Gefedereerd en autonoom beheer van multimediadiensten

Federated and Autonomic Management of Multimedia Services

Jeroen Famaey

Promotoren: prof. dr. ir. F. De Turck, prof. dr. ir. B. Dhoedt
Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Computerwetenschappen

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. D. De Zutter
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2011 - 2012

UNIVERSITEIT
GENT

ISBN 978-90-8578-523-1
NUR 986, 988
Wettelijk depot: D/2012/10.500/49

Universiteit Gent
Faculteit Ingenieurswetenschappen & Architectuur
Vakgroep Informatietechnologie

Promotoren: prof. dr. ir. Filip De Turck
prof. dr. ir. Bart Dhoedt

Universiteit Gent
Faculteit Ingenieurswetenschappen & Architectuur

Vakgroep Informatietechnologie
Gaston Crommenlaan 8 bus 201, B-9050 Gent, België

Tel.: +32-9-331.49.00
Fax.: +32-9-331.48.99

Dit werk kwam tot stand in het kader van een specialisatiebeurs van het IWT-Vlaanderen (Instituut voor de aanmoediging van Innovatie door Wetenschap en Technologie in Vlaanderen).

Proefschrift tot het behalen van de graad van Doctor in de Ingenieurswetenschappen: Computerwetenschappen
Academiejaar 2011-2012

Dankwoord

Het dankwoord schrijven is typisch één van de laatste taken die je als doctorandus uitvoert. Eens je hieraan begint weet je dat het einde sowieso nabij is. Ik vind het dan ook de perfecte gelegenheid om even terug te blikken op de jaren dat ik door het leven ben gegaan als doctoraatsstudent. In die periode, van toch wel bijna 5 jaar, heb ik heel wat bijgeleerd, zowel op technisch als persoonlijk vlak. Hetgeen me het meest zal bijblijven, op niet-technisch vlak, is zeker en vast dat wat je samen bereikt meer is dan de som van wat je afzonderlijk verwezelijkt. Hoewel het doctoraat op zich iets persoonlijks is, is de weg ernaartoe dus alles behalve een *one-man show*. Veel van de ideeën die je terugvindt in dit boek zijn dan ook het gevolg van intensieve discussies en samenwerkingen met anderen. Ik draag de laatste woorden die ik als doctoraatsstudent schrijf dan ook graag aan hen op.

Vooreerst zou ik mijn promotoren, Filip De Turck en Bart Dhoedt, willen bedanken voor het vertrouwen dat ze in me hebben gesteld, de interessante onderzoeksideeën die ze hebben aangereikt en de gedetailleerde feedback die ze hebben gegeven. Kortom, zonder hun intensieve begeleiding zou mijn doctoraat niet zijn geweest wat het nu is. Filip zou ik in het bijzonder willen bedanken om me in te wijden in de wondere wereld van het netwerkbeheer en me in contact te brengen met tal van gerenommeerde onderzoekers in dit vakgebied. Aangezien promotoren drukbezette mensen zijn, ben ik blij dat ik tijdens de beginperiode van mijn doctoraat voor dagdagelijkse begeleiding, problemen en vragen terecht kon bij twee ervaren collega's: Bart De Vleeschauwer en Tim Wauters. Bart en Tim, bedankt voor de uitmuntende begeleiding tijdens het schrijven van mijn IWT-aanvraag en de wenperiode erna. Daarbovenop is het mede dankzij Bart's nauwgezette opvolging van mijn masterproef dat ik gestart ben met een doctoraat. Dankzij de succesvolle IWT-aanvraag heb ik mijn doctoraatsonderzoek kunnen uitvoeren vrij van financiele bekommernissen, waarvoor mijn dank dan ook uit gaat naar het agentschap voor Innovatie door Wetenschap en Technologie (IWT). Ook Piet Demeester verdient hier een vermelding, aangezien hij me de kans heeft gegeven om mijn doctoraatsonderzoek uit te voeren aan de IBCN onderzoeksgroep. Finally, I would like to thank John Strassner for his guidance and sharing his years of experience in the field of autonomic network management. Our conversations and his in-depth feedback on my work have been a great source of inspiration for me.

