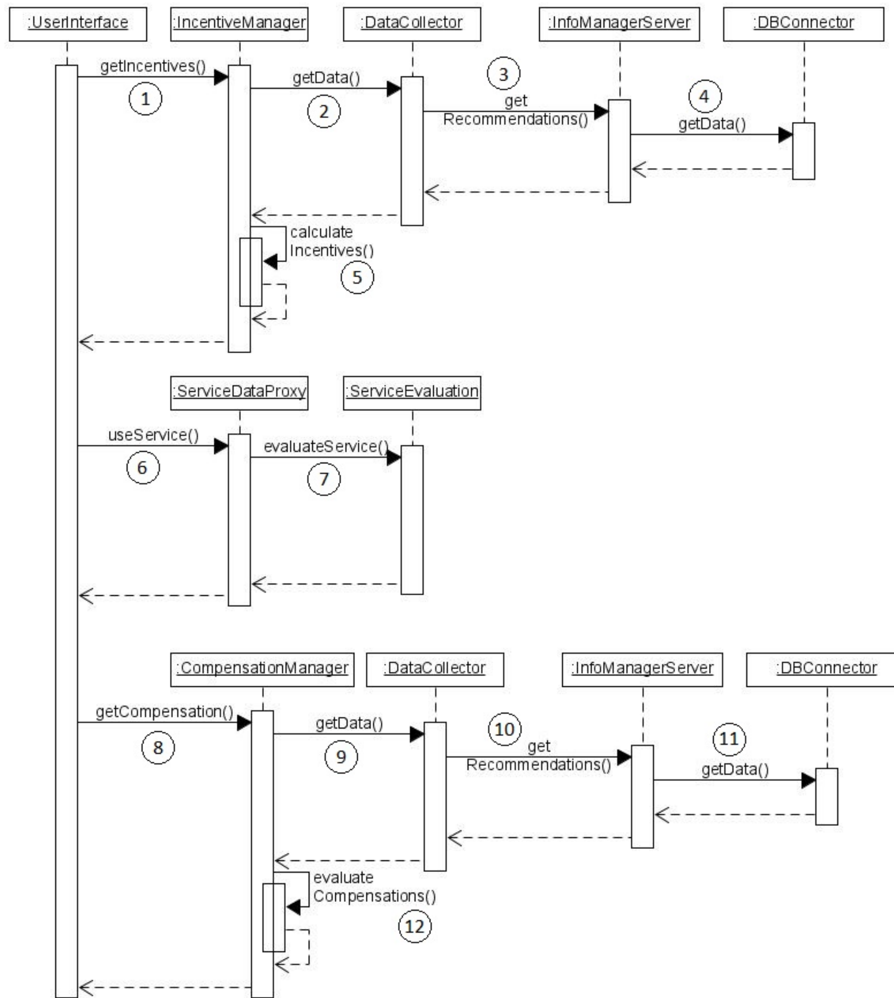

Figure 5.3: Centralized Architecture: Sequence diagram for incentives, utilization and compensations part.

Incentives, Utilization and Compensations

In this sequence diagram, we inspect the acquisition of available incentives, as an addition to the already provided ranking, the utilization of a chosen service through its corresponding proxy and, finally, the provision of compensations, if the outcome of the interaction was inferior to the one expected under normal circumstances. As far as the process described in Section 3.5.1, the equivalent Steps are 7-11 and the diagram can be seen in Fig. 5.3.

The following steps are taken to perform these aforementioned actions:

1. The requesting client's *UserInterface* requests available incentives from the *IncentiveManager*.

2. The *IncentiveManager* utilizes the *DataCollector* to acquire any data pertaining to the evaluation of the appropriate model (e.g. previous service performance, current *T/D* values etc.).

3. Information are requested from the *InfoManagerServer* regarding specific recommendations.

4. The *InfoManagerServer* queries the accompanying *DBConnector* component, after creating the appropriate queries.

5. After the required information has been obtained, the model is evaluated and the provided incentives are calculated and presented to the *UserInterface* (see Section 4.7.4).

6. The service client chooses a service and uses it through the *ServiceDataProxy* component.

7. After the interaction is concluded, the *ServiceDataProxy* sends all captured data to the *ServiceEvaluation* component and requests an evaluation of the service(see *OT/OD values, Section 4.1*). This evaluation can be a result of goal model reasoning, or any other method chosen by the user.

8. The *UserInterface* requests the *CompensationManager* for any applicable compensations, if the service QoS was subpar.

9. The *CompensationManager* utilizes the *DataCollector* to acquire any data pertaining to the evaluation of the appropriate model.

10. Information is requested from the *InfoManagerServer* regarding specific recommendations.

11. The *InfoManagerServer* queries the accompanying *DBConnector* component, after creating the appropriate queries.

12. As soon as all the information is received, the *CompensationManager* calculates the offered compensation (see Section 4.7.5), if any, and makes them known to the *UserInterface*, which then presents to the user the option of accepting them or not.

T/D and R values Update

As soon as the service client decides on the observed trust and distrust obtained through the interaction with a service, another part of the process is initiated. This part has to do with updating the values of anyone who participated in the previous processes. That includes the recommenders who proposed the utilized service, as well as the service itself. The steps corresponding to the process described in Section 3.5.1 are Steps 12-13 and the diagram is depicted in Fig. 5.4.

The following steps are required for the update of all relevant values:

1. The *UserInterface* of the service client makes a request to the *EvaluationManagerFacade* component to update all relevant values.

2. The *EvaluationManagerFacade* forwards this request to the *AlgorithmEvaluators* component.

3. The *AlgorithmEvaluators* asks the *DataBroker* to get the previous value the service client had for the used service, if available.

4. Information is requested from the *InfoManagerServer* regarding said pair of service and service client.

5. The *InfoManagerServer* queries the accompanying *DBConnector* component, after creating the appropriate query.

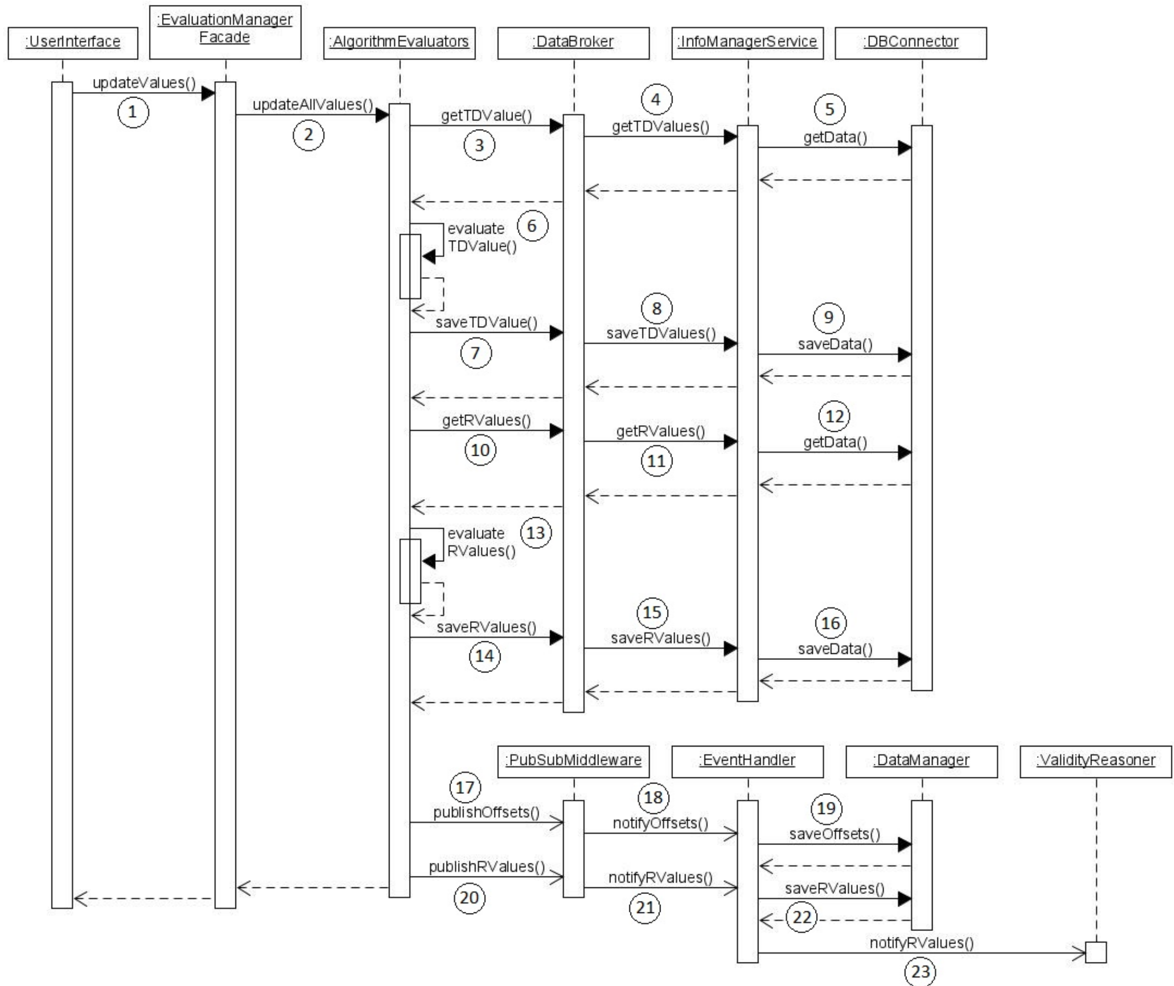

Figure 5.4: Centralized Architecture: Sequence diagram for updating *T/D* and *R* values after service utilization.

6. The provided information is taken into consideration and the appropriate algorithm is run to evaluate the new *T/D* values (see Section 4.2).

7. The *AlgorithmEvaluators* asks the *DataBroker* to forward the newly updated value for saving.

8. The *DataBroker* instructs the *InfoManagerServer* to save the value.

9. The *InfoManagerServer* saves the value to the database using the *DBConnector* component.

10. The *AlgorithmEvaluators* asks the *DataBroker* to get the previous reputation values the service client had for the recommenders, depending on the source they belong to (*R* value for *Friends*, *AR* value for *Experts* and path value for *Friends of Friends*).
11. Information is requested from the *InfoManagerServer* regarding said recommenders.
12. The *InfoManagerServer* queries the accompanying *DBConnector* component, after creating the appropriate queries.
13. The provided information is taken into consideration and the appropriate algorithm is run to evaluate the new *R* values (see Section 4.3).
14. The *AlgorithmEvaluators* asks the *DataBroker* to forward the newly updated or created values for saving.
15. The *DataBroker* instructs the *InfoManagerServer* to save the values.
16. The *InfoManagerServer* saves the values to the database using the *DBConnector* component.
17. The offsets calculated for the reputation values of the recommenders are published to the *PubSubMiddleware* component.
18. The corresponding *EventHandler* is notified about the new offsets, receives them and timestamps them.
19. The *DataManager* is asked by the *EventHandler* to save said offsets to a database.
20. The updated *R* values are, also, published to the *PubSubMiddleware* component.
21. The corresponding *EventHandler* is notified about the *R* values.
22. The *DataManager* is asked by the *EventHandler* to save said *R* values to a database.
23. The *EventHandler* notifies the *ValidityReasoner* to start the other part of the process related to selecting *R* values for calculating overall reputation values (see below).

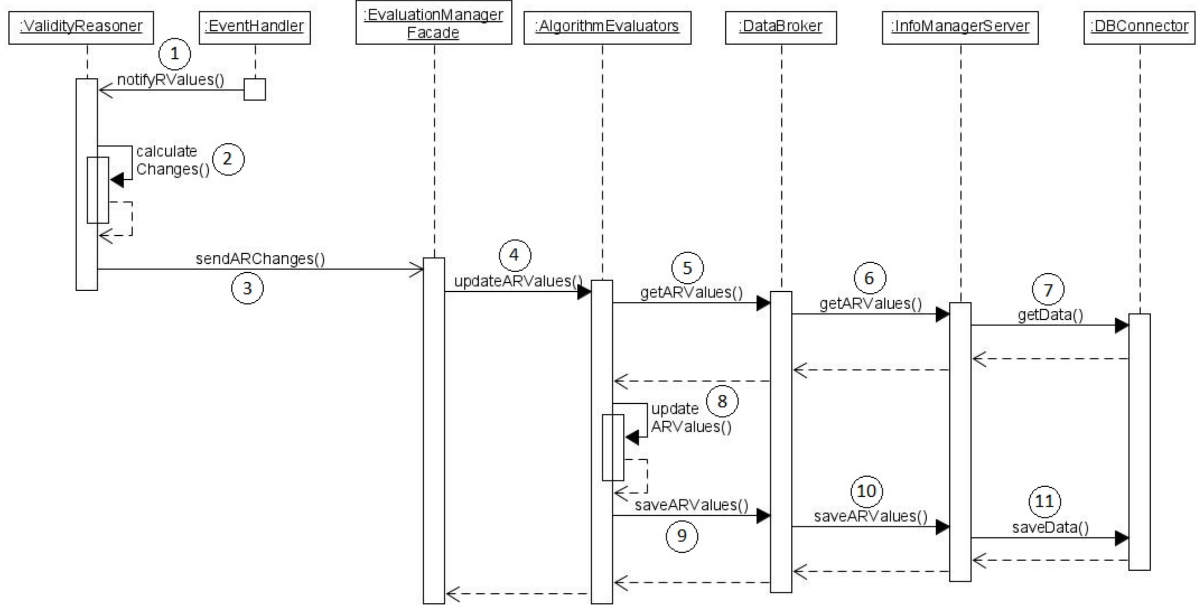

Figure 5.5: Centralized Architecture: Sequence diagram for deciding on important *R* values and updating *AR* values.

Overall Reputation Value Update

Once a reputation value is created or updated, the final part of the process described in Section 3.5.1 is started (i.e. Step 14). The algorithm tasked with selecting the most important *R* values to be considered for an *AR* value is run and, if required, the algorithm to update said value is, also, executed. That part can be seen in Fig. 5.5.

The steps are as follows:

1. The *ValidityReasoner* component is notified about changes in *R* values by the *EventHandler*.

2. The algorithm for deciding which values are important and what needs to to be replaced in the calculation of overall reputation values is run (see Section 4.6).

3. A request is sent to the *EvaluationManagerFacade* to update a set of *AR* values, indicating the *R* values that need to be added and/or removed.

4. The *EvaluationManagerFacade* forwards that request to the *AlgorithmEvaluators* component.

5. The *AlgorithmEvaluators* asks the *DataBroker* to fetch the previous *AR* values.

6. The *DataBroker* asks the *InfoManagerServer* for those information.

7. The *InfoManagerServer* queries the accompanying *DBConnector* component, after creating the appropriate queries.

8. Upon receiving all required information, the *AlgorithmEvaluators* component run the appropriate algorithm and calculates the new *AR* value (see Section 4.5).

9. The *AlgorithmEvaluators* component asks the *DataBroker* to forward the newly updated or created values for saving.

10. The *DataBroker* instructs the *InfoManagerServer* to save the values.

11. The *InfoManagerServer* saves the values to the database using the *DBConnector* component.

Timeouts

The process of removing the *T/D* and *R* values that are considered old and obsolete is completely separate from the main process described in Section 3.5.1. This side process is time-based and is executed in a completely different thread. Its functionality includes discovering older interaction and caused changes in values and reverting them. Note that this timeout process usually, also, causes the triggering of the part of the main process, described in the previous Section (i.e. selecting important *R* values and updating *AR* values). The sequence of actions required to deal with timeouts is available in Fig. 5.6.

1. The *ValidityReasoner* component requests saved offsets for *T/D* and *R* values from the *DataManager*.

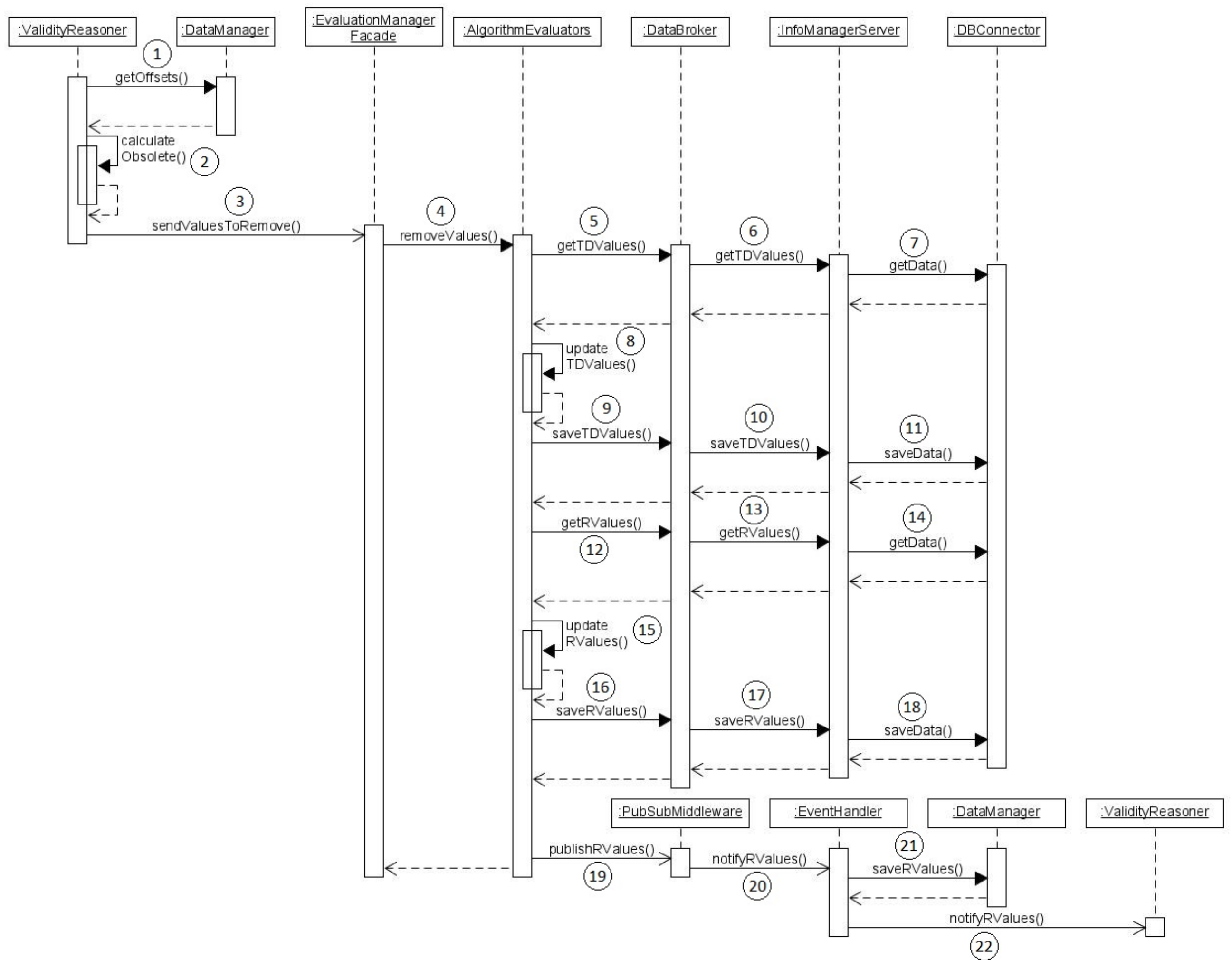

Figure 5.6: Centralized Architecture: Sequence diagram for removing obsolete *T/D* and *R* values.

2. Based on the set time window, entries that have to be deleted are calculated by the *ValidityReasoner*.

3. A request is made to the *EvaluationManagerFacade* to remove those values from consideration.

4. The *EvaluationManagerFacade* forwards this request to the *AlgorithmEvaluators* component.

5. The *AlgorithmEvaluators* asks the *DataBroker* to get the previous values for the pairs of service clients and services that need to be updated.

6. Information is requested from the *InfoManagerServer* regarding said pairs of services and service clients.

7. The *InfoManagerServer* queries the accompanying *DBConnector* component, after creating the appropriate query.

8. The provided information is taken into consideration and the appropriate algorithm is run to update the *T/D* values (see Section 4.4).

9. The *AlgorithmEvaluators* asks the *DataBroker* to forward the newly updated values for saving.

10. The *DataBroker* instructs the *InfoManagerServer* to save the values.

11. The *InfoManagerServer* saves the value to the database using the *DBConnector* component.

12. The *AlgorithmEvaluators* asks the *DataBroker* to get the previous reputation values for the pairs of service clients and recommenders that have to be updated.

13. Information is requested from the *InfoManagerServer* regarding said recommenders.

14. The *InfoManagerServer* queries the accompanying *DBConnector* component, after creating the appropriate query.

15. The provided information is taken into consideration and the appropriate algorithm is run to evaluate the new *R* values (see Section 4.4).

16. The *AlgorithmEvaluators* asks the *DataBroker* to forward the newly updated or created values for saving.

17. The *DataBroker* instructs the *InfoManagerServer* to save the values.

18. The *InfoManagerServer* saves the values to the database using the *DBConnector* component.

19. The updated *R* values are published to the *PubSubMiddleware* component.

20. The corresponding *EventHandler* is notified about the *R* values.

21. The *DataManager* is asked by the *EventHandler* to save said *R* values to a database.

22. The *EventHandler* notifies the *ValidityReasoner* to start the other part of the process related to selecting *R* values for calculating overall reputation values (see previous Section).

5.2 Distributed Architecture

The framework can, also, be deployed in a distributed manner. In scenarios where we need the service clients to act as separate entities and maintain control of their data and operations, this architecture variation can be utilized. Note that this variation is not fully decentralized, as a discovery service is required for new users to be able to discover other clients' network addresses and request recommendations or other information.

5.2.1 Architecture Overview

The distributed version of the framework's architecture is pretty similar to the centralized version. Only a certain amount of components, interfaces and connections are different. The component diagram for the distributed version can be observed in Fig. 5.7. Note that in this version, the whole framework is run by each service client and the users communicate with each other to request and provide recommendations both for services and other service clients, i.e. recommenders.

The components that are different to the centralized version are the following:

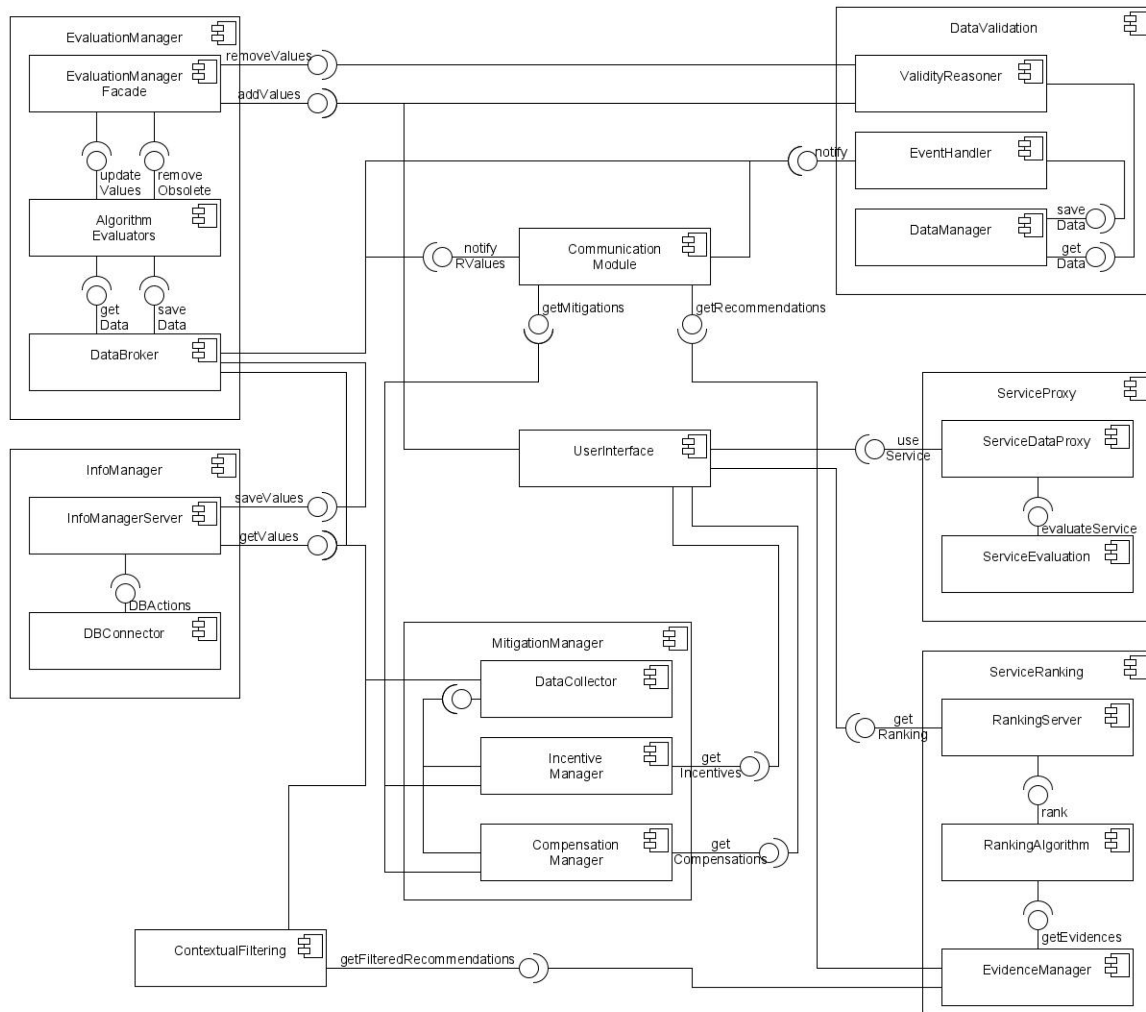

Figure 5.7: Component diagram for distributed architecture.

- **InfoManager:** This component is tasked with maintaining the information regarding only the connection between the service client who is running this instance of the framework and other recommenders and service providers (see Sections 3.3.2, 3.3.3, 3.3.4). No values assigned by other service clients are saved here.

- **CommunicationModule:** A separate component is utilized to deal with the communication with other users of the framework. Recommendations for service providers or other recommenders are requested through this module, as well as information regarding incentives and compensations. The framework is, also, notified for any changes, in *R*

values that are relevant to the calculation of *AR* values, by this component.

Note that a lot of the connections between the components have changed. This is due to the fact that, each service client only has direct access to the values they have assigned to other recommenders and service providers. For all other information required for recommendations, ranking of services, incentives, compensations and updates of overall reputation values, the framework instance has to communicate with the corresponding instances of other clients through the *CommunicationModule*. The *PubSubMiddleware* component has been removed, since each user run one instance of each component and there is no need for decoupling between the *EvaluationManager* component and the *DataValidation* component.

5.2.2 Interface Specification

In this Section, we specify the interfaces offered by all components participating in the distributed version of the framework. Most of them, however, are identical to the ones in the centralized variation, so we will only describe the ones that are different.

Component Name	Interface Name	Description
~~*PubSubMiddleware*~~	~~*publish*~~	~~This interface is utilized to publish offsets and *R* values.~~
	~~*subscribe*~~	~~This interface is utilized to subscribe to specific offsets and information about updates in *R* values.~~
CommunicationModule	*notifyRValues*	This interface is utilized to notify the running instance about changes in *R* values.
	getMitigations	This interface is utilized to acquire information about incentives and compensations, including changes in provided models.
	getRecommendations	This interface is utilized to inquire other service clients for recommendations regarding services or other recommenders.

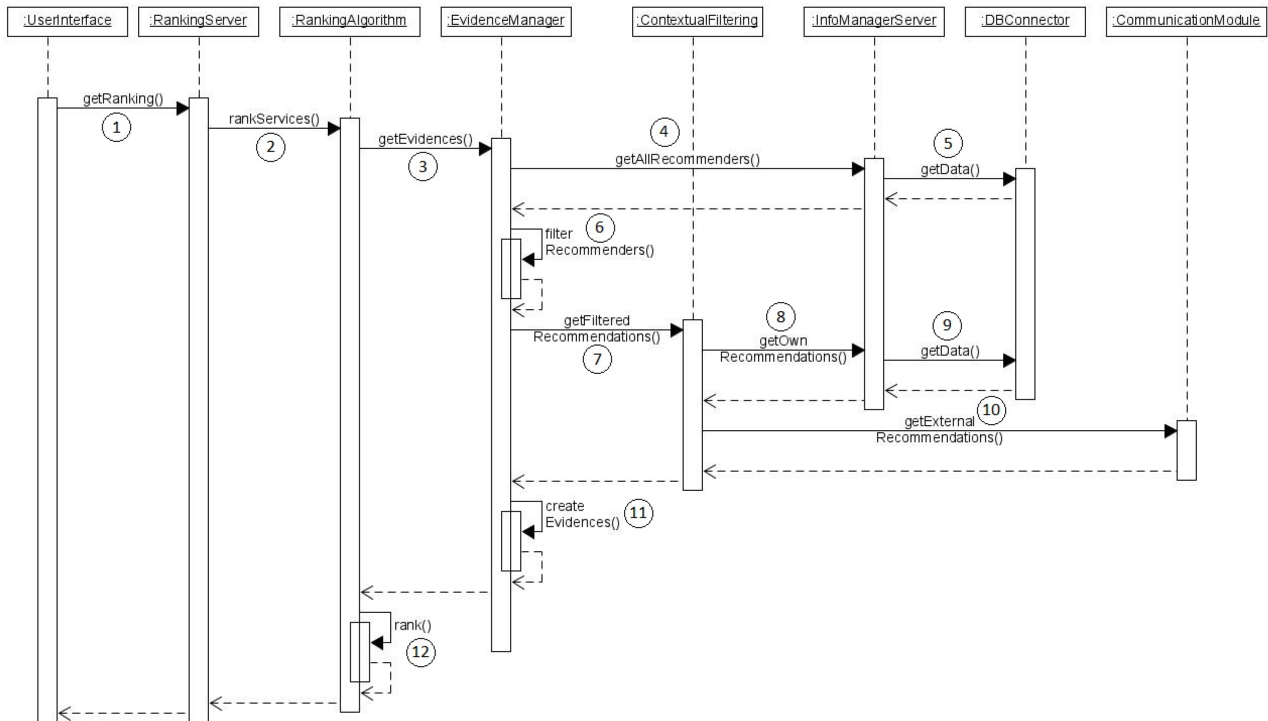

Figure 5.8: Distributed Architecture: Sequence diagram for ranking part.

5.2.3 Process Sequence Diagrams

Once more, to further explain the process required to acquire recommendations and use a service as a service client, as well as the independent processes run to discover obsolete values and decide on important values for the calculation of the overall reputation values, we provide a set of sequence diagrams. The steps are adapted to accommodate a distributed deployment of the framework. Most of the steps, however, are exactly the same as the ones identified in the centralized variation. So, for each of the diagrams, we, still, specify the steps of the process, described in Section 3.5.1, it corresponds to, but only describe the steps in the sequence diagram that are distinct to the ones described in the equivalent section for the centralized variation.

Recommendations and Ranking

The sequence described in this section corresponds to the request for recommendations by a service client, followed by the selection of recommenders to consult, acquisition of recommendations and, finally, transformation to evidences and calculation of service ranking. Said sequence accounts for Steps 1-6 of the process described in Section 3.5.1 and can be observed in Fig. 5.8.

The steps that are different in the distributed variation are:

10. An extra step is added after Step 9. In this variation the service client only maintains their own recommendations. For recommendations from other recommenders, the *CommunicationModule* is utilized to request them.

Incentives, Utilization and Compensations

In this sequence diagram, we inspect the acquisition of available incentives, as an addition to the already provided ranking, the utilization of a chosen service through its corresponding proxy and, finally, the provision of compensations, if the outcome of the interaction was inferior to the one expected under normal circumstances. As far as the process described in Section 3.5.1, the equivalent Steps are 7-11 and the diagram can be seen in Fig. 5.9.

The following steps are additional to the sequence in the centralized variation:

2. After the original request for incentives, the *IncentiveManager* component asks for updated versions of the incentive models pertaining to the available services.

3. If any incentive model requires information from other service clients, the are requested through the *CommunicationModule*.

11. After the request for compensations regarding the utilized service, the *CompensationManager* inquires for any update in the compensation model pertaining to said service.

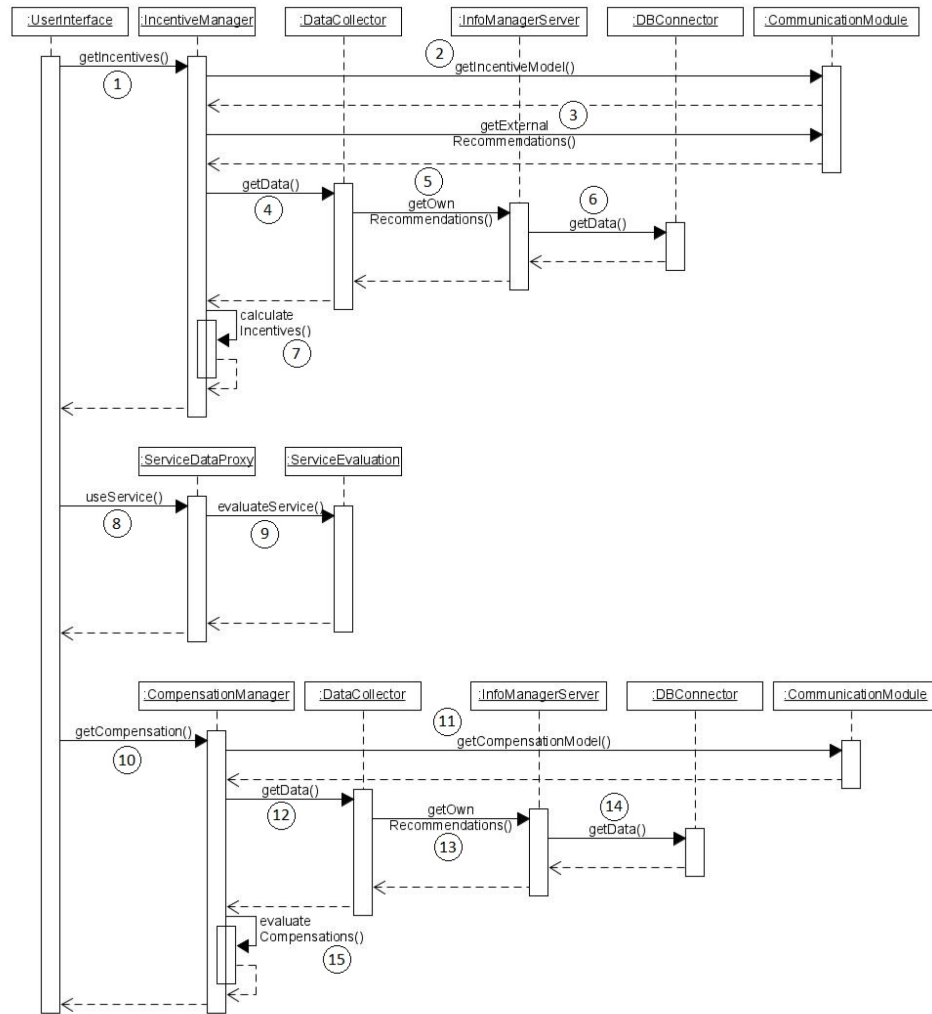

Figure 5.9: Distributed Architecture: Sequence diagram for incentives, utilization and compensations part.

T/D and R values Update

As soon as the service client decides on the observed trust and distrust obtained through the interaction with a service, another part of the process is initiated. This part has to do with updating the values of anyone who participated in the previous processes. That includes the recommenders who proposed the utilized service, as well as the service itself. The steps corresponding to the process described in Section 3.5.1 are Steps 12-13 and the diagram is depicted in Fig. 5.10.

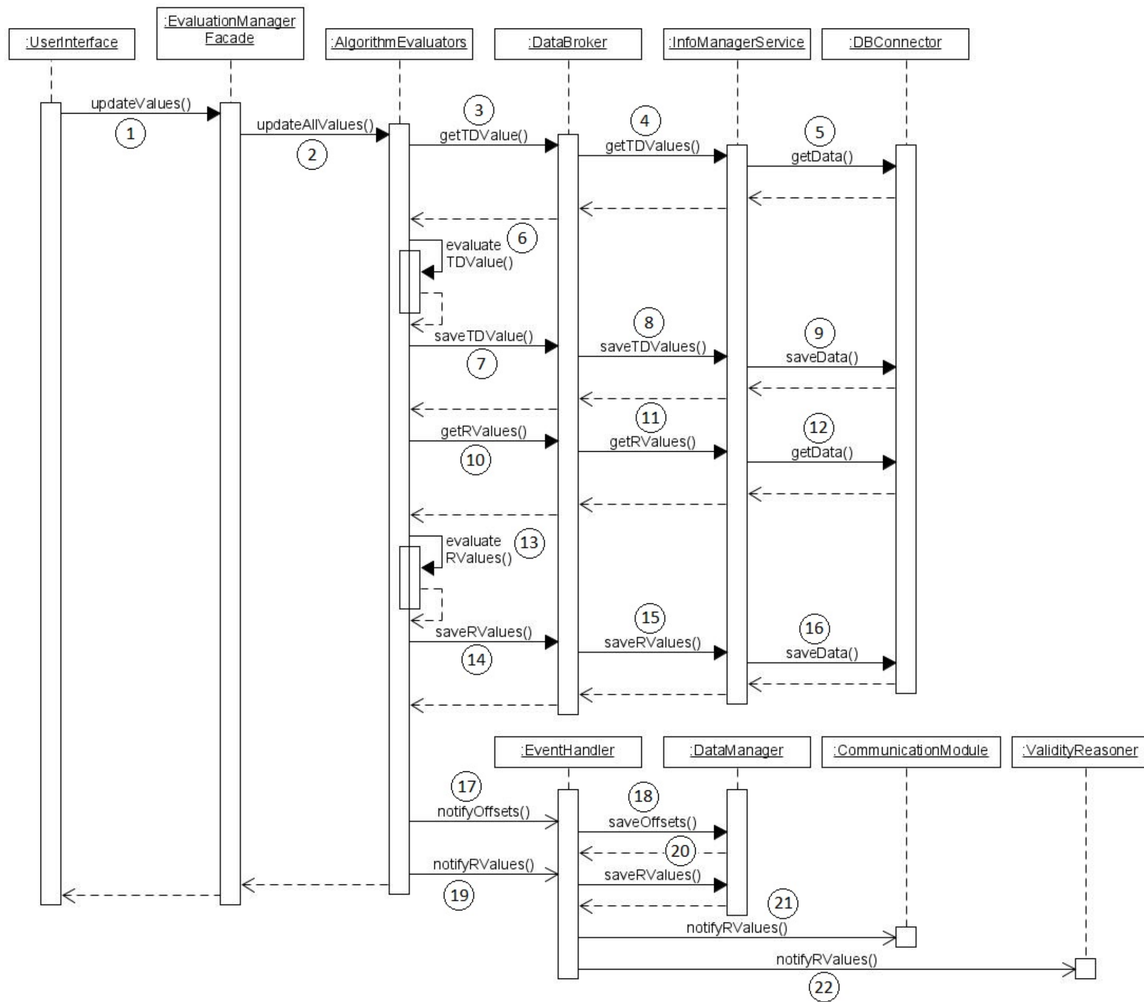

Figure 5.10: Distributed Architecture: Sequence diagram for updating *T/D* and *R* values after service utilization.