Naast mijn promotoren is er één collega die een onuitwisbare nadruk heeft nagelaten op mijn onderzoek en bijgevolg dit boek: Steven Latré. Het is dan ook geen toeval dat zijn naam op de auteurslijst van zo goed als elk hoofdstuk prijkt. Tijdens MANWEEK 2008 (wat een toepasselijke naam), op het exotische

Samos, ontdekten we dat onze onderzoeksinteresses toch wel heel dicht bij elkaar aanleunden. Ironisch genoeg luidde de (tot nog toe) laatste vergadering van het mythische Autonomic Communications Forum (ACF), die we daar allebei hebben bijgewoond, de start van onze samenwerking in. De vele vergaderingen, brainstorms en gesprekken die we sindsdien gevoerd hebben zijn een grote bron van inspiratie geweest bij het uitvoeren van mijn onderzoek. Steven, je bent een top-onderzoeker, top-collega en top-vriend!

Tijdens het uitvoeren van mijn doctoraat heb ik ook de kans gekregen om in een breder kader onderzoek te doen en samen te werken met collega onderzoekers in de context van verscheidene Europese en Belgische projecten. Na de start van mijn doctoraat belandde ik reeds snel op het Europese project ALPHA. Hier leerden Filip en Bart Lannoo me de kneepjes van het vak. De allereerste projectvergadering in een afgelegen middeleeuws kasteel in een donker Scandinavisch bos zit nog vers in mijn geheugen. Meer recent kwam ik terecht op een reeks projecten rond multimedianetwerken: RUBENS, OCEAN, Mistral en CDN-i. Dit boeiende onderwerp is uiteindelijk ook een belangrijke rol gaan spelen in mijn eigen onderzoek. Ik heb hierrond dan ook fijn samengewerkt met heel wat collega's: Bart D.V., Steven, Wim Van de Meerssche, Tim W., Stijn Melis, Frédéric Iterbeke, Klaas Roobroeck en Niels Bouten. Hoewel we niet meteen hebben samengewerkt in de context van een officieel onderzoeksproject, zou ik toch ook onze locale ontologie-experts, Stijn Verstichel en Femke Ongenae, willen bedanken. Ze stonden steeds paraat om mijn vragen en problemen rond ontologieën (en semantiek in het algemeen) te beantwoorden en op te lossen.

In het kader van wetenschappelijke congressen en voorgenoemde Europese projecten heb ik tal van (niet altijd even) exotische locaties bezocht. Ik zou dan ook mijn vaste reisgezellen, Filip en Steven, willen bedanken voor het leuke gezelschap tijdens deze trips. In een poging tot volledigheid, zou ik ook mijn andere reisgezellen willen bedanken: Bart L., Jelle Nelis, Maarten De Grootte, Tim W., Niels Sluijs, Philip Leroux, Bruno Van Den Bossche, Klaas, Hendrik Moens, Tim De Pauw, Matthias Strobbe en Niels B. Ik moet eerlijk toegeven dat ik heel wat leuke, grappige, serieuze en ronduit bizarre herinneringen overhoud aan deze reizen. Ik denk bijvoorbeeld met veel plezier terug aan rammelende propellervliegtuigen, gigantische Long Island parkings, toiletgrappen en stoffige vulkanen.

Naast onderzoek heb ik me tijdens mijn doctoraat ook gewaagd aan onderwijs. Ik heb met veel plezier de practica en projecten van het vak ODS begeleid. Ik wil het hele team dan ook bedanken voor de toffe samenwerking en het gezellig samen doden van de tijd tijdens vraagloze momenten. Bedankt Kristof Steurbaut, Steven, Stijn V., Matthias, Femke O., Bert Vankeirsbilck, Samuel Dauwe, Femke De Backere, Dieter Verslype, Hendrik, Niels B., Tim Verbelen en Klaas.

Doorheen de jaren heb ik heel wat administratieve en technische hindernissen moeten overwinnen. In het bijzonder zou ik Martine Buysse en Davinia Stevens willen bedanken voor de ontelbare administratieve vragen en problemen waarbij ze me hebben geholpen. Verder zou ik zeker ook de admins (Bert D.V., Pascal, Johan, Wouter, Bert D.K., Jonathan, Serge en Joeri), de Virtual Wall goeroes (Jos en Maarten) en de mensen van financiën (Bernadette, Karien, Nathalie, Joke en

Dalila) willen bedanken voor de ondersteuning. Tenslotte verdienen ook Sandra en Sabrina hier een vermelding, voor het proper houden van de Zuiderpoort.