The following steps are different to the ones specified for the centralized architecture:

17. The *EventHandler* component is directly notified about the new offsets, receives them and timestamps them. No *PubSubMiddleware* is required.

18. The *DataManager* is asked by the *EventHandler* to save said offsets to a database.

19. The *EventHandler* is directly notified about the *R* values. Again, no *PubSubMiddleware* is required.

20. The *DataManager* is asked by the *EventHandler* to save said *R* values to a database.

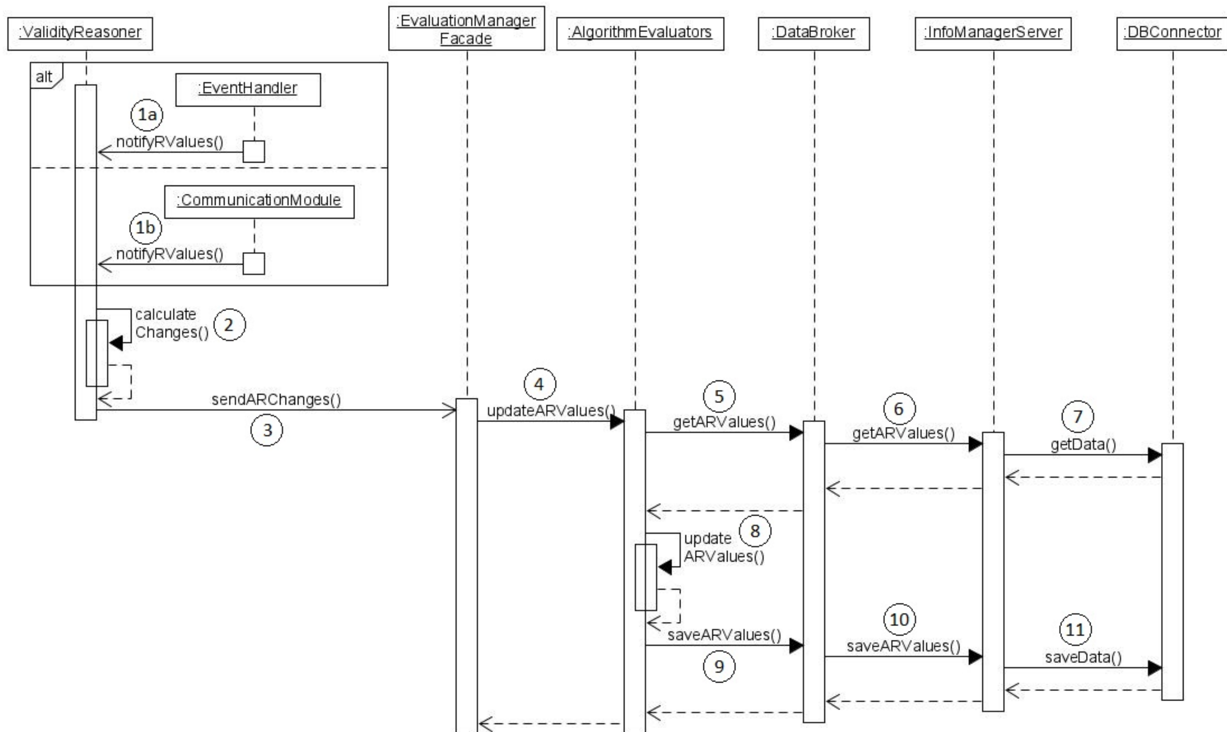

Figure 5.11: Distributed Architecture: Sequence diagram for deciding on important *R* values and updating *AR* values.

21. The *CommunicationModule* is asked to notify other service clients about the changes in the *R* values.

22. The *EventHandler* notifies the *ValidityReasoner* to start the other part of the process related to selecting *R* values for calculating overall reputation values (see below).

Overall Reputation Value Update

Once a reputation value is created or updated, the final part of the process described in Section 3.5.1 is started (i.e. Step 14). The algorithm tasked with selecting the most important *R* values to be considered for an *AR* value is run and, if required, the algorithm to update said value is, also, executed. That part can be seen in Fig. 5.11.

The steps that have been altered for the distributed variation are as follows:

1. The *ValidityReasoner* component is notified about changes in *R* values by another component of the framework. In the distributed version, there are two possible components that can do that.

   (a) The *EventHandler* component notifies the *ValidityReasoner* about updated *R* values that have occurred from within that system instance.

   (b) The *CommunicationModule* component notifies the *ValidityReasoner* about updated *R* values that were calculated by other service clients, external to the system.

Timeouts

The process of removing the *T/D* and *R* values that are considered old and obsolete is completely separate from the main process described in Section 3.5.1. This side process is time-based and is executed in a completely different thread. Its functionality includes discovering older interaction and caused changes in values and reverting them. Note that this timeout process usually, also, causes the triggering of the part of the main process, described in the previous Section (i.e. selecting important *R* values and updating *AR* values). The sequence of actions required to deal with timeouts is available in Fig. 5.12.

Some of the final steps are different and they can be seen below:

19. The *EventHandler* is directly notified about the *R* values. Again, no *PubSubMiddleware* is required.

20. The *DataManager* is asked by the *EventHandler* to save said *R* values to a database.

21. The *CommunicationModule* is asked to notify other service clients about the changes in the *R* values.

22. The *EventHandler* notifies the *ValidityReasoner* to start the other part of the process related to selecting *R* values for calculating overall reputation values (see above).

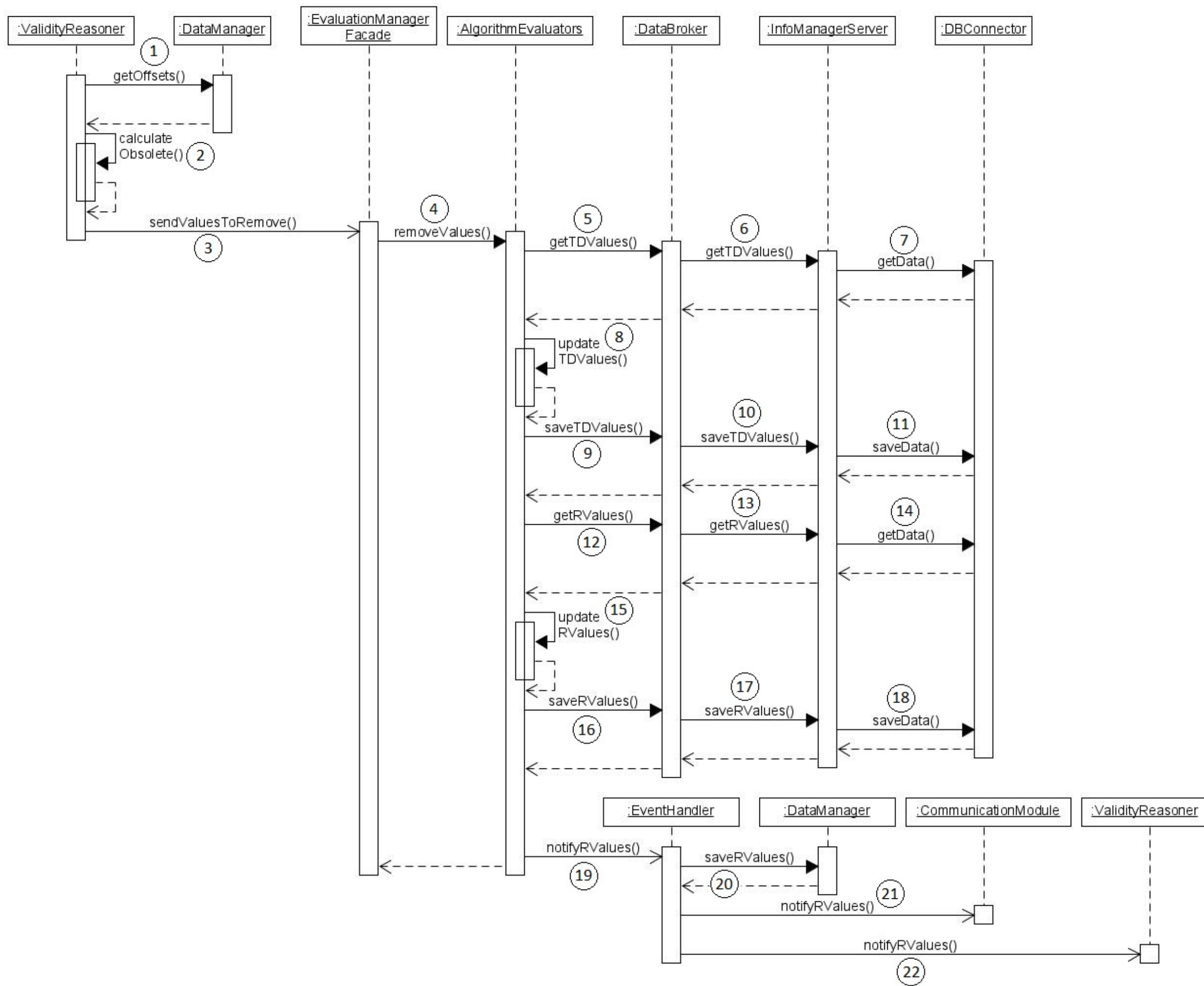

Figure 5.12: Distributed Architecture: Sequence diagram for removing obsolete *T/D* and *R* values.

5.3 Blockchain Architecture

If one wants to combine the simplicity of the centralized approach with the decentralization and fault tolerance of the distributed variation, the framework can, also, be deployed in any blockchain platform that supports smart contracts written in an object-oriented language.

Since every participant in the blockchain runs a copy of each available smart contract, the centralizes architecture can be utilized. Each component and/or class pertaining to that variation can be deployed as a separate smart contract, with each of them interacting with the

rest as described through the interfaces and sequence of actions described in Section 5.1.2 and Section 5.1.3 respectively. The main difference lies in the fact that no REST APIs are required, as smart contracts can refer to and use one another, utilizing just the blockchain address that corresponds to them.

Furthermore, each service client and service provider can be identified by their wallet address, provided by the blockchain that is hosting the framework. There is, also, no need for separate instances of the *UserInterface* component, just an extra parameter in the methods, corresponding to each operation, to specify the user who acts.

5.4 Messaging Protocols

Regardless of whether the framework is deployed using the centralized or the distributed approach, a messaging protocol has to be established for the main interactions between component. Those protocols are especially important in the centralized version, since all communication is performed in a RESTful way and handled by servers, but are, also, helpful in the distributed variation of the architecture, as they allow for changes without altering method signatures or creating unnecessary adapters

A specific template is utilized for each interaction requiring data. The templates are based on JSON representation.

5.4.1 Data Acquisition

In the part of the process that requires communicating with the *InfoManager* component to retrieve or save data in the accompanying database, one must follow the following templates to request the data and will receive the response based on the corresponding template. In case of saving data, the latter template is, also, used to send the information.

```
1 Requesting message:
2 ["R1","R448","R207",...]
3
4 Response to request/Message to save:
```

```
5  {
6          "R1":{
7                          "aggrRecommendation":44.435797,
8                          "numOfRecs":55
9          },
10         "R448":{
11                         "aggrRecommendation":39.999091,
12                         "numOfRecs":49
13         },
14         "R207":{
15                         "aggrRecommendation":33.918042,
16                         "numOfRecs":41
17         },
18         ...
19 }
```

Listing 5.1: Requesting and saving *AR* values

```
1  Requesting message:
2  {
3          "from":["R757" ,...],
4          "to":["R206","R723","R400" ,...]
5  }
6
7  Response to request/Message to save:
8  {
9          "R757":{
10                 "R206":{
11                         "recommendation":0.981126,
12                         "count":1
13                 },
14                 "R723":{
15                         "recommendation":0.800908,
16                         "count":1
17                 },
18                 "R400":{
19                         "recommendation":0.731771,
20                         "count":1
21                 },
22                 ...
23         }
24 }
```

Listing 5.2: Requesting and saving *R* values

```
1  Requesting message:
2  {
3          "from":["R966","R227" ,...],
4          "to":["SP91","SP2","SP3","SP81","SP60" ,...]
5  }
6
7  Response to request/Message to save:
8  {
9          "R966":{
```

```
            "SP91":{
                "trust":0.967272,
                "distrust":0.003114,
                "count":1
            },
            "SP2":{
                "trust":0.910291,
                "distrust":0.084155,
                "count":1
            },
            "SP3":{
                "trust":0.989693,
                "distrust":0.002089,
                "count":2
            },
                ...
        },
        "R227":{
            "SP81":{
                "trust":0.857505,
                "distrust":0.024561,
                "count":1
            },
            "SP60":{
                "trust":0.89202,
                "distrust":0.071898,
                "count":1
            },
                ...
        },
        ...
}
```

Listing 5.3: Requesting and saving *T/D* values

5.4.2 Update values after interaction

After an interaction with a service, certain values need to be updated. The template for the message requesting said updates is as follows.

```
Requesting message:
{
        "tdEntry":{
            "userID":"R253",
            "trust":0.9779683718148557,
            "distrust":0.002093833120475899,
            "serviceProviderID":"SP3"
        },
        "rEntry":{
            "recThroughR":{
                "R573":{
```

```
12                     "trust":1.0,
13                     "distrust":0.002039,
14                     "count":3
15                 },
16                 "R807":{
17                     "trust":0.989377,
18                     "distrust":0.002096,
19                     "count":5
20                 }
21             },
22             "recThroughAR":{
23                 "R370":{
24                     "trust":0.998778,
25                     "distrust":0.002009,
26                     "count":2},
27                 "R60":{
28                     "trust":0.994302,
29                     "distrust":0.002035,
30                     "count":1}
31             },
32             "recThrough2stepR":{
33                 "R757":{
34                     "R206":{
35                         "trust":0.998778,
36                         "distrust":0.002009,
37                         "count":3
38                     },
39                     "R723":{
40                         "trust":1.0,
41                         "distrust":0.002039,
42                         "count":6
43                     },
44                     ...
45                 }
46             }
47         }
48 }
```

Listing 5.4: Updating *R* and *T/D* values

5.4.3 Updating AR values

If a new *R* value is added, or another one is removed, for consideration when calculating an overall reputation value, the request for those actions is done utilizing the following template.

```
1 Requesting message:
2 {
3     "toRemove":{
4         "R868":{
5             "R473":{
6                 "recommenderARValue":0.812729,
```

```
7                   "recommendation":1.059357
8               },
9               "R142":{
10                  "recommenderARValue":0.813596,
11                  "recommendation":1.047187
12              },
13              ...
14          }
15      },
16      "toAdd":{
17          "R943":{
18              "R368":{
19                  "recommenderARValue":0.8212,
20                  "recommendation":1.0495
21              }
22          },
23          "R526":{
24              "R584":{
25                  "recommenderARValue":0.811176,
26                  "recommendation":1.0737496
27              },
28              ...
29          },
30          ...
31      }
32  }
```

Listing 5.5: Updating *AR* values

5.4.4 Publications of Events

When an offset is applied to a *R* or a *T/D* value, the *EventHandler* component has to be notified, either through the *PubSubMiddleware* component or directly. The templates for those publications are shown here.

```
1   Publication of R value:
2   {
3       "recommender":"R53",
4       "recommenderAR":0.778833761904762,
5       "recommendee":"R496",
6       "recommendation":0.8823443126200042,
7       "offset":0.0727942792866708,
8       "count":1,
9       "resets":true,
10      "timestamp":"May 27, 2022, 5:35:02 PM"
11  }
```

Listing 5.6: Publishing *R* values

```
1  Publication of T/D value:
2  {
3          "userID":"R647",
4          "trust":0.9987913185460922,
5          "distrust":0.0020746573342638936,
6          "serviceProviderID":"SP3",
7          "timestamp":"May 27, 2022, 5:35:56 PM"
8  }
```

Listing 5.7: Publishing *T/D* values

5.4.5 Removing obsolete values

Whenever an offset, applied to either a *T/D* or a *R* values, is deemed obsolete, a request is made for it to be removed from consideration. The request template is as follows.

```
1  Request message:
2  {
3          "tdEntries":[
4                  {
5                          "userID":"R301",
6                          "trust":0.974798,
7                          "distrust":0.00173,
8                          "serviceProviderID":"SP66",
9                          "timestamp":"May 27, 2022, 5:09:37 PM"
10                 },
11                 {
12                         "userID":"R987",
13                         "trust":0.975056,
14                         "distrust":3.83E-4,
15                         "serviceProviderID":"SP95",
16                         "timestamp":"May 27, 2022, 5:09:37 PM"
17                 },
18                 ...
19         ],
20         "rEntries":[
21                 {
22                         "recommender":"R455",
23                         "trustOffset":0.070709,
24                         "recommendee":"R532"
25                 },
26                 {
27                         "recommender":"R367",
28                         "trustOffset":0.06951,
29                         "recommendee":"R284"
30                 },
31                 ...
32         ]
33 }
```

Listing 5.8: Publishing *T/D* values

5.5 Summary

Summarizing, the proposed framework can be deployed in a number of different settings, depending on the requirements of the domain and the accompanying technologies or needs.

A centralized architecture is proposed in Section 5.1. All of the required components are described in Section 5.1.1, as well as their purpose and connections to the rest of the system. The interfaces involved in those connections are defined in Section 5.1.2 and the sequences of actions required to perform the main process of selecting recommenders, acquiring recommendations and using a service, as well as the parallel ones involving timeouts and calculation of overall reputation values, are specified and explained in Section 5.1.3.

In cases where the system needs to scale up to accommodate larger amount of users, each component specified as part of the proposed architecture can be separately deployed as a microservice. Functionality of each component is distinct and container technology can be used to run them in an isolated manner. Solutions pertaining to the management of containerized applications, such as *Kubernetes* [148], can perform replication and load balancing between multiple instances of the same component, thus allowing for practically unlimited number of service clients and providers participating in the framework. As far as data storing in concerned, distributed databases, handled by replicas of the *InfoManager* component, can accommodate any need for scalability of the proposed approach.

To further improve the performance and extensibility of the framework, one could perform *contextual filtering* on the ranking process. Based on the type of service required by the requesting service client, logic can be incorporated to the *ContextualFiltering* component that will allow the recommendations and resulting ranking of available services to be filter as required depending on the provided context or type of service required in that particular inter-

action. Of course, in real-world scenarios separate social networks and corresponding values can be maintained for different types of services, even if they are offered by a single service provider and recommended by the same service clients.

Moreover, filtering could be performed on a *geographical* basis. In some cases, utilizing a service may depend on proximity, either due to network restrictions or because physical interaction is required at some point. In those scenarios, different instances of the framework should be deployed and used by different geographical regions. Filtering could, also, be performed if further granularity is required when choosing an offered service.

A variation that allows the approach to be deployed in a distributed manner is presented in Section 5.2. The corresponding architecture (see Section 5.2.1), as well as the differences involved both in the available interfaces (see Section 5.2.2) and in the action sequences (see Section 5.2.3), are, also, explained.

The possibility of utilizing the centralized architecture to deploy the proposed framework in a blockchain environment is explored, indicating the minor alterations that would be required, mostly in the communication between components.

Finally, the protocol and associated templates, required for the making requests to acquire and/or update values within the system, are presented, allowing for better understanding of the framework's functionality.

Chapter 6

Implementation and Experiments

6.1 Overall Infrastructure

Based on our proposed approach, we implemented a prototype following the architecture described in Section 5. The centralized version was the one chosen for the implementation, but each component was created to be deployable as a separate server or microservice and some third-party frameworks were incorporated to provide a scalable version of the implementation.

6.1.1 Framework implementation

A fully functional version of the framework was implemented as part of this approach. The implementation was done using the *Java* programming language and each component described in the previous chapter utilized the builtin HTTP server class to provide the offered API in the form of a REST service. No other frameworks were used for the server parts (e.g. Spring) since the functionality required was pretty straightforward and there was no need for additional communication overhead. Separate threads were, also, utilized for discovering obsolete values (see Section 4.4) and selecting R values for calculating overall reputation values (see Section 4.6), since the process involved for these actions is entirely separate to the main process described in Section 3.5.1.

6.1.2 Event Propagation

Communication between components to accommodate the steps involved in the main process (i.e. getting recommendations and using a service) is pretty linear and straightforward, so direct calls to the REST APIs offered by the different parts of the framework were utilized.

When it came to handling old values to be discarded and deciding on important values to be considered for overall reputation, though, we opted for a more event-based approach. The motivation behind that choice came from the fact that components can be replicated to allow for increased throughput and better performance. Since the scenarios explored in this case are not part of the main process, but rather separate processes triggered based on information availability, and a state is required for each set of values, coupling should be avoided. Keep in mind that different instances of the component, responsible with discarding *T/D* and *R* values, may maintain information for different subsets of available nodes. So, to avoid issues with discovery of appropriate instance to notify and runtime changes in available instance replicas, we incorporated a *Publish/Subscribe* framework. The chosen framework was *MQTT*, offered by the *Eclipse Foundation*, and offsets applied to either *T/D* or *R* values, as well as updated cumulative *R* values, are published to it using appropriately formulated topics. Interested parties, i.e. components tasked with handling old values to be discarded and deciding on important values to be considered for overall reputation, subscribe to said topics. If a replica of the components instance is required, one would only have to unsubscribe and/or resubscribe to the appropriate topics without messing with any other parts of the system.

6.1.3 Database Utilization

Since one of our main objectives is for our approach to be scalable and have high throughput in the presence of large number of users, we had to use a value storing solution that could be utilized in a microservice environment and would allow for replication and load balancing. To accommodate those requirements, a NoSQL database program, called *MongoDB*, was chosen. *MongoDB* can be deployed centrally as a single node, but can, also, be deployed as a microser-

vice pod and be replicated to provide higher throughput and/or data redundancy. The database program is not available to all components, but is utilized by the component dealing with the information management, as described in Chapter 5.

6.2 Experiments

In order to evaluate the performance and behaviour of the proposed system we had to conducted a series of experiments, using the implemented prototype. Service rankings requested by various sources were monitored and juxtaposed with the actual ranking of services based on their predetermined behaviour (i.e. QoS). The experiments focused on four aspects *a)* stability of the system in the presence of malicious recommenders; *b)* how quickly degrading services are recognized; *c)* how connections are formed as the system operates; *d)* how the different types of recommenders (best overall, friends, friends-of-friends) are distributed over time; *e)* how incentives allow for introduction or re-institution of services into the top tier of the rankings and; *f)* what is the transaction success rate when dealing with various percentages of dishonest clients and how our system compares to other approaches.

6.2.1 Simulator

Looking through the related work on reputation systems, no appropriate dataset was discovered. This is totally expected, since reputation systems have been geared to accommodate several different types of networks and a single dataset couldn't possible account for all available scenarios. Moreover, most experiments are conducted to test each approach's resilience in the face of high percentages of malicious users and ability to quickly converge, which is something that cannot be easily captured by a specific dataset.

Due to those reasons, the approaches that do test their resilience, ability to converge and overall performance are utilizing a simulator. The majority implement their own simulation solution based on the approach's specific nature. A few, however, use an already available

simulator. The caveat is that those approaches are explicitly interested in addressing issues of specific types of networks. More specifically, CONFIDANT [13] utilizes a simulator for mobile ad-hoc networks, namely GloMoSim [142], while R2Trust's simulation is based on QueryCycleSimulator [143] that simulates interactions in the context of P2P file sharing networks. Every other proposed framework [3, 8, 10, 11, 12, 19, 138] implements their own simulator to address their specific needs. XRep [6] is, also, an exception, since they modified a *Gnutella* client and then monitored an actual network, but they only reported findings on assumptions made in their approach regarding distribution of files and clients, not algorithm resilience against malicious users.

For our approach, we, also, implemented a simulator to test the behaviour of our framework. The simulator was implemented using the same programming language as the rest of the system, i.e. Java, and acted as a number of external service clients and service providers registering to the framework.

The simulator accepts the number of service clients and service providers as parameters and initializes the appropriate instances for each of them. User Interfaces are created for each service client (see Section 5) and proxies are registered for each requested service.

The behaviour of services is mocked by the simulator and an interface for altering a service's behaviour at runtime is also implemented. When it comes to service clients, the User Interface offered by the main framework had to be enhanced to allow for malicious behaviour, if needed. In practice, an extra method was added that allowed the simulator to adjust the observed values obtained after an interaction. Note, though, that, other than that simple modification, the framework remained intact and the simulator never meddled with its functionality.

Moreover, an extra interface was added to the framework's API to facilitate the simulator in importing a set of already existing relations. This was implemented to help speed up the experiments' execution, since lack of such functionality would require a much lengthier bootstrapping phase (see Section 6.2.2). Again, the approach was not affected at all by this addition, since it only provided a way for importing relations that could have occurred through regular

interactions from within our framework.

The simulator supports a plethora of other parameters, as well. One can set the connectivity of the network that will be created as part of the initialization phase. The user can specify the number of connections a service client will have with other service clients and service providers and the simulator will randomly create them and import them into the framework using the aforementioned interface. Even though the values corresponding to those relations are randomly generated, the user can set the range allowed for each of those values.

When it comes to the actual execution of the experiments, parameters are offered to specify the number of a) user interactions that constitute an *iteration*, b) iterations, if the type of experiment requires a set amount, and c) times the experiment is to be repeated.

Last but not least, the simulator's user can instruct a percentage of service clients to act in a malicious way. Further parameterization of maliciousness is available, where clients can have different levels of maliciousness and can either overvalue or undervalue a specific service or a set of services.

6.2.2 Experimental Setup

Simulated Network Setup

For the experiments presented below, a network of 1,000 recommenders and 100 service providers was simulated using our simulator implementation described in Section 6.2.1.

Each service registered was assigned a pair of random values corresponding to their baseline *trust* and *distrust* values. These values ranged from 0.7 to 1.0 and 0.0 to 0.3, for *trust* and *distrust* respectively. Note that the simulator provides values that are within ±2.5% of the baseline values every time an interaction with a service occurs. Also, bear in mind, that the simulator can later override the baseline values in required by the experiment.

Each recommender was randomly connected to 10 service providers and the initial *T/D* values towards each service were randomized within ±5% of the baseline values selected for that particular service. Each recommender was, also, connected to 10 other recommenders

with a random *R* value that ranged between 0.6 and 0.9.

Experiments Execution

A random subset consisting of 5% of all users would complete the process described in Section 3.5.1 in what we consider to be an *iteration* of the experiment.

After the completion of each *iteration*, information about the ranking of the services was received through three different sources: *a)* an external user who was only requesting the rankings but never participating in the network (i.e. using any service),*b)* a random subset of non-malicious recommenders that was determined at the beginning of the experiment and remained the same for all iterations, and *c)* a random subset of non-malicious recommenders that would change for each iteration.

Experiments would consist of 200 iterations, if a specific amount was required, or until a condition was met. Every set of experiments was run five times using the same parameters, and an average of the results is reported here. This was another safeguard against the threat to validity of our experiments due to the randomized *R* values between recommenders.

Bootstrapping phase

Even though the *T/D* values assigned to services by recommenders were representative of their actual simulated behaviour, the *R* relations between recommenders were entirely random. So, before initiating the actual experiments, a stabilization phase was necessary. During that phase, the system was allowed to run for a few iterations until an equilibrium was achieved. Ensuring that the top services in the rankings of all three aforementioned sources remained the same for at least 5 iterations was the way to achieve the required homeostasis of the network. This initial phase is referred in the related literature as *bootstrapping*.

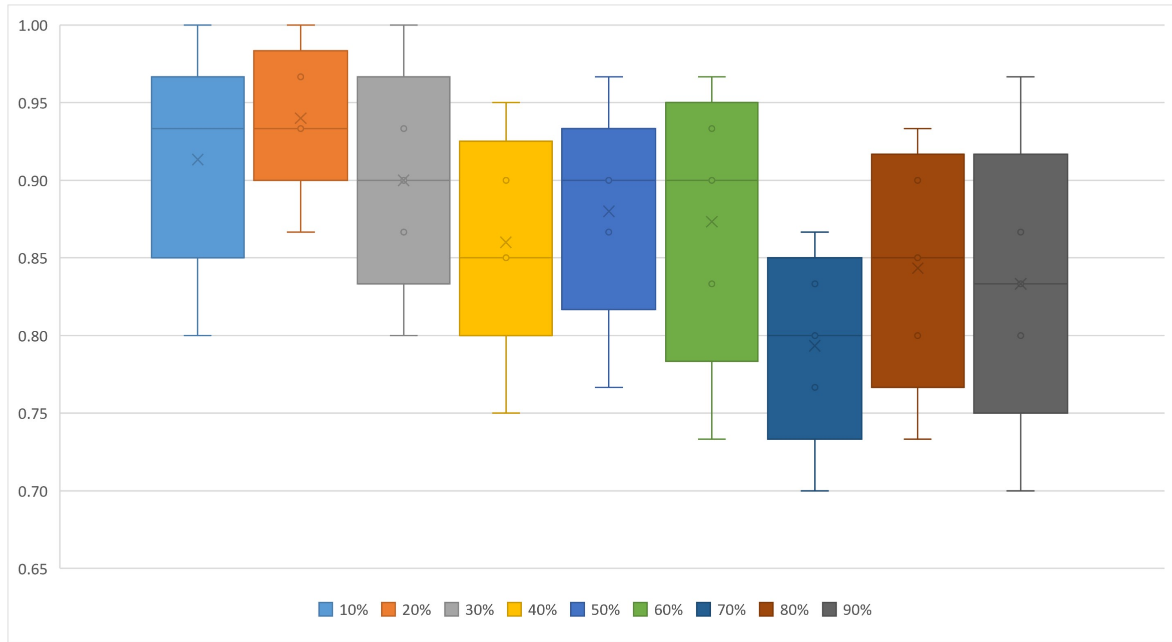

Figure 6.1: System stability vs Percentage of Malicious Components

6.2.3 Stability in the Presence of Malicious Components

To verify the stability of our approach in the presence of malicious users, with respect to contribution *C1* regarding the system's resiliency against dishonest service clients, several experiments were conducted with various percentages of components providing maliciously false recommendations. Each experiment consisted of 200 iterations as described in Section 6.2.2 and users were instructed to turn malicious right after the initial phase (see Section 6.2.2).

Experiments included various percentages of total recommenders being malicious (i.e. knowingly providing false observed *T/D* values after interactions with services) in each corresponding set of experiments. Initial ones started with 10% of service clients acting dishonestly, going all the way up to 90% of malicious clients, in increments of 10%.

For each different percentage of malicious users introduced into the system, the simulator would record the top services, as provided through the baseline values assigned to them during the network setup phase. Said original top was, then, compared to the top services produced by the random subsets of non-malicious clients after each iteration of the experiment. The similarity was calculated and the range of values observed for all iterations for each percentage

of malicious users is presented in Fig. 6.1.

When conducting this set of experiments, our main goal was to discover whether non-malicious users could receive trustworthy recommendations, despite the rising percentages of malicious counterparts. The similarity of the original ranking of top services and the one produced after each iteration indicates the system's ability to filter out bad recommendations, thus maintaining a fairly unchanged ranking for those clients who remain honest.

Of course, we were expecting the similarity to drop for higher percentages of malicious users, but we wanted to make sure that the majority of the services belonging to the ranking would remain the same. Even though there is some variation, the results indicate that the system remains stable even for 90% of users being malicious. The top service providers reported in the rankings, deriving from recommendations from all sources, are 70-95% the same as the ones reported in the ranking compiled using the predetermined and objective behaviour of the available services. For lower percentages of malicious clients (i.e. 10-50%), the similarity ranges between 80-100%, with only some occasional interaction outliers dropping lower than that.

The fact that the proposed framework relies on dynamic assessment of the behaviour of both the recommenders and the service providers allows non-malicious users to try, assess and learn from previous experience as to what sources of information are reliable or not. As expected, even for high percentages of malicious users, *R*connections between honest users are strengthened and said users value and rely on the opinions of those they trust to evaluate the degree to which a service provider actually performs as promised. *Experts* are overwhelmed when malicious users account for the majority of clients but, even though it would be tempting to completely exclude the opinion of groups currently deemed dishonest, we opted to allow the user to receive and accordingly weigh those opinions, so as to enable the adaptation of our framework to changes in the future.

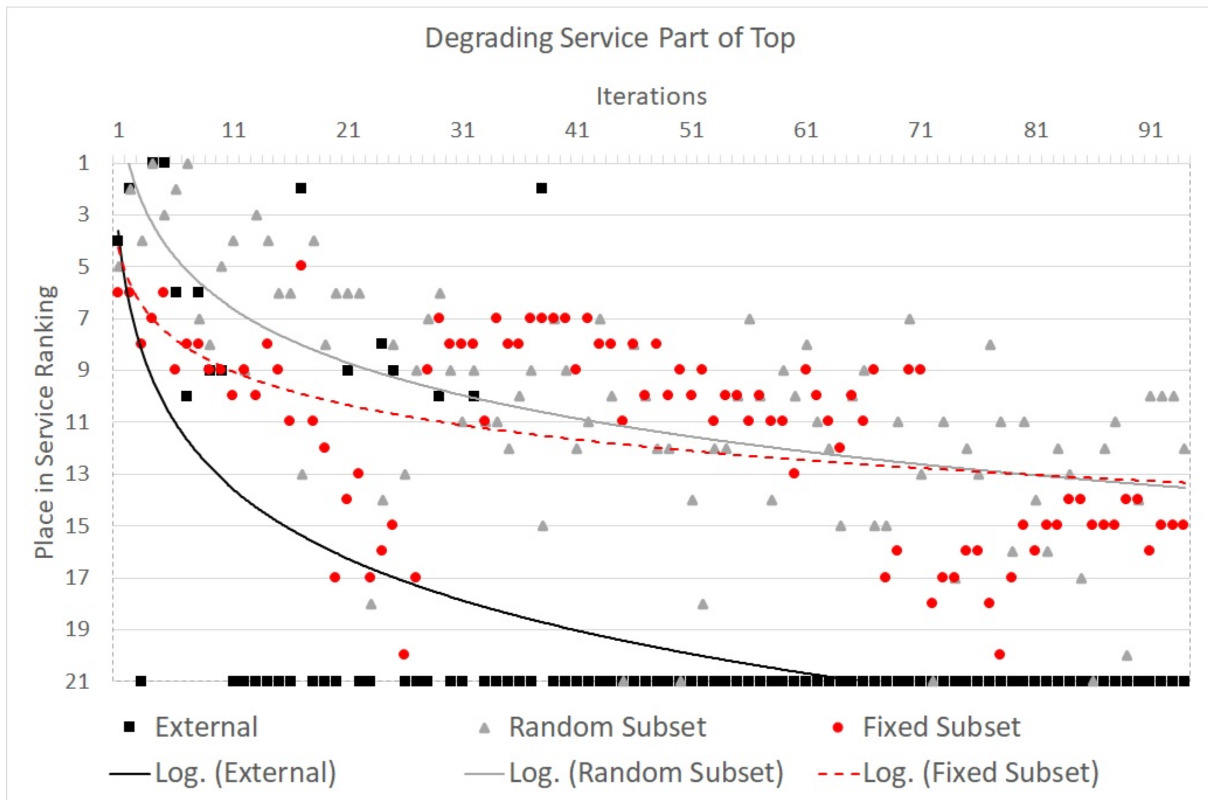

Figure 6.2: Recognition of degrading service

6.2.4 Degrading Services

To evaluate the proposed framework's dynamic behaviour, which is improved through the use of data aging for available values, as is specified in contribution *C6*, another set of experiments was run. Those experiments were conducted to ensure that our approach adapts to changes in the behaviour of a well-established service over time. Since the level of trustworthiness of any service is not guaranteed to remain the same forever, it is crucial that our framework identifies dynamic behaviour and allows for adjustment of any service's reputation within the network. This particular scenario, also, covers the possibility of a malicious service provider who might provide high quality service to lure-in potential customers and improve their reputation before deteriorating the quality of offered service and taking advantage of said customers.

To monitor that type of behaviour, the simulator's capability of adjusting a service's baseline *trust* and *distrust* values was utilized. Similarly to the previous set of experiments, the

initial *bootstrapping* phase was performed (see Section 6.2.2) and the top service, based on the predefined baseline *trust* and *distrust* values, was identified. Experiments were run for as many iterations as required, before all three sources of ranking would report that the degrading service is no longer in the top of the retrieved rankings.

The expectations from this set of experiments were related to the framework's ability to identify changes in behaviour in a timely manner. Discarding old and obsolete values, while at the same time accounting for new observations, should allow the proposed approach to quickly find out whether a service's behaviour has been altered.