De sfeer op de werkvloer is natuurlijk ook van groot belang. Dat ik mijn doctoraat mocht uitvoeren in een leuke en gezellige omgeving heb ik grotendeels te danken aan mijn eilandgenootjes: Femke O. en Femke D.B. Starten met een doctoraat is toch een beetje een stap in het onbekende: veel nieuwe gezichten en een vreemde omgeving. Ik was (en ben) Femke O. dan ook uiterst dankbaar dat we deze stap samen hebben kunnen zetten en we gaandeweg elkaar steeds hebben kunnen steunen en helpen. Daarbovenop is ze als drijvende kracht achter de vele 2.20 bureauactiviteiten een echte sfeermaker. Femke D.B. kwam er later bij. Ik denk dat ik gerust kan stellen dat we, sinds ze ons eiland heeft vervoegd, goede vrienden zijn geworden. Ik ben haar dan ook oprecht dankbaar voor het gezelschap, de leuke babbels bij een koffie (en ice-tea), de steun, de motiverende woorden en natuurlijk het nalezen van een aantal belangrijke secties van dit boek! Naast mijn eilandgenootjes zou ik ook mijn andere (ex-)bureaugenoten willen bedanken voor hun niet te onderschatten bijdrage tot de aangename werkomgeving: Bruno Volckaert, Sofie Verbrugge, Tom Verdickt, Nico Goeminne, Gregory Van Seghbroeck, Bert, Samuel, Dirk De Schrijver, Dirk Gorissen, Minh Huu Nguyen, Wannes Kerckhove, Thomas Dupont, Hendrik, Ali Farhan Azmat, Lien Deboosere en Pol Dockx. Tenslotte konden we de laatste jaren tijdens onze (uitgebreide) bureauactiviteiten ook nog rekenen op het gezelschap van een aantal andere collega's: Steven, Kristof, Stijn V., Philip, Bram Gadeyne en Niels B.

Het leven is natuurlijk meer dan werken alleen. Op tijd en stond even ontspannen en de zinnen verzetten, is goed voor de motivatie en inspiratie. Ik ben mijn vrienden dan ook dankbaar om hiervoor te zorgen, in de vorm van gezellige etentjes, toffe spelletjesavonden en leuke cafébezoekjes. Bedankt Annelies & Jeroen, Bart, Bert, David, Davy, Dimitri, Femke & Karel, Femke & Stephen, Jeroen, Michael, Ruben & Ellen, Steven & Sara, Stéphanie en Toon & Gijs. Verder zou ik nog graag mijn (schoon)familie en in het bijzonder mijn ouders, zus en petekindje Noor uitvoerig willen bedanken. Ze hebben steeds in me geloofd, me gemotiveerd om ervoor te gaan en staan altijd voor me klaar. Zonder hun onvoorwaardelijke steun zou ik dit dankwoord nu waarschijnlijk niet aan het schrijven zijn. Daarbovenop waren de gezellige familiewandelingen (en nog meer de terrasbezoekjes en bijhorende bitterballen) op zondagnamiddag een leuke afleiding.

Mensen die me goed kennen, weten dat ik geen man van tradities ben. Desondanks zou toch graag de traditie om in de laatste paragraaf van het dankwoord diegenen te bedanken die me het nauwst aan het hart liggen willen hooghouden. Bij deze: bedankt Veronic en Senne, voor alles en nog veel meer! Veronic, bedankt voor je humor, je uitmuntende relativeringsvermogen, je steun als het even wat moeilijker ging en natuurlijk het delen van je toch wel omvangrijk kennis van de statistiek. Senne, bedankt voor je lach, liefde en genegenheid. Bedankt allebei om me tonen wat echt belangrijk is in het leven...

Gent, 21 juni 2012
Jeroen Famaey

Table of Contents

Dankwoord **i**

Samenvatting **xxxi**

Summary **xxxv**

1 Introduction **1**
1.1 The Internet revolution 1
1.2 Problem statement 4
1.3 Definitions & terminology 6
1.4 Research contributions 8
1.5 Outline of this dissertation 10
1.6 Publications 13
1.6.1 A1: Journal publications indexed by Web of Science . . . 13
1.6.2 P1: Conference proceedings indexed by Web of Science . 14
1.6.3 C1: Other international conference proceedings 15
1.6.4 Other publications 16
References 17

2 Federated management of the Future Internet **19**
2.1 Introduction 20
2.2 Definition 21
2.3 Architectures and models 22
2.3.1 Layered Federation Model 22
2.3.2 Autonomic Internet Architecture 24
2.3.3 Summary and comparison 26
2.4 Status and challenges 27
2.4.1 Security and trust 28
2.4.2 Semantic interoperability 31
2.4.3 Agreement negotiation 34
2.4.4 Resource discovery and matchmaking 36
2.4.5 End-to-end resource configuration 39
2.4.6 Management coordination 40
2.5 Conclusion 43
References 46

3 Federated resource management for end-to-end multimedia services 53
3.1 Introduction . . . 54
3.2 Related work . . . 56
3.2.1 Multi-constrained optimal path problem . . . 56
3.2.2 End-to-end Quality of Service . . . 57
3.2.3 SLA negotiation and management . . . 58
3.3 Federated content delivery framework . . . 59
3.3.1 Stakeholders . . . 59
3.3.2 Interactions . . . 60
3.4 End-to-end content delivery federations . . . 63
3.4.1 Notations & assumptions . . . 63
3.4.2 Problem formulation . . . 67
3.5 End-to-end resource reservation algorithm . . . 68
3.5.1 Finding QoS-constrained core paths . . . 68
3.5.2 Setting up the initial delivery paths . . . 69
3.5.3 Iteratively improving the delivery paths . . . 70
3.5.4 Additional considerations . . . 75
3.6 Results and discussion . . . 76
3.6.1 Evaluation scenario . . . 77
3.6.2 Scalability . . . 79
3.6.3 Storage site merits . . . 81
3.6.4 Cache sharing merits . . . 84
3.6.5 Caching in the access network . . . 84
3.7 Conclusion . . . 87
References . . . 89

4 Towards a predictive cache replacement strategy 93
4.1 Introduction . . . 94
4.2 Related work . . . 95
4.3 Predicting content popularity . . . 96
4.4 Predictive cache replacement strategy . . . 99
4.5 Results & discussion . . . 100
4.5.1 Evaluation scenario . . . 101
4.5.2 Prediction algorithm evaluation . . . 101
4.5.3 Predictive cache replacement strategy evaluation . . . 106
4.6 Conclusion . . . 109
References . . . 111

5 A hierarchical approach to autonomic network management 115
5.1 Introduction . . . 116
5.2 FOCALE autonomic management architecture . . . 117
5.3 Hierarchical management architecture . . . 118
5.3.1 Cluster management . . . 120
5.3.2 Context dissemination . . . 121
5.3.3 Policy interaction . . . 123

5.3.4 Autonomic element collaboration 124
5.3.5 Management algorithms 125
5.4 Evaluation 125
5.4.1 Analytical model 126
5.4.2 Results 128
5.5 Conclusion & future work 131
References 133

6 Semantic context dissemination and service matchmaking 137
6.1 Introduction 138
6.2 Related work 141
6.2.1 Semantic publish/subscribe systems 141
6.2.2 Semantic service matchmaking 144
6.3 Semantic communications bus architecture 145
6.4 Use case: cloud infrastructure management 147
6.4.1 Scenario description 147
6.4.2 Ontological concepts 147
6.5 Semantic context dissemination 150
6.5.1 OWL filter rules 152
6.5.2 SWRL filter rules 153
6.5.3 Jena filter rules 154
6.6 Semantic service matchmaking 155
6.6.1 Subsumption relationships 156
6.6.2 Inputs & outputs 158
6.6.3 Preconditions & effects 159
6.6.4 Variable binding matches 160
6.6.5 Illustrative examples 161
6.7 Evaluation & results 163
6.7.1 Implementation & evaluation set-up 164
6.7.2 Filter rule subscription 165
6.7.3 Context publication 167
6.7.4 Service matchmaking 170
6.8 Conclusion 171
References 173