Results are shown in Fig. 6.2, where the position of the identified top service after every iteration can be seen, as well as a logarithmic trendline that facilitate the identification of the general trend for each source of rankings.

It is perfectly normal for the system to require a number of iterations, before a change can be identified, so our requirement for this set of experiments to be considered successful was that the inspected service be dropped from the top 10 in less that 50 iterations.

We can see that even though the degrading service is at the top of the rankings in the beginning of the experiment, recommenders very quickly realize of the degrading QoS provided and this is reflected in the rankings. In as few as nine iterations, the service is not part of the top 10 of the external user's ranking. Further drop in the rankings requires a bit longer, since most users don't use the service after it drops in the rankings. Another issue to be mentioned regarding this set of experiments is that the external user identifies the degrading service quicker as they only consult the experts of the network (it has no other connection to the network). The random subsets, either the one that is fixed throughout the experiment or the one that changes in every iteration, take longer to figure out that something is wrong, since there might be a few iterations before the majority of their members have used a service. However, the eviction of the degrading service by the subsets is a clear indication that the entirety of the network is now aware of the issue.

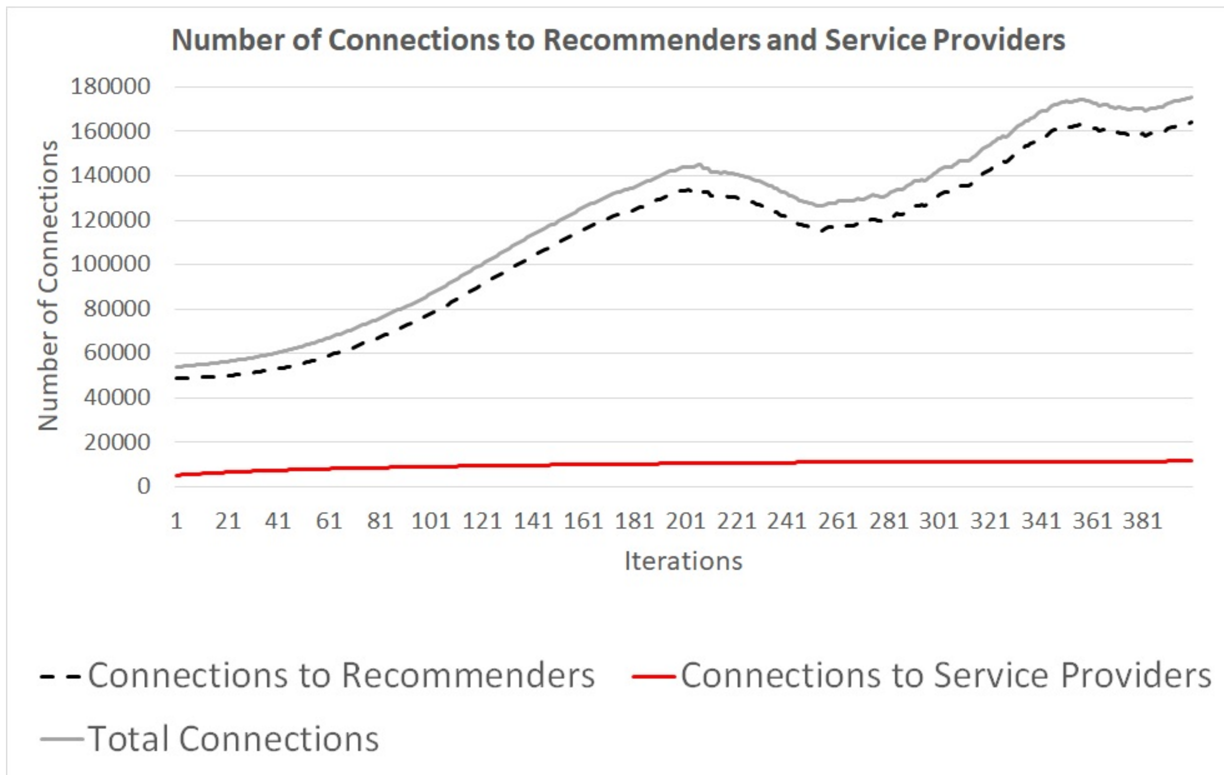

Figure 6.3: Connections to recommenders and service providers

6.2.5 Connections to Recommenders and Service Providers

In an attempt to further evaluate the behaviour of our framework, with respect to contributions *C2* and *C6* and concerning its ability to evolve over time in a dynamic and adapting manner, we run a set of experiments to evaluate the number of connections towards both recommenders and service providers as the system operates. We extended the length of the experiment to 400 iterations, in order to get a better insight on the way the system behaves. The network is constructed with the same characteristics (see Section 6.2.2) , but we do not perform a separate stabilization phase (ee Section 6.2.2) for these experiments, as we want to monitor all the connections created and removed throughout the execution.

The expectations from this set of experiments were pretty straightforward. Connections to service providers are expected to plateau at a specific number, signifying that opinions for all available services have been obtained by current user. Connections to recommenders, however,

are supposed to change in an unpredictable way, since the interactions within the simulator are randomly selected and obsolete values are removed after a certain amount of time.

The results are shown in Fig.6.3 and the trends are evident. As expected, the connections to service providers climb and stabilize around a certain number (lower line in Fig. 6.3) depending on the network's specifics, indicating that the best service providers are discovered from the totality of the framework's users and eventually no more connections are required. As far as connections to recommenders are concerned, the data depict a different story. Since in this set of experiments no malicious components are present, all recommenders in the system are considered honest and good recommenders. In this context, and due to the system's dynamic nature, the number of connections to recommenders is generally increasing, but throughout the experiment certain connections become stale and therefore are deemed obsolete and removed using the algorithm presented in Section 4.4. This is shown in the downward slopes of the middle line in Fig. 6.3.

Note that, even though the process applies to service providers as well, there are way fewer services than recommenders in the network. Moreover, their performance is usually more distinct, as opposed to recommenders whose reputation might be extremely close to each other, thus leading to a more dynamic choice of friends. This behaviour is welcome, since maintaining knowledge of well-behaving services is important, but the sources of recommendation do not have to remain unchanged. If anything, evolution of the social graph is a desirable trait for the system.

6.2.6 Sources of Recommendation

Another interesting dimension of the proposed framework is the pluralism of recommendations from a variety of sources, as specified in contribution *C5*, where the multiple sources of recommendation, combined with the evaluation of both positive and negative evidence, lead to a better evaluation of the ranking of services.

In order to evaluate and assess said polyphony, the nature of the recommenders, and corre-

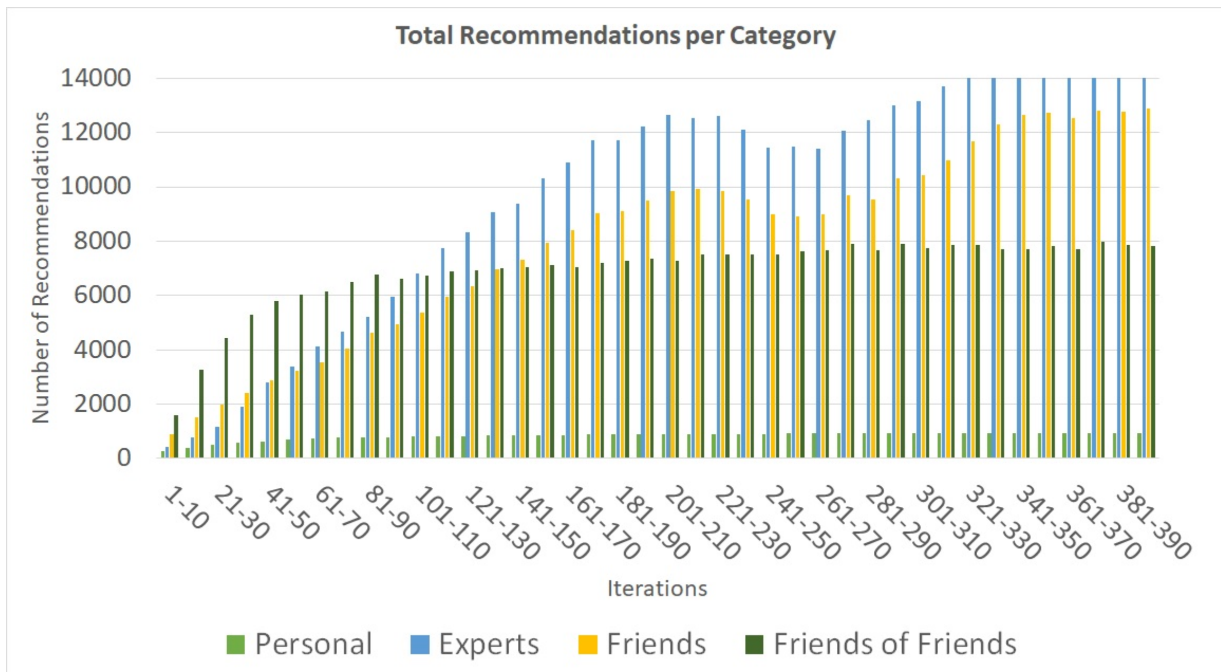

Figure 6.4: Recommendation Sources

sponding recommendations that led to the choice of a specific service provider, were identified through yet another set of experiments that consisted of 400 iterations. Again, the network was created but no *bootstrapping* phase was performed.

Again, as in the previous set of experiments, we expected service clients to acquire opinions for all service providers at some point, and recommendations from other recommenders to fluctuate in a trend following the one portrayed by the presence of connections with other service clients.

Results were summed up in intervals of 10 iterations, so as to make the figure more legible. As seen in Fig. 6.4, recommendations from all types of recommenders are taken into consideration throughout the system's run.

Both recommendations stemming from previous *personal experience* and ones coming from *Friends of Friends* increase over time, until a certain threshold is reached. For personal opinions, the reason is that only one recommendation can be provided per service, so once each service client has connections to the best performing services, their opinion will always

be considered, but with a cardinality of one. When it comes to recommendations from *Friends of Friends*, the limit is specified through a parameter within the framework. Since exploring more paths of the social graph would increase the complexity of the process, the user can set a maximum number of recommendations coming from that source. Once this limit is reached, the ranking algorithm does not consider any more opinion from *Friends of Friends*. This indicates that the system reaches a stable point where a certain amount of connections has already been made and enough information is derived from such sources.

As expected, when it comes to friends and experts, the numbers mostly follow the trend provided by the connections to other recommenders (see Fig.6.3). Recommendations originating both from *Friends* and *Experts* are initially increasing, but stale values are removed at some point, temporarily reducing the number of opinions received. This is indicative of the dynamic behaviour of our approach, since connections and corresponding recommendations are not set in stone but constantly evolve and change.

At any point, though, a variety of sources are providing recommendations, allowing the proposed system to adapt to changes in an efficient and adaptive manner. As described in Section 4.7.2, different "buckets" are created per recommendation source and the values are aggregated, since the Dempster-Shafer evidence theory is sensitive to amount of evidence and opinions coming from *Friends* or *Experts* would overwhelm the ranking otherwise.

6.2.7 Effect of Considering Distrust Values

Our proposed approach offers the ability to maintain two distinct and individually evaluated values pertaining to positive and negative evidence (i.e. trust and distrust values) for each service provider. This significantly improves the proposed framework's expressivity with regards to its capability of defining both wanted and unwanted characteristics of a service.

The ranking process is, also, affected by the presence of both positive and negative evidence. Maintaining a pair, instead of a single value, allows our approach to provide belief intervals for each service that are much more in tune with the actual behaviour demonstrated

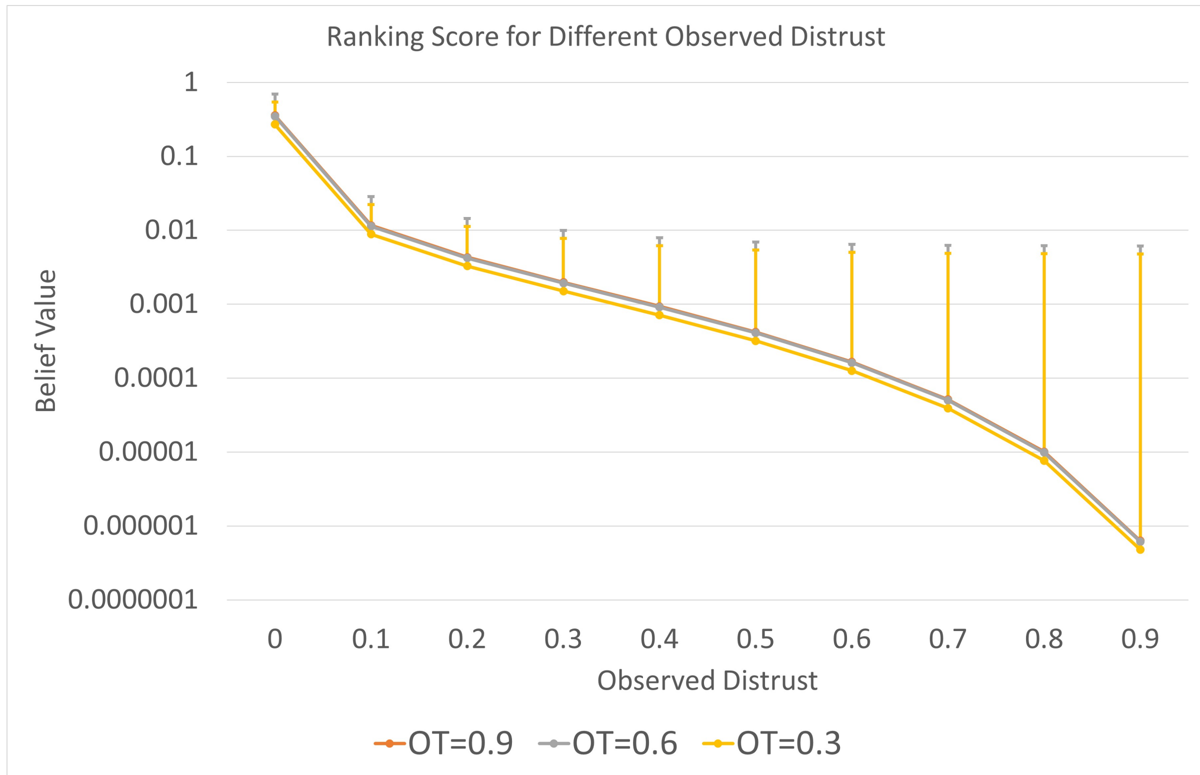

Figure 6.5: Ranking Score for Different Observed Distrust

by it.

To verify the effect an individual distrust value has on the belief interval calculated for a service, affecting both contributions *C1* and *C5*, we conducted a set of experiments that change distrust values portrayed by different service providers and measure the belief intervals calculated.

Once more, a network was created, as indicated in Section 6.2.2, and the bootstrapping phase, as described in Section 6.2.2, was run to allow the system to reach an equilibrium.

Three services were chosen based on the assigned QoS, and more specifically the observed trust value they would provide to service clients after each interaction. The first one would report an *OT* value of 0.9 as its baseline, with the second one providing a value of 0.6 and the third one a value of 0.3. Those particular services were selected to account for services with *high*, *medium* and *low* quality of service.

The system would, then, be allowed to run for 50 iterations and an external user (i.e. a user who never participates in the network but only consults *Experts* for recommendations) would be consulted to provide the calculated belief intervals for the services in question. An average of belief intervals reported in each iteration was calculated and provided as the reported value in each part of the experiment. This process was run for varying *distrust* values portrayed by each of the selected service providers. *Observed distrust* values ranged between 0.0 and 0.9 to allow for verification of its effect on the produced belief intervals.

Since the importance of negative evidence was already known based on the theory of evidence provided by Dempster-Shaffer [22] and adapted by us for the purposes of our approach, we expected a significant change in the calculated belief interval as the values of *observed distrust* would increase. This would indicate a significant contribution of said value in the process of evaluating and ranking available service providers.

The results obtained from this set of experiments can be observed in Fig. 6.5 and are very indicative of the involvement of *distrust* values in calculated ranking scores.

As expected, a change of even 0.1 in *distrust* values, observed after interactions with selected service providers, results in noteworthy variations in the produced belief interval. Note here that the axis pertaining to the belief interval values is logarithmic, so the actual adjustments are much larger in absolute numbers. Another important finding occurring from these experiments is that, even though, the lower end of the belief interval (i.e. the one represented by the lines in the provided figure) is significantly affected by the changes in observed distrust values, the upper end (i.e. the one demonstrated through the error bars in the graph) is only significantly altered for lower distrust values, but remains fairly unaffected when they further increase.

6.2.8 Incentives

Since a service's behaviour might be improved over time, and said improvement can be identified by users through the offer of incentives and the corresponding choice to give providing

service another chance, it is important to evaluate our framework's ability to identify changes in a timely manner.

To verify our proposed approach's ability to utilize incentives, with respect to contribution *C7*, and provide opportunities for new or reformed service providers to be selected over their already established counterparts, we implemented and run experiments that manipulate and monitor such behaviours. More specifically, a network with the characteristics specified in Section 6.2.2 was created and the system was allowed to complete the initial phase described in Section 6.2.2.

Two services were, subsequently, chosen to change their behaviour and be monitored for the duration of the experiments. The first one was the service residing at the bottom of the rankings (i.e. 20th service in the top 20 ranking), whereas the second one was a random one that was entirely out of the rankings. The simulator's functionality would, then, be utilized to improve the baseline behaviour of those services by 20% and 30% respectively.

As far as incentives go, since the concrete implementation of incentives and corresponding reasoning for their provision is out of scope for this paper, we simulated a mock behaviour for the users. Service clients would use the service that was at the top of the rankings in 50% of the cases. Then, to simulate offering of incentives and subsequent choices, they would select the second service in 15% of the cases, the third in 10%, and so on, with the last one selected in only 0.02% of cases and ones not included in the rankings with 0.01% probability.

Two instances of the experiment were run, with each instance considering the rankings provided by different sources. In one case, an external user that has no connections to the network but gets all recommendations from *Experts* was consulted, while in the other one we considered the combined rankings received from a random group of service clients. In both cases, the experiment was allowed to run until the two selected services have reached the top of the rankings (i.e. higher than 5th place).

According to our expectations for that set of experiments, both services (i.e. the reformed and the new one) are supposed to be given a chance to prove themselves. Given their improved

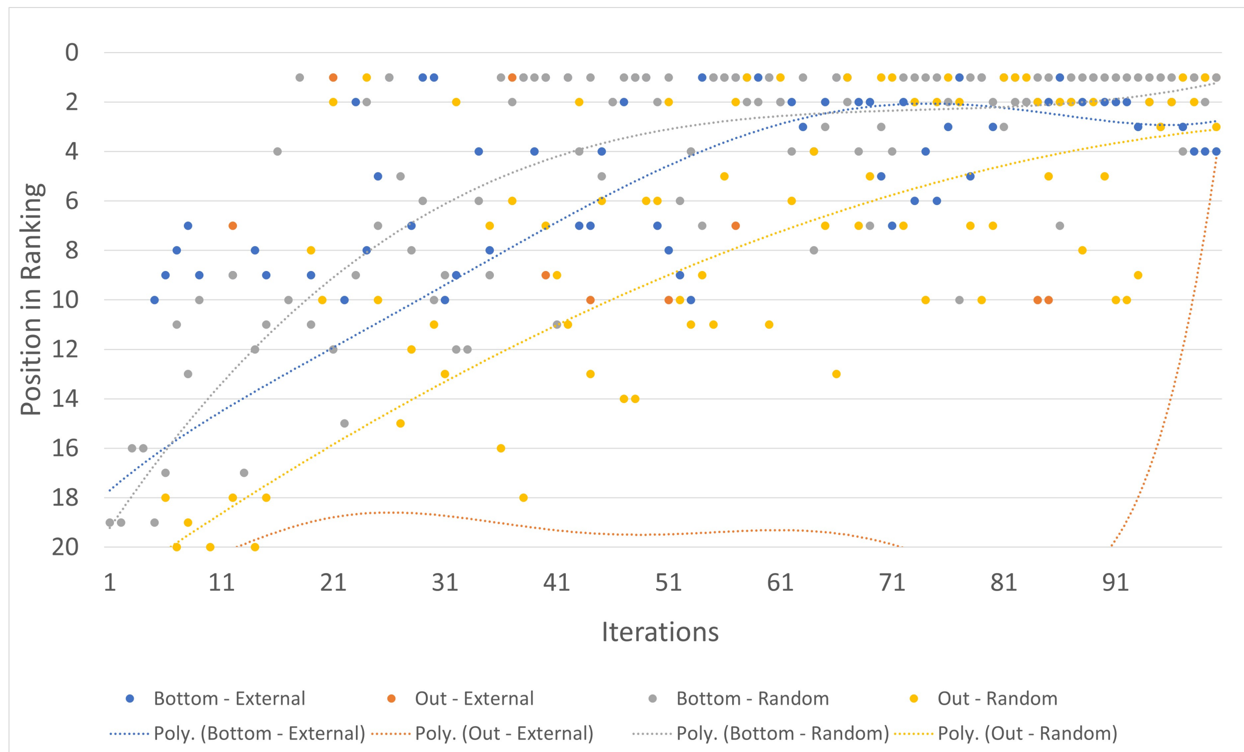

Figure 6.6: Services at bottom and out of ranking

provided QoS, service clients should become aware of said improvements and put the services higher in their computed rankings. Doing this in a timely manner is, also, a requirement for this set of experiments to be considered successful. Since the percentage chosen for an unknown service to be selected based on incentives was very low and users are randomly selected to interact in each iteration, we expected the services to reach the top 3 in about a 100 iterations.

The results can be seen in Fig. 6.6, where the position in the ranking for both services based on both sources is reported, along with a polynomial trendline to facilitate the discovery of ongoing trends.

When it comes to the service originally residing at the bottom of the rankings, we can see that its improved behaviour is identified and it proceeds to climb to the top 5 of the rankings in as few as 40 iterations. The random subset discovers the change a little faster, since the external user consults only the *Experts* within the framework, limiting the sources of information utilized.

The other service, which was initially not part of the provided rankings, requires significantly more iterations to prove its reformation, especially as far as the external user is concerned. This is entirely expected, since the selection percentage (i.e. percentage of interactions were a service is selected over others based on provided incentives) of services that are outside the rankings is much lower. The service, also, has to compete with other services, both in the ranking and out of it. Eventually, though, the service's new behaviour is discovered and it ends up being in the top 3 of produced rankings, in a few as 100 iterations.

Through this set of experiments, it becomes evident that even with very few of the users choosing to accept incentives and utilize the providing service, improved behaviour is identified and the knowledge is propagated quickly within the framework's network, both for services that lower in the ranking and ones that not even part of it. It is worth noting here that, the scenario involving the second service is applicable to a) very badly performing services that alter their behaviour, and b) new services introduced to the system that have no prior connections to service clients. In both cases, their only way of being utilized is through incentives, since no recommendation is offered for them.

6.2.9 Comparison

Regarding contribution *C1*, another set of experiments was run to compare our framework's performance against other approaches that are applicable to similar domains and have similar semantics, especially when malicious users are involved. RATEWeb [19] and PeerTrust-V [90] were chosen because they both make a distinction between service clients and service providers and are geared towards service-oriented needs.

More specifically, for each experiment, we run our system for 200 iterations, with a percentage of users turning malicious after the initial phase described in Section 6.2.2. Each iteration consisted of 100 interactions between random users, regardless of maliciousness status, and service provider selected through the ranking process. Malicious users would provide lower values than those actually observed in each interaction by 20%.

So, we observed 20,000 interactions in total. After each interaction, the opinions of the recommenders of the used service were considered. The average recommended *T/D* values were calculated based on those recommendations and the transaction was evaluated. We would consider a transaction successful if the values observed, through the proxy component for the interaction (*OT/OD* values), were close to the aforementioned average values. The transaction success rate is considered to be the number of successful transactions over the total transaction performed in each experiment (i.e. 20,000).

Note that, same as before, each experiment was run 5 times for each percentage of malicious users, which ranged from 10% to 100%, and the average rate is reported here.

Our main goal was to propose an approach that would outperform existing approaches to date and be able to improve ones ability to identify malicious intent. Improvements of more than 5% were expected, especially in scenarios where the majority of service clients were acting maliciously.

The rate of successful transactions can be seen in Figure 6.7 for our approach, in comparison to the ones reported by RATEWeb [19] and PeerTrust-V [90].

The polyphony of opinions from different sources and selectivity of recommenders to be consulted allows our framework to maintain high rates of success, even when the percentage of malicious users is high.

For lower percentages of malicious users, our framework performs comparably to the other approaches. Not every node's recommendation is taken into consideration, as is the case in other frameworks, but the presence of diverse sources of information provides more than enough information for the user to choose the correct service and have a successful transaction.

When the majority of users are dishonest (i.e. malicious users > 50%), the group of *Expert* recommenders will mostly comprise of malicious users, especially if they are colluding. The ability, however, to a) create new connections and maintain a reputation,even for bad recommenders, b) ask recommendations and personally assess the reputation of *Friends* and *Friends*

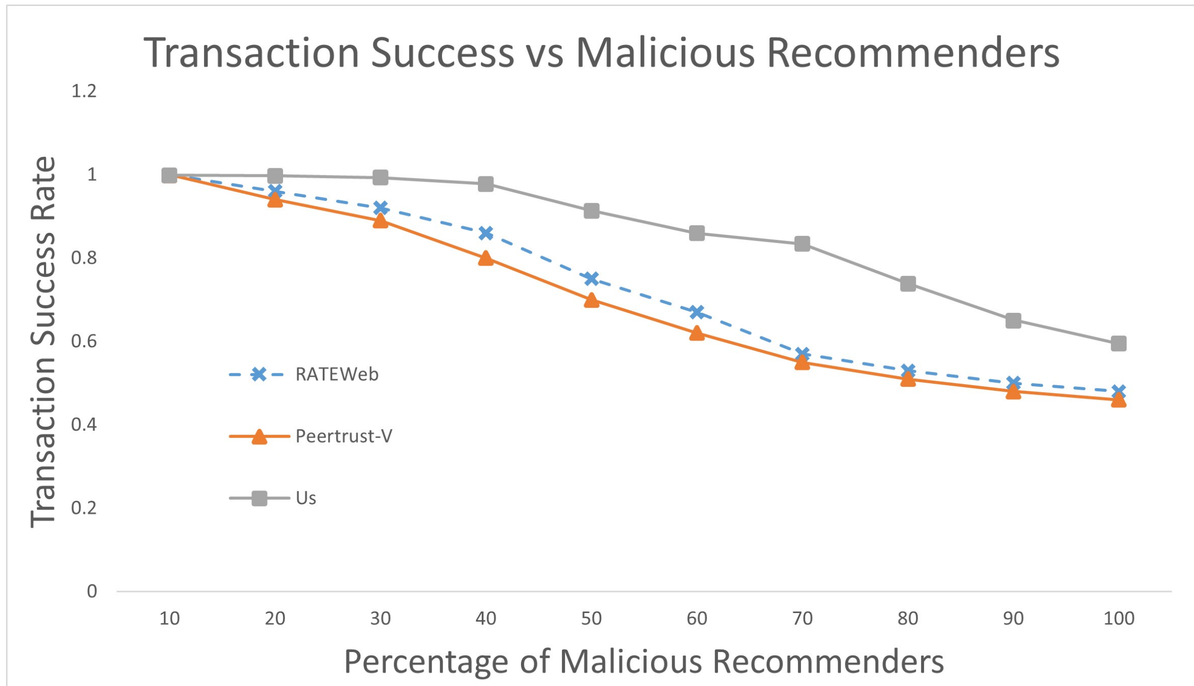

Figure 6.7: Transaction success with respect to malicious users

of Friends nodes, c) choose to consult only the top percentile of users in each category, and d) take personal opinions about service providers into consideration, enables our approach to outperform other approaches in those settings. We can see that, for lower percentages of malicious users, our approach outperforms other frameworks by almost 5%. For higher concentrations of dishonest recommenders, the observed improvement over others jumps to about 20%.

6.3 Summary

Summarizing, in order to evaluate our approach, a prototype version of the proposed framework was implemented. Said prototype was created with scalability, performance and adaptability in mind. All of the novel algorithms and methods presented in Chapter 4 were included and the architecture proposed in Chapter 5 was followed. A complimentary *simulator* was implemented, as well, to allow us to conduct the experiments required for the evaluation of the approach.

Several sets of experiments were run regarding the framework's stability in the presence

of malicious users (see Section 6.2.3) and its ability to detect changes in the behaviour of a service, both when said behaviour is degrading (see Section 6.2.4) and in case of a new or reformed service with the help of incentives (see Section 6.2.8). In all cases, the proposed approach portrayed the ability to identify changes in behaviour, both when the change was caused by intended maliciousness on the recommenders' side and when a service's QoS was altered, in a timely and definitive manner.

The framework's overall way of behaving was, also, investigated through experiments involving the evolution of active connections between service clients and between service clients and service providers (see Section 6.2.5), as well as the nature and importance of different sources of recommendations (see Section 6.4). As it became evident, connections are changing over time, with new ones added and old ones deleted, when necessary, and multiple sources of recommendation are consulted, in varying degrees specific to the particular interaction. These findings further solidify our approach's capability of simulating dynamic behaviour, similar to the one observed in actual human interactions, and avoiding reaching a stale state were no changes are made.

Last but not least, we compared our approach to two other approaches that address similar problems and model their entities in a way similar to ours (i.e. RATEWeb [19] and PeerTrust-V [90]). The transaction success rate of our framework was about 5% higher when the percentages of malicious users were lower, and outperformed the competition by up to 20% when the majority of recommenders were dishonest.

Chapter 7

Conclusion and Future Work

7.1 Summary of the Approach

In this thesis we tackle the problem of assessing and assigning trust when a large number of software components interact in dynamic environments, that is environments where different service clients and service providers can be provisioned and interact in an ad-hoc manner. For this thesis we consider the concept of *trust* in the context of reputation systems. In this respect, *trust* is considered as a meta-requirement related to the belief or disbelief a client has that the provider will deliver the behaviour it promises to deliver.

More specifically, we have presented a technique, associated algorithms, and interaction protocols allowing for service clients to evaluate and assign trust to one or more service providers as a function of the behaviour the different service clients have experienced when using the services offered by the service providers.

The technique is based on the formation of a labelled directed typed multi-graph structure where nodes represent service clients and service providers, and edges represent interaction relations between clients, and between clients and service providers. The interaction relation between clients denote recommendations one client has received from another, while the interaction relation between clients and service providers denote that a service client has used

a service provider. The nodes are labelled by the name/id of the corresponding client or the corresponding service provider, whatever is applicable for a given node. The directed edges, which as we said above, represent interaction relations, are also labelled by values denoting the levels of *trust* and *distrust*, or *reputation*, the source node has assigned on the service provider, or recommender respectively.

The labelled directed typed multi-graph forms a network of interactions. We proposed two types of nodes: *service clients*, and *service providers*. Our approach allows for more than one type of relation to exist between two nodes, hence the multi-graph. We proposed six types of relations and each relation is labelled with a trust value.

The $T_{[client]}^{[provider]}$ relation denotes the overall trust the client has on the provider, based on *all* interactions it has had with the provider, so far. The value assigned for this relation is denoted as $T(client, provider)$.

The $D_{[client]}^{[provider]}$ relation denotes the overall distrust the client has on the provider, based on *all* interactions it has with the provider, so far. The value assigned for this relation is denoted as $D(client, provider)$.

The $R_{[client]}^{[recommender]}$ relation denotes the perceived reputation a client has assigned on another client (i.e. a recommender), indicating how good of a recommender it was, when considering all elapsed interaction the requesting client has had with service providers, as a result of recommendations from said recommender. The value assigned on this relation is denoted as $R(client, recommender)$.

The $AR^{[recommender]}$ relation denotes the *overall reputation* that has, so far, been assigned to a recommender by the framework, and is a function of the individual reputation assigned on the recommender by other clients. That is the $AR^{[recommender]}$ relation denotes the *overall reputation* of the recommender. The value assigned for this relation is denoted as $AR(recommender)$.

The $OT_{[client]}^{[provider]}$ relation is a “helper” relation and denotes the trust the client has on the provider, based on an individual interaction. The value assigned for this relation is denoted as $OT(client, provider)$.

The $OD^{[provider]}_{[client]}$ relation is a "helper" relation and denotes the distrust the client has on the provider, based on an individual interaction. The value assigned for this relation is denoted as $OD(client, provider)$.

After a recommendation by a recommender is followed by a client to use a provider, the *R(client, recommender)* value, corresponding to the relation between the client and the recommender, is updated, or created if it's the first interaction between the two. The *R(client, recommender)* values are used to calculate the overall reputation of the recommender. After a client uses a provider, a pair of values (i.e. $< OT(client, provider), OD(client, provider) >$) are created indicating the level of trust (satisfaction) or distrust (dissatisfaction) a client has on a provider after using the provider in an individual single interaction. The *OT(client, provider)* and *OD(client, provider)* values are, then used to update the *cumulative T(client, provider)* and *D(client, provider)* values (i.e. *trust* and *distrust*) a client has that a provider can deliver what it promises, as well as the *R(client, recommender)* values of those who recommended the utilized service. The final ranking of services, as requested by a client who is looking to use one, is performed by considering all positive and negative recommendations said client has received. The method proposed is a variation of the Dempster-Shaffer evidence theory.

Timeout algorithms are utilized for dealing with old (i.e. stale) values and for updating the recommenders list, thus allowing new recommenders to be considered. In this respect, the approach uses sliding time windows to deal with stale values, and an algorithm based on the Adaptive Replacement Cache (ARC) policy, which has been adapted for the context of this problem.

7.2 Contributions

Referring to the contributions listed in Section 1.3, we outline here how these contributions have been achieved and what is their significance to enhancing the state-of-the-art.

With respect to contributions ***C1*** and ***C2***, the novel algorithms proposed for the continu-

ous evaluation of the trust and distrust values for service providers (see Section 4.2), as well as the individual (see Section 4.3) and global (see Section 4.5) reputation values for service clients, ensure that historical values are considered to a certain degree, allowing for resilience in case of radical but temporary oscillations in the behaviour of nodes in the system. Those algorithms, combined with the consultation of multiple different sources of recommendation and the proposed method of selecting a subset of all available opinions (see Section 3.4), have resulted in our framework's ability to deal with very high percentages of malicious users, a significant improvement compared to approaches up to date. The vast majority of approaches maintain either global or individual reputation values, which hinder their capabilities in one way or the other. Approaches who only utilize global values can only accommodate malicious users, provided they do not constitute the majority of participating entities, as opposed to our approach that can deal with much higher percentages of dishonest participants. The choice to consider just individual values comes with another set of caveats, namely slow adaptation to changes in behaviour and thus slower isolation of malicious users. By using a novel method to utilize a combination of both values, our approach not only gets the best of both worlds, but achieves added benefits resulting from the additional information available. As far as requests for recommendations are concerned, most of the approaches consider all available information to rank, propose and, subsequently evaluate services and corresponding recommenders, with a selected few setting a static threshold. We, on the other hand, propose a novel approach to the selection of recommenders. This allows us to dynamically disregard opinions that the requesting user deems unimportant, as well as opinions that originate from users who have been malicious in the past.

Simulation results (see Chapter 6) indicate that the proposed technique exhibits significant resilience, even when a large number of malicious recommenders (i.e. up to 80%) infiltrates the network, and also allows for degrading service providers to be identified and isolated quickly. In this context, malicious components are shown to be isolated after a short number of interactions, while non-malicious components form their own communities and continue to receive

accurate assessments of available service providers.

Further explaining the contribution mentioned in ***C1***, we proposed the use of different values for service providers (see Section 3.3.2) and recommenders (see Section 3.3.3), as well as the application of distinct algorithms for their evaluation. This approach has allowed us to differentiate between malicious recommenders and malicious service providers, thus further improving the approach's resilience against dishonest parties, and accommodate use cases not covered by most available approaches, which assume that a user's ability to provide recommendations and its servicing capabilities are one and the same. Our approach is applicable to peer-to-peer networks, but is, also, usable in any other kind of scenarios, such as web services, metaverse interactions, social commerce etc. Only a few of the available approaches support separate values for service providers and service recommenders, but even they do not provide completely distinct methods of evaluating them, but rather use the same algorithm or one with minor alterations.

Regarding contribution ***C3*** pertaining to the applicability of our framework in different environments, we proposed a main architecture of our framework, as well as a number of variations of it, that allow its utilization in a variety of different contexts and scenarios. A centralized architecture is proposed for use cases where the system is to be used as part of an application maintained by a central authority, such as an e-commerce website (see Section 5.1). In this variation, users only communicate with the framework through an interface and have access to specific functionality, allowing them to get rankings of services, use the selected one, rate the interaction, if manual input is required, and use be notified about available incentives and compensations. Note that due to the nature of blockchain and smart contracts, the same architecture can be used to deploy the framework in any blockchain platform (see Section 5.3). Even though the architecture is centralized, the semantics of smart contracts allow us to consider the blockchain platform as a central authority and the replication and validation of operations is left to the distributed nature of the platform. Another version of the architecture is, also, provided, where certain alterations are made to allow for the decentralization of the approach (see

Section 5.2). In that variation, the framework is run by every service client participating in the system. Opinions are exchanged through the network and each user only maintains information about their own perceived values. Bear in mind that the system is not fully distributed, since a discovery service is still required to facilitate communication between users who are not acquainted (i.e. have never interacted in the past). The majority of the proposed methods for assigning trust and reputation do not provide a specific architecture for deployment in a specific environment. Most of them, however, are geared towards per-to-peer systems and the implicit assumption is made that an distributed deployment will be used. The communication and collaboration between different users, though, is not explicitly defined, with the exception of a few approaches that deal with specific contexts, such as ad-hoc networks. In our approach, we propose a very concrete set of closely related architectures for deploying the framework in both centralized and decentralized environments. We are, therefore, not constrained to specific uses of our reputation system, in contrast to available approaches so far.