7 A hierarchical context dissemination framework for clouds 177
7.1 Introduction 178
7.2 Related work 180
7.2.1 Federated network management 180
7.2.2 Semantics in network management 182
7.3 Hierarchical context dissemination 183
7.3.1 Architectural overview 183
7.3.2 Resource management 184
7.3.3 Cluster management 184
7.3.4 Domain management 185

7.3.5 Federated management . . . 185
7.3.6 Context continuum . . . 186
7.4 Autonomic element architecture . . . 186
7.4.1 Knowledge base . . . 186
7.4.2 Context disseminator . . . 188
7.4.3 Autonomic manager . . . 189
7.4.4 Service repository . . . 189
7.4.5 Contract repository . . . 190
7.4.6 Policy framework . . . 190
7.5 Managing the cloud . . . 191
7.5.1 Cloud management . . . 192
7.5.2 Data center management . . . 195
7.5.3 Server management . . . 197
7.6 Results & discussion . . . 198
7.6.1 Prototype implementation . . . 199
7.6.2 Context dissemination scalability . . . 199
7.6.3 Semantic reasoning overhead . . . 203
7.7 Conclusion & future work . . . 208
References . . . 210

8 Conclusions and research perspectives 215
8.1 End-to-end Quality of Service guarantees . . . 215
8.2 Management complexity of the Internet . . . 217
8.3 Research perspectives . . . 219
8.3.1 Discovery of network domains and capabilities . . . 219
8.3.2 Inter-domain information models . . . 219
8.3.3 Optimized delivery of multimedia services . . . 220
8.3.4 Generic self-governance . . . 220
8.3.5 Policy refinement . . . 220

A Context authoring for federated autonomic management 223
A.1 Introduction . . . 224
A.2 Related work . . . 225
A.3 Exchanging context in a federated architecture . . . 226
A.4 Use case: multimedia video delivery . . . 229
A.5 Context authoring process details . . . 231
A.5.1 Detailed process requirements . . . 232
A.5.2 Computational overview . . . 233
A.6 Performance evaluation . . . 243
A.6.1 Experimental setup . . . 243
A.6.2 Network model . . . 243
A.6.3 Evaluation result details . . . 246
A.7 Conclusions . . . 251
References . . . 253

B Hierarchical application placement algorithm in large scale clouds 257
B.1 Introduction . . . 258
B.2 Related work . . . 259
B.3 System architecture . . . 260
B.4 Formal problem description . . . 262
B.5 Hierarchical management . . . 263
B.5.1 Hierarchical management structure . . . 263
B.5.2 Algorithm details . . . 264
B.6 Evaluation results . . . 267
B.7 Conclusion . . . 273
References . . . 275

C Ambient-aware care through semantic context dissemination 277
C.1 Introduction . . . 278
C.1.1 Background . . . 278
C.1.2 Ambient-aware continuous care . . . 278
C.1.3 Related work . . . 279
C.1.4 Objective & organization . . . 281
C.2 Architecture of the ACCIO platform . . . 281
C.3 Use case: Ontology-based Nurse Call System . . . 284
C.3.1 Scenario description . . . 284
C.3.2 Co-creation methodology . . . 285
C.3.3 Resulting continuous care core ontologies . . . 286
C.3.4 Flexible and semantic publish/subscribe mechanism . . . 288
C.4 Implementation details & results . . . 291
C.5 Conclusion . . . 296
References . . . 297

List of Figures

1.1 The growth of the Internet over the years in terms of the number of Autonomous Systems; In fifteen years time, the Internet has increased twentyfold in size (data obtained from [2]) 2

1.2 Categorization of global Internet traffic in the recent past and as expected in the near future; By 2015, multimedia services are expected to make up over 60% of all Internet traffic (data obtained from [4]) 3

1.3 Evolution of the relative cost of hard- and software compared to the management thereof; The management cost of computing and communications systems has increased drastically over the years, due to their increased complexity and size (data obtained from [5]) 4

1.4 Schematic overview of the contributions of this dissertation; The focus of the different chapters and appendices is highlighted . . . 11