With respect to contribution ***C4***, a specific method and accompanying algorithm was proposed to limit the reputation values considered for the overall reputation values of recommenders (see Section 4.6). In order to specify the individual values that are important to each overall value, a novel adaptation of a policy originating from the field of cache management (i.e. *Adaptive Replacement Cache* policy) was proposed. Frequency and recency are investigated and a subset of opinions is selected and considered towards the calculation of said value. Once more, our framework utilizes an event-driven approach, allowing for higher throughput and smaller network fingerprint. Another advantage of selecting only the values considered important has to do with further supporting the framework's dynamic behaviour. Opinions that are not recent enough or originate from clients who have interacted with a recommender in a sparse and non-consistent way may taint the overall reputation of said recommenders regarding its recommendation capabilities. Our proposed method allows us to disregard values that are old and should bear no significance in the calculation of overall reputation values, as well as maintain incidental values for a much shorter period of time compared to ones coming from

consistent clients. Not all approaches support the notion of global or overall reputation values, and the vast majority of the ones who do consider all available individual values. Very few approaches propose the use of a static threshold for considering opinions that does not capture the ideas of recency or frequency, which is what our approach is based on.

Regarding contribution ***C5*** and the corresponding ranking process, we proposed the use of both positive and negative evidence (i.e. trust and distrust values), which are independently evaluated and considered on their own merits, as well as a ranking technique and corresponding algorithm (i.e. *Dempster-Shafer* adaptation) that takes advantage of the distinct nature of said positive and negative evidence and provides a belief interval, instead of a single value (see Section 4.7). The utilization of a pair of independent values, rather than a single one, allows the user to provide specific requirements both regarding wanted and unwanted behaviours, when it comes to a specific type of service provider, that are evaluated separately and are transformed into discrete evidence. The proposed ranking algorithm takes full advantage of that duality and, after accounting for all available evidence, provides an interval that informs the user about both the *belief* and the *plausibility* of the service being trustworthy. The majority of approaches deal with a single value for prospective service providers, as opposed to our framework that maintains distinct trust and distrust values that are considered as *in-favour* or *against* evidence for a specific service provider. A few of the proposed approaches introduces a value pertaining to risk, but its use does not account for negative evidence, but rather distinction between short and long term opinions. Separate values for trust and distrust are researched in the context of Group Decision Making frameworks. However, they are considered as highly correlated values and are not individually computed, as is the case in our proposed approach. Their ranking process, therefore combines them, rather than treat them as separate evidence and account for the differences in provided information, which is what our proposed method does.

With respect to contribution ***C6*** that addresses the data aging issue, we have provided a novel method to account for stale trust, distrust and reputation values(see Section 4.4). Our *timeout* algorithms, combined with the event-driven approach proposed, allow us to discard

specific values in an efficient and scalable manner. The disposal of old values ensures that the system dynamically adapts to changes in both clients and providers behaviours, thus preventing the underlying network from reaching an unchanging state and failing to accommodate the ever-changing reality of real-world applications. Most approaches do not deal with the issue of data aging at all, while a few select ones choosing to apply a decay function to the entirety of the cumulative values, irrespectively discounting all previous interactions regardless of recency. A couple of approaches, also, propose the disposal of stale values, but, unlike our approach, they maintain a vector of all historical values and recalculate the corresponding cumulative value, thus requiring additional resources. Our approach achieves the same effect with lay less calculations and memory consumption, which allows it to scale and support a high number of users.

With respect to contributions ***C7*** and ***C8***, a few additional features were proposed for dealing with specific circumstances that occur in the context of assigning trust and distrust to different service providers. Firstly, regarding ***C7***, to address the issue of undiscovered behaviour, the use of incentives was proposed (see Section 4.7.4). In case a new service is introduced into the system, or an existing one is reformed, a way to find out about its previously unknown behaviour and provide it with a chance to be selected has to be offered. This need is accommodated by allowing service providers to provide incentives, which will enable them to be chosen over other, already established, service providers. Experiments (see Chapter 6) show that, even for low acceptance rates of incentives, well-performing services are recognized and, after a few iterations, ascend the produced rankings. To our knowledge, the idea of allowing the provision of incentives for selection of a service provider over a more well-established one has not been included in other approaches. Incentives have been used to motivate users to accurately report the quality of service observed after an interaction, but never in the context of advertising improved or not yet discovered service behaviour. Secondly, regarding ***C8***, compensations can be offered to deal with the issue of already elapsed interactions that did not perform as expected, meaning that their provided QoS was much lower than usually expected (see Section 4.7.5).

In such cases, we proposed the use of models to provide compensations to affected users, thus maintaining a stable score if extenuating circumstances were involved. This allows our system to avoid unnecessary fluctuations in service providers' *trust* and *distrust* values, as well as corresponding recommenders' *reputation* values, when said changes are not the result of actual deterioration in quality of provided service, but rather a temporary issue, occurring due to problems that are out of the provider's control. Even though some approaches have dealt with discovering discrepancies in expected vs provided QoS, they mostly deal with other types of systems. None of the proposed reputation systems mention the idea of ameliorating the result of a bad interaction. Our process and architecture allow for the evaluation of compensation models as part of the framework's functionality.

7.3 Discussion of Limitations

The proposed approach is not without challenges, since a few limitations present themselves when accounting for real-world scenarios.

Even though new service providers can be added to the system and be used based on their provided QoS, through utilization of the proposed *incentives* feature, use of new service clients as recommenders is not as straightforward. Service clients joining the system at a later time will still have access to recommendations from other clients, but they will not be able to provide their opinions to other users, except under very specific circumstances. Since they have no individual reputation value assigned to them by another service client, their overall reputation will be set to the default value. If one chooses to set that value to the average or median value observed in the system, said client will never be selected as a recommender by other requesting clients, thus never creating new connections and updating its overall reputation value. If, on the other hand, a higher default value is selected to promote inclusion of new users, the problem of *whitewashing* [139, 140, 141] needs to be addressed, since an inherent advantage is offered to newly introduced users. The *whitewashing* issue involves malicious

users, who connect to the system under a new identity every time their reputation drops below a certain level. In the original version of our approach, no such issue presents itself, since nothing is gained by starting over. The difficulty of incorporating new recommenders into the framework, and receiving recommendations from them, is not very problematic in case of the centralized approach, since no actual node is overwhelmed by requests for opinions. All information are maintained by a central authority and having a minority of recommenders providing most recommendations is not an issue. In the distributed variation, though, a peer could be overwhelmed if the majority of users request its opinions. Solutions to this issue could include further filtering of recommender set based on availability, temporary high overall reputation values for newcomers, or improved communication protocols to alleviate the burden put on *popular* users.

When it comes to evaluating an elapsed interaction with a specific service, each service client defines and utilizes its own model. This behaviour can be perceived as a limitation by some, since different models will, most likely, result in different *trust* and *distrust* values. This, in turn, will affect the reputation a client assigns on other clients (i.e. recommenders), possibly refraining it from trusting their recommendations. That particular outcome, however, is not necessarily an disadvantage of the proposed method. Having a common approach to rating the quality of a specific service is, actually, desirable when receiving recommendations and the ability to individually assign reputation to different clients, based on both their honesty and lack of discrepancy in what one perceives to be a successful transaction, ensures that a client will consult and take into consideration opinions by other clients who think alike. In the long term, this behaviour will result in the creation of communities, meaning that users who have a common way of evaluating services will form stronger connections and hold each other's opinions to a higher standard. This is practically what seems to be happening in the presence of malicious users. Honest clients tend to form close-knit communities, even when they are considered a minority, and receive accurate recommendations from within said community.

Another limitation of our approach has to do with the availability of modelling methods for

incentives and compensations. Even, though, some research approaches have tackled the issue of discovering discrepancies between promised and provided QoS to provide compensations, they mostly deal with specific contexts, such as socio-technical systems. On the other hand, no formal modelling method could be found regarding the provision of promotional incentives. We mocked the behaviour of such models for our experiments, but further research is required to provide efficient and useful ways of creating and evaluating such models.

Lastly, even though the centralized variation of our architecture can be used to deploy our proposed framework in a blockchain platform, maybe a more blockchain-specific approach could be found that will minimize the cost of deployment and required communication between smart contracts representing different parts of the system.

7.4 Future Work

When it comes to future expansion of our proposed approach, a few possible directions can be considered.

In the current approach, the use of a service is considered as an individual and distinct event and is evaluated as one. However, in a lot of real-world applications several services are used as part of a business process. To accommodate for that functionality another layer can be created and superimposed on top of the proposed framework. Said layer would have the ability to model business processes, and their corresponding services, and would allow for evaluation and ranking of different types of services, as part of said business process. Different combinations of services could be provided as options, based on the combined consideration of the individual parts of the model.

Another way our proposed framework could be further extended involves the imposition of certain obligations on service providers. Negotiation of expected behaviour could be performed through model comparison and, upon agreement, specific obligations could be set for every pair of service client and service provider. Of course, violation of obligations would have to incur

a penalty that is either predetermined or evaluated on a per case basis. Said extension is much more viable if the proposed framework is deployed and utilized in the context of a metaverse. Service providers would have to stake a certain amount of the native currency and penalties could result in actual monetary loss on behalf of the service provider.

Trust and reputation have been also researched as part of consensus protocols [100, 101, 102, 103] in the context of different blockchain platforms. All of those approaches, however, focus on solving specific consensus and agreement problems related to crowdsourcing services and IoT devices, where due to the nature of the addressed issues, certain assumptions are made. Those assumptions include the binary result of a transaction (i.e. the transaction is either successful or unsuccessful), and the conviction that opinions of all participants bear the same significance, only weighted differently in case of discrepancies in frequency of transactions they have participated. Our approach allows for a much more granular specification of trust and reputation and more complex underlying social graph. Therefore, a possible area of future work is to investigate the use of reputation-based trust systems to support a consensus or a pseudo-consensus protocol in general blockchain platforms. This has the potential to significantly increase the throughput of such systems, as other computationally intensive consensus algorithms can be applied partially or completely replaced.

Bibliography

[1] Kai Hwang et al. Distributed and Cloud Computing: From Parallel Processing to the Internet of Things, Morgan Kaufmann publishers, 1st edition (2011)

[2] H. Yu, et al., A Survey of Multi-Agent Trust Management Systems. In IEEE Access, vol. 1, pp. 35-50, 2013

[3] S.D. Kamvar et al., The eigentrust algorithm for reputation management in P2P networks, in: Proceedings of the Twelfth International Conference on World Wide Web, ACM, 2003, pp. 640–651 .

[4] Page, L., et al..: The PageRank Citation Ranking: Bringing Order to the Web. Tech. Report, Stanford Digital Library Tech. Project (1998)

[5] Zhou, R., Hwang, K.: Powertrust: a robust and scalable reputation system for trusted peer-to-peer computing. IEEE Trans. Parallel Distributed Syst. 18(4).

[6] Damiani, E.,et al..: A reputation-based approach for choosing reliable resources in peer-to-peer networks. In: ACM Conf. on Computer and Communications Security, pp. 207–216 (2002)

[7] Sabater, J., Sierra, C.: Reputation and social network analysis in multi-agent systems. In: Proc. of the first Intl. Joint Conf. on Autonomous Agents and Multiagent Systems, pp. 475–482, Bologna, Italy (2003)

[8] Aberer, K., Despotovic, Z.: Managing trust in a peer-2-peer information system. In: CIKM, pp. 310–317 (2001)

[9] C. Tian, B. Yang, R2Trust: a reputation and risk based trust management framework for large-scale, fully decentralized overlay networks, Fut. Gen. Comput. Syst. 27 (2011) 1135–1141

[10] Z.Q. Liang, W.S. Shi, PET: a personalized trust model with reputation and risk evaluation for P2P resource sharing, in: Proceedings of the 38th International Conference on System Science, Hawaii, USA, 2005.

[11] S.S. Song, K. Hwang, R.F. Zhou, Trusted P2P transactions with fuzzy reputation aggregation, IEEE Internet Computing (2005) 18–28.

[12] M. Wang, F. Tao, Y. Zhang, G. Li, An adaptive and robust reputation mechanism for P2P network, in: Proceedings of the IEEE International Conference on Communications, Cape Town, South Africa, 2010.

[13] Buchegger, S., Le Boudec, J.-Y.: Performance analysis of the CONFIDANT protocol. In: Proc. of the 3rd ACM Intl. Symposium on Mobile Ad Hoc Networking and Computing, pp. 226–236, June 9–11 (2002)

[14] Rocha, B.G., Almeida, V., Guedes, D.: Increasing qos in selfish overlay networks. IEEE Internet Comput. 10(3), 24–31 (2006)

[15] Maximillien, E.M., Singh, M.P.: Conceptual model of web service reputation. SIGMOD Record 31(4), 36–41 (2002)

[16] Maximilien, E.M., Singh, M.P.: A framework and ontology for dynamic web services selection. IEEE Internet Comput. 8(5), 84–93 (2004)

[17] Maximilien, E.M., Singh, M.P.: Toward autonomic web services trust and selection. In: ICSOC 2004: Proceedings of 2nd International Conference on Service Oriented Computing, November (2004)

[18] Maximilien, E.M., Singh, M.P.: Agent-based trustmodel involving multiple qualities. In: Proc. of 4th International AAMAS, July (2005)

[19] Z. Malik, A. Bouguettaya, RATEWeb: reputation assessment for trust establishment among web services, VLDB J. 18 (2009) 885–911.

[20] R. Guha, et al., Propagation of trust and distrust, in: Proc. of the 13th Intl. Conference on World Wide Web, 2004, pp. 403–412 .

[21] P. Bonacich , P. Lloyd , Eigenvector-like measures of centrality for asymmetric relations, Soc. Netw. 23 (2001) 191–201.

[22] Glenn Shafer, A mathematical theory of evidence, Princeton Univ. Press, Princeton, New Jersey, 1976

[23] Horkoff J, et al., Goal-oriented requirements engineering: an extended systematic mapping study. Requir Eng. 2019;24(2):133-160.

[24] G. Chatzikonstantinou, K. Kontogiannis: Efficient parallel reasoning on fuzzy goal models for run time requirements verification. Softw. Syst. Model. 17(4).

[25] Megiddo, Nimrod and Dharmendra S. Modha. “ARC: A Self-Tuning, Low Overhead Replacement Cache.” FAST (2003).

[26] Jon Crowcroft, Richard Gibbens, Frank Kelly, Sven Östring, Modelling incentives for collaboration in mobile ad hoc networks, in: Performance Evaluation, Volume 57, Issue 4, 2004, pp. 427-439

[27] Jeffrey Banks, Sridhar Moorthy, A model of price promotions with consumer search, in: International Journal of Industrial Organization, Volume 17, Issue 3, 1999, pp. 371-398

[28] William Rand, Roland T. Rust, Agent-based modeling in marketing: Guidelines for rigor, in: International Journal of Research in Marketing, Volume 28, Issue 3, 2011, pp. 181-193

[29] Dalpiaz, F., Giorgini, P., Mylopoulos, J., Adaptive socio-technical systems: a requirements-based approach, in: Requirements Eng 18, 1–24 (2013).

[30] P. Victor , C. Cornelis , M. De Cock , P. Pinheiro da Silva , Gradual trust and distrust in recommender systems, Fuzzy Sets Syst. 160 (2009) 1367–1382 .

[31] Lamport, Leslie, The Part-Time Parliament, ACM Transactions on Computer Systems 16, 2 (May 1998), 133-169.

[32] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung and P. Verissimo, ”Efficient Byzantine Fault-Tolerance,” in IEEE Transactions on Computers, vol. 62, no. 1, pp. 16-30, Jan. 2013.

[33] Hendrikx, F., Bubendorfer, K., Chard, R., Reputation systems: A survey and taxonomy. Journal of Parallel and Distributed Computing. 75. (2014).

[34] P. Resnick, R. Zeckhauser, Trust among Strangers in Internet Transactions: Empirical Analysis of eBay's Reputation System, Advances in Microeconomics: A Research Annual 11 (2002) 127-157.

[35] P. Resnick, K. Kuwabara, R. Zeckhauser, E. Friedman, Reputation Systems, Communications of the ACM 43 (2000) 45-48.

[36] eBay, http://www.ebay.com/, last accessed 2022-06-17.

[37] P. Resnick, R. Zeckhauser, J. Swanson, K. Lockwood, The Value of Reputation on eBay: A Controlled Experiment, Experimental Economics 9 (2003) 79-101.

[38] A. Josang, R. Ismail, C. Boyd, A Survey of Trust and Reputation Systems for Online Service Provision, Decision Support Systems 43 (2) (2007) 618-644.

[39] E. Koutrouli, A. Tsalgatidou, Taxonomy of Attacks and Defense Mechanisms in P2P Reputation Systems - Lessons for reputation system designers, Computer Science Review 6 (2) (2012) 47-70.

[40] J. Sabater, C. Sierra, REGRET: Reputation in Gregarious Societies, in: AGENTS '01: Proceedings of the Fifth international conference on Autonomous agents, ACM, New York, NY, USA, 2001, pp. 194-195.

[41] M. Gupta, P. Judge, M. Ammar, A Reputation System for Peer-to-Peer Networks, in: NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video, ACM, New York, NY, USA, 2003, pp. 144-152.

[42] E. Koutrouli, A. Tsalgatidou, Reputation-based trust systems for P2P applications: design issues and comparison framework, Trust and Privacy in Digital Business (2006) 152-161.

[43] J. Sabater, C. Sierra, Review on Computational Trust and Reputation Models, Artificial Intelligence Review 24 (2005) 33-60.

[44] K. Hoffman, D. Zage, C. Nita-Rotaru, A Survey of Attack and Defense Techniques for Reputation Systems, ACM Computing Surveys 42 (1) (2009) 1-31.

[45] F. Hendrikx, K. Bubendorfer, Malleable Access Rights to Establish and Enable Scientific Collaboration, in: eScience (eScience), 2013 IEEE 9th International Conference on, IEEE, Beijing, China, 2013, pp. 334-341.

[46] F. Hendrikx, K. Bubendorfer, Policy Derived Access Rights in the Social Cloud, in: eScience (eScience), 2013 IEEE 9th International Conference on, IEEE, Beijing, China, 2013, pp. 365-368.

[47] Amazon, http://www.amazon.com/, last accessed 2022-06-17.

[48] D. Houser, J. Wooders, Reputation in auctions: Theory, and evidence from eBay, Journal of Economics and Management Strategy 15 (2) (2006) 353-369.

[49] M. I. Melnik, J. Alm, Does a Seller's eCommerce Reputation Matter? Evidence from eBay Auctions, The journal of industrial economics 50 (3) (2002) 337-349.

[50] Y. Wang, J. Vassileva, Toward Trust and Reputation Based Web Service Selection: A Survey, International Transactions on Systems Science and Applications 3 (2) (2007) 118-132.

[51] S. Marti, H. Garcia-Molina, Taxonomy of Trust: Categorizing P2P Reputation Systems, Computer Networks 50 (4) (2006) 472-484.

[52] A. Schlosser, M. Voss, L. Bruckner, On the Simulation of Global Reputation Systems, Journal of Artificial Societies and Social Simulation 9 (2005) 1.

[53] A. Abdul-Rahman, S. Hailes, Supporting trust in virtual communities, in: System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on, IEEE, 2000, pp. 9-18.

[54] Y. Yuan, S. Ruohomaa, F. Xu, Addressing common vulnerabilities of reputation systems for electronic commerce, Journal of theoretical and applied electronic commerce research 7 (1) (2012) 1-20.

[55] R. Ismail, C. Boyd, A. Josang, S. Russell, An Efficient Off-Line Reputation Scheme Using Articulated Certificates, in: WOSIS-2004: Proceedings of the Second International Workshop on Security in Information Systems, 2004, pp. 53-62.

[56] P. Dewan, P. Dasgupta, Pride: Peer-to-Peer Reputation Infrastructure for Decentralized Environments, in: WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers and posters, ACM, New York, NY, USA, 2004, pp. 480-481.

[57] G. Zacharia, A. Moukas, P. Maes, Collaborative reputation mechanisms for electronic marketplaces, Decision Support Systems 29 (4) (2000) 371-388.

[58] A. Josang, R. Ismail, The Beta Reputation System, in: Proceedings of the 15th bled electronic commerce conference, 2002, pp. 41-55.

[59] J. Patel, W. L. Teacy, N. R. Jennings, M. Luck, A Probabilistic Trust Model for Handling Inaccurate Reputation Sources, in: Trust Management, Springer, 2005, pp. 193-209.

[60] Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, Guangquan Zhang, Recommender system application developments: A survey, Decision Support Systems, Volume 74, 2015, Pages 12-32.

[61] M. Pazzani, D. Billsus, Content-based recommendation systems, in: P. Brusilovsky, A. Kobsa, W. Nejdl (Eds.), The Adaptive Web, Springer, Berlin Heidelberg 2007, pp. 325–341.

[62] R. Burke, Hybrid recommender systems: survey and experiments, User Modeling and User-Adapted Interaction 12 (2002) 331–370.

[63] M. Deshpande, G. Karypis, Item-based top-N recommendation algorithms, ACM Transactions on Information Systems (TOIS) 22 (2004) 143–177.

[64] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, Item-based collaborative filtering recommendation algorithms, Proceedings of the 10th International Conference on World Wide Web, ACM 2001, pp. 285–295.

[65] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, J. Riedl, GroupLens: an open architecture for collaborative filtering of netnews, Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work, ACM, Chapel Hill, North Carolina, USA 1994, pp. 175–186.

[66] Q. Shambour, J. Lu, A hybrid trust-enhanced collaborative filtering recommendation approach for personalized government-to-business e-services, International Journal of Intelligence Systems 26 (2011) 814–843.

[67] B. Smyth, Case-based recommendation, in: P. Brusilovsky, A. Kobsa, W. Nejdl (Eds.), The Adaptive Web, Springer, Berlin Heidelberg 2007, pp. 342–376.

[68] S. Middleton, D. Roure, N. Shadbolt, Ontology-based recommender systems, in: S. Staab, R. Studer (Eds.), Handbook on Ontologies, Springer, Berlin Heidelberg 2009, pp. 779–796.

[69] I. Cantador, A. Bellogín, P. Castells, A multilayer ontology-based hybrid recommendation model, AI Communications 21 (2008) 203–210.

[70] X. Amatriain, A. Jaimes, N. Oliver, J. Pujol, Data mining methods for recommender systems, in: F. Ricci, L. Rokach, B. Shapira, P.B. Kantor (Eds.), Recommender Systems Handbook, Springer, US 2011, pp. 39–71.

[71] K. Yu, V. Tresp, S. Yu, A nonparametric hierarchical bayesian framework for information filtering, Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, Sheffield, United Kingdom 2004, pp. 353–360

[72] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, Z. Chen, Scalable collaborative filtering using cluster-based smoothing, Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, Salvador, Brazil 2005, pp. 114–121

[73] K.-j. Kim, H. Ahn, A recommender system using GA K-means clustering in an online shopping market, Expert Systems with Applications 34 (2008) 1200–1209.

[74] A. Zenebe, A.F. Norcio, Representation, similarity measures and aggregation methods using fuzzy sets for content-based recommender systems, Fuzzy Sets and Systems 160 (2009) 76–94.

[75] D. Ben-Shimon, A. Tsikinovsky, L. Rokach, A. Meisles, G. Shani, L. Naamani, Recommender system from personal social networks, Advances in Intelligent Web Mastering, Springer, 2007. 47–55.

[76] P. Massa, P. Avesani, Trust-aware collaborative filtering for recommender systems, On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, Springer, 2004. 492–508.

[77] J.A. Golbeck, Computing and Applying Trust in Web-based Social Networks, University of Maryland, 2005

[78] L. Quijano-Sanchez, J.A. Recio-Garcia, B. Diaz-Agudo, G. Jimenez-Diaz, Social factors in group recommender systems, ACM Transactions on Intelligent Systems and Technology (TIST), 4 2013, pp. 1–30.

[79] M. O'Connor, D. Cosley, J. Konstan, J. Riedl, PolyLens: a recommender system for groups of users, in: W. Prinz, M. Jarke, Y. Rogers, K. Schmidt, V. Wulf (Eds.), European Conference on Computer Supported Cooperative Work 2001, Springer, Netherlands 2002, pp. 199–218.

[80] J. Masthoff, Group modeling: selecting a sequence of television items to suit a group of viewers, User Modelling and User-Adapted Interaction 14 (2004) pp.37–85.

[81] L. Penserini, A. Perini, A. Susi, J. Mylopoulos, High variability design for software agents: Extending Tropos,in: TAAS, 2(4), 2007.

[82] Dardenne A, van Lamsweerde A, Fickas S "Goal-directed requirements acquisition". Sci Comput Program 20(1–2):3–50. 1993.

[83] Yu ESK "Towards modelling and reasoning support for early-phase requirements engineering". In: Proceedings of the 3rd IEEE international symposium on requirements engineering (RE'97). Washington, DC. 1997.

[84] Sotirios Liaskos, Shakil M. Khan, Mikhail Soutchanski, and John Mylopoulos. 2013. Modeling and Reasoning with Decision-Theoretic Goals. In Proceedings of the 32nd International Conference on Conceptual Modeling - Volume 8217 (ER 2013). Springer-Verlag, Berlin, Heidelberg, 19–32.

[85] Silva Souza, V., Lapouchnian, A., Robinson, W., Mylopoulos, J., Awareness Requirements for Adaptive Systems, in: SEAMS (2011). 60-69.

[86] Michalis Bachras, et al., "Goal Modelling Meets Service Choreography: A Graph Transformation Approach". In Proc. of EDOC'20, pp. 30-39.

[87] Fischer, M., Lynch, N. and Paterson, M. (1985). Impossibility of distributed consensus with one faulty process. Journal of the ACM, Vol. 32, No. 2, pp. 374–82.

[88] Chandra, T. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. Journal of the ACM. Vol 43, No. 2, pp. 225–67.

[89] Coulouris, G. F. (2011). Consensus and related problems. In Distributed systems: Concepts and design. essay, Addison-Wesley. pp. 659-670.

[90] Xiong, L., Liu, L.: PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities. IEEE Trans. Knowl. Data Eng. (TKDE) 16(7), 843–857 (2004)

[91] Resnick, P., Zeckhauser, R.: Trust among strangers in internet transactions: empirical analysis of eBay's reputation system. Adv. Appl. Microecon. 11, 127–157 (2002)

[92] P. J. Denning, "Working sets past and present," IEEE Trans. Software Engineeing, vol. SE-6, no. 1, pp. 64–84, 1980.

[93] A. V. Aho, P. J. Denning, and J. D. Ullman, “Principles of optimal page replacement,” J. ACM, vol. 18, no. 1, pp. 80–93, 1971.

[94] Giamanco; Barbara; Gregoire, Kent (2012). ”Tweet me, friend me, make me buy” (PDF). Harvard Business Review. 90 (7): 89–93. Archived from the original (PDF) on 2014-12-13.

[95] Jensen, C.D. (2014). The Importance of Trust in Computer Security. In: Zhou, J., Gal-Oz, N., Zhang, J., Gudes, E. (eds) Trust Management VIII. IFIPTM 2014. IFIP Advances in Information and Communication Technology, vol 430. Springer.

[96] Rose, Scott, et al. Zero trust architecture. No. NIST Special Publication (SP) 800-207. NIST, 2020.

[97] Lv, S., Li, H., Wang, H., Wang, X. (2020). CoT: A Secure Consensus of Trust with Delegation Mechanism in Blockchains. In: , et al. Blockchain Technology and Application. CBCC 2019. Communications in Computer and Information Science, vol 1176. Springer.

[98] Jennings, N.R. (2002). Agent-Based Computing. In: Musen, M.A., Neumann, B., Studer, R. (eds) Intelligent Information Processing. IIP 2002. IFIP — The International Federation for Information Processing, vol 93. Springer, Boston, MA.

[99] Park M. and Kim Y-G, A Metaverse: Taxonomy, Components, Applications, and Open Challenges. In IEEE Access, vol. 10, pp. 4209-4251, 2022.

[100] Zou, J., Ye, B., Qu, L., Wang, Y., Orgun, M.A., Li, L 2019, ’A Proof-of-Trust consensus protocol for enhancing accountability in crowdsourcing services’, IEEE Transactions on Services Computing, vol. 12, no. 3, pp. 429-445.

[101] H. Chai, S. Leng, K. Zhang and S. Mao, ”Proof-of-Reputation Based-Consortium Blockchain for Trust Resource Sharing in Internet of Vehicles,” in IEEE Access, vol. 7, pp. 175744-175757.

[102] Jingyu Feng, Xinyu Zhao, Guangyue Lu, and Feng Zhao. 2019. PoTN: A Novel Blockchain Consensus Protocol with Proof-of-Trust Negotiation in Distributed IoT Networks. In Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things (IoT S&P'19). Association for Computing Machinery, New York, NY, USA, 32–37.

[103] X. Zhu, Y. Li, L. Fang and P. Chen, "An Improved Proof-of-Trust Consensus Algorithm for Credible Crowdsourcing Blockchain Services," in IEEE Access, vol. 8, pp. 102177-102187

[104] SENTZ, KARI, and FERSON, SCOTT. 2002. "Combination of Evidence in Dempster-Shafer Theory". United States.

[105] Thierry Denźux. 2016. 40 years of Dempster-Shafer theory. Int. J. Approx. Reasoning 79, C (December 2016), 1–6.

[106] T. Denoeux, "A neural network classifier based on Dempster-Shafer theory," in IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 30, no. 2, pp. 131-150, March 2000

[107] Denoeux, T. (2008). A k-Nearest Neighbor Classification Rule Based on Dempster-Shafer Theory. In: Yager, R.R., Liu, L. (eds) Classic Works of the Dempster-Shafer Theory of Belief Functions. Studies in Fuzziness and Soft Computing, vol 219. Springer

[108] T. M. Chen and V. Venkataramanan, "Dempster-Shafer theory for intrusion detection in ad hoc networks," in IEEE Internet Computing, vol. 9, no. 6, pp. 35-41, Nov.-Dec. 2005

[109] J.A. Malpica, M.C. Alonso, M.A. Sanz, Dempster–Shafer Theory in geographic information systems: A survey, Expert Systems with Applications, Volume 32, Issue 1, 2007, Pages 47-55

[110] Mamdani, E.H. (1974). ”Application of fuzzy algorithms for control of simple dynamic plant”. Proceedings of the Institution of Electrical Engineers. 121 (12): 1585–1588.

[111] Takagi, Tomohiro; Sugeno, Michio (January 1985). ”Fuzzy identification of systems and its applications to modeling and control”. IEEE Transactions on Systems, Man, and Cybernetics. SMC-15 (1): 116–132.

[112] Han-Xiong Li, Lei Zhang, Kai-Yuan Cai and Guanrong Chen, ”An improved robust fuzzy-PID controller with optimal fuzzy reasoning,” in IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 35, no. 6, pp. 1283-1294, Dec. 2005

[113] Hong Peng, Jun Wang, Mario J. Pérez-Jiménez, Hao Wang, Jie Shao, Tao Wang, Fuzzy reasoning spiking neural P system for fault diagnosis, Information Sciences, Volume 235, 2013, Pages 106-116

[114] Yan, R., Yu, Y., Qiu, D. Emotion-enhanced classification based on fuzzy reasoning. Int. J. Mach. Learn. and Cyber. 13, 839–850 (2022).

[115] Meimei Gao, M. Zhou, Xiaoguang Huang and Zhiming Wu, ”Fuzzy reasoning Petri nets,” in IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 33, no. 3, pp. 314-324, May 2003

[116] N. Naik and P. Jenkins, ”Enhancing Windows Firewall Security Using Fuzzy Reasoning,” 2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), 2016, pp. 263-269

[117] Emanuel Onica, Pascal Felber, Hugues Mercier, and Etienne Rivière. 2016. Confidentiality-Preserving Publish/Subscribe: A Survey. ACM Comput. Surv. 49, 2, Article 27 (June 2017), 43 pages.

[118] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The many faces of publish/subscribe. ACM Comput. Surv. 35, 2 (June 2003), 114–131.

[119] T. Ding, S. Qian, J. Cao, G. Xue and M. Li, ”SCSL: Optimizing Matching Algorithms to Improve Real-time for Content-based Pub/Sub Systems,” 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2020, pp. 148-157

[120] Y. Zhao, Y. Li, Q. Mu, B. Yang and Y. Yu, ”Secure Pub-Sub: Blockchain-Based Fair Payment With Reputation for Reliable Cyber Physical Systems,” in IEEE Access, vol. 6, pp. 12295-12303.

[121] Ibrahim, M., Rehfeldt, K. and Rausch, A., Conception of a Type-based Pub/Sub Mechanism with Hierarchical Channels for a Dynamic Adaptive Component Model. ADAPTIVE 2018, pp. 53-59

[122] Chelloug, S. (2015) Energy-Efficient Content-Based Routing in Internet of Things. Journal of Computer and Communications, 3, 9-20.

[123] Q. Wang, D. Chen, N. Zhang, Z. Ding and Z. Qin, ”PCP: A Privacy-Preserving Content-Based Publish–Subscribe Scheme With Differential Privacy in Fog Computing,” in IEEE Access, vol. 5, pp. 17962-17974, 2017.

[124] S. Tarun, R. S. Batth and S. Kaur, ”A Review on Fragmentation, Allocation and Replication in Distributed Database Systems,” 2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), 2019, pp. 538-544.

[125] J. Domaschka, C. B. Hauser and B. Erb, ”Reliability and Availability Properties of Distributed Database Systems,” 2014 IEEE 18th International Enterprise Distributed Object Computing Conference, 2014, pp. 226-233.

[126] Ali Davoudian, Liu Chen, and Mengchi Liu. 2018. A Survey on NoSQL Stores. ACM Comput. Surv. 51, 2, Article 40 (March 2019), 43 pages.

[127] Bonomi, F., Milito, R., Natarajan, P., Zhu, J. (2014). Fog Computing: A Platform for Internet of Things and Analytics. In: Bessis, N., Dobre, C. (eds) Big Data and Internet of Things: A Roadmap for Smart Environments. Studies in Computational Intelligence, vol 546. Springer, Cham.

[128] DECENCIÈRE, Etienne et al. FEEDBACK ON A PUBLICLY DISTRIBUTED IMAGE DATABASE: THE MESSIDOR DATABASE. Image Analysis and Stereology, [S.l.], v. 33, n. 3, p. 231-234, aug. 2014.

[129] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. 2011. Big data and cloud computing: current state and future opportunities. In Proceedings of the 14th International Conference on Extending Database Technology (EDBT/ICDT '11). Association for Computing Machinery, New York, NY, USA, 530–533.

[130] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2013. Spanner: Google's Globally Distributed Database. ACM Trans. Comput. Syst. 31, 3, Article 8 (August 2013), 22 pages.

[131] Dmitry Efanov, Pavel Roschin, The All-Pervasiveness of the Blockchain Technology, Procedia Computer Science, Volume 123, 2018, Pages 116-121.

[132] Urena, R., Kou, G., Dong, Y., Chiclana, F. and Herrera-Viedma, E., 2019. A review on trust propagation and opinion dynamics in social networks and group decision making frameworks. Information Sciences, 478, pp.461-475.

[133] J. Wu , F. Chiclana , E. Herrera-Viedma , Trust based consensus model for social network in an incomplete linguistic information context, Appl. Soft. Comput. 35 (2015) 827–839.

[134] J. Wu , L. Dai , F. Chiclana , H. Fujita , E. Herrera-Viedma , A minimum adjustment cost feedback mechanism based consensus model for group decision making under social network with distributed linguistic trust, Inf. Fusion 41 (2018) 232–242.

[135] J. Wu , L. Dai , F. Chiclana , H. Fujita , E. Herrera-Viedma , A new consensus model for social network group decision making based on a minimum adjustment feedback mechanism and distributed linguistic trust, Inf. Fusion 41 (2018) 232–242.

[136] J. Wu , R. Xiong , F. Chiclana , Uninorm trust propagation and aggregation methods for group decision making in social network with four tuple infor- mation, Knowl. Based Syst. 96 (2016) 29–39.

[137] R.R. Yager , Quantifier guided aggregation using OWA operators, Int. J. Intell. Syst. 11 (1996) 49–73.

[138] R. Ureña, F. Chiclana, E. Herrera-Viedma, DeciTrustNET: A graph based trust and reputation framework for social networks, Information Fusion, Vol.61, 2020, pp.101-112.

[139] M. Feldman, C. Papadimitriou, J. Chuang and I. Stoica, ”Free-riding and whitewashing in peer-to-peer systems,” in IEEE Journal on Selected Areas in Communications, vol. 24, no. 5, pp. 1010-1019, May 2006.

[140] J. Chen, H. Lu and S. D. Bruda, ”A Solution for Whitewashing in P2P Systems Based on Observation Preorder,” 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing, 2009, pp. 547-550.

[141] Shabnam Seradji, Mehran S. Fallah, A Bayesian Game of Whitewashing in Reputation Systems, The Computer Journal, Volume 60, Issue 8, August 2017, Pages 1223–1237.

[142] X. Zeng, R. Bagrodia and M. Gerla, ”GloMoSim: a library for parallel simulation of large-scale wireless networks,” Proceedings. Twelfth Workshop on Parallel and Distributed Simulation PADS ’98 (Cat. No.98TB100233), 1998, pp. 154-161

[143] Schlosser, Mario T., Tyson E. Condie, Sepandar D. Kamvar, and Ar D. Kamvar. ”Simulating a P2P file-sharing network.” In First workshop on semantics in p2p and grid computing, pp. 69-79. 2002.

[144] https://influencermarketinghub.com/metaverse-stats/, last accessed 2022-08-03.

[145] https://www.nytimes.com/2022/07/19/business/amazon-fake-reviews-lawsuit.html, last accessed 2022-08-03.

[146] https://www.bloomberg.com/professional/blog/metaverse-may-be-800-billion-market-next-tech-platform/, last accessed 2022-08-03.

[147] https://earthweb.com/how-many-people-use-the-metaverse/, last accessed 2022-08-03.

[148] Kubernetes, https://kubernetes.io/, last accessed 2022-08-03.