2.1 Layered Federation Model (from [7]) 23

2.2 Autonomic Internet (AutoI) Architecture (from [9]) 25

2.3 The identified challenges and their relationship to each other and the underlying network 29

3.1 An overview of the stakeholders involved in the end-to-end content delivery process: content providers, storage sites, access ISPs and transit ISPs 59

3.2 A sequence diagram detailing the negotiation process between the content provider and an access ISP customer 61

3.3 A sequence diagram detailing the interactions involved in finding an end-to-end delivery path from the content provider to an access ISP 62

3.4 Flowchart depicting the steps and flow of the resource reservation algorithm; The sections in which the steps are described are denoted in parentheses 69

3.5 A graphical example of how expanded path set candidates are created; The path set $\widehat{\Pi}_o$ is transformed into a new path set $\widehat{\Pi}_{o'}$; The dotted lines represent paths containing zero or more content caches 72

3.6 A graphical example of how a group of path sets can be combined into a single path set with a shared cache; The original set of path sets $\{\widehat{\Pi}\}_s$ is transformed into a new combined path set $\widehat{\Pi}_{o'}$; The dotted lines represent paths containing zero or more content caches 73
3.7 Execution time of the algorithm 80
3.8 Influence of the storage cost sc on the merits of intermediary content caches; comparing end-to-end QoS-aware content delivery with and without intermediary storage sites 82
3.9 Influence of the storage site vicinity sv on the merits of intermediary content caches; comparing end-to-end QoS-aware content delivery with and without intermediary storage sites 83
3.10 Influence of the access ISP vicinity av on the efficiency of cache sharing 85
3.11 Influence of caching in the access domain on caching efficiency of the storage sites 86

4.1 Optimal fit of the exponential and gaussian distributions to an example cumulative request pattern; the vertical line represents the current point in time t, on its left is the 400-hour known history, on its right the 48-hour predicted future 99
4.2 A graphical representation of the Video on Demand dataset . . . 102
4.3 The absolute prediction error averaged over all objects as a function of the history request pattern length for $W = 1$ hour 103
4.4 The absolute prediction error of the exponential fit for different prediction windows (in hours) as a function of the history request pattern length 105
4.5 The processing time required to fit a single distribution to a request pattern as a function of the history request pattern length 106
4.6 The cache hit rate as a function of the prediction window W for different cache sizes C 108
4.7 Comparison of the different cache replacement strategies in terms of cache hit rate, as a function of cache size for $W = 12$ hours . . 109

5.1 The FOCALE control loops [8] 118
5.2 A hierarchical autonomic element manages a set of managed resources and child autonomic elements, together forming the set of *managed entities*; The autonomic elements are structured in a logical tree topology 119
5.3 A cluster can be dynamically split into several sub-clusters to improve scalability 121
5.4 Public context of an AE is aggregated and filtered before it is disseminated to the parent AE; Private context is not made available to the parent 122
5.5 High-level policies are propagated down the cluster hierarchy; The AEs can be mapped to the views of the Policy Continuum 123

5.6 Mapping of different hierarchical structures onto an exemplary network topology, spanning multiple domains. 124
5.7 An example hierarchical AE topology, with $R = 9$, $C = 3$, and $L = 3$ 127
5.8 The total overhead per AE (in megabits per second) as a function of the neighbour count N (for the flat architecture) and the cluster size S (for the hierarchical architecture) 129
5.9 The maximum received context per AE (in megabits per second) as a function of the neighbour count N (for the flat architecture) and the cluster size S (for the hierarchical architecture) 130
5.10 The total overhead per AE (in megabits per second) as a function of the size of the network R 131

6.1 An overview of the interactions between the network's resources, management components and human operators 140
6.2 The SCB plays a central role in the interactions between AEs and network resources; its core ontologies are used in the matchmaking and context dissemination processes 146
6.3 An overview of classes representing the physical resources of the cloud management use case; they are connected to the logical resources through the *VirtualMachine* class 148
6.4 An overview of classes representing the logical resources of the cloud management use case 149
6.5 An overview of the classes representing measured values 150
6.6 A detailed overview of the context dissemination process and the involved actors 151
6.7 A flowchart depicting the five steps of the semantic matchmaking algorithm 156
6.8 The evolution of total reasoning time as more OWL filter rules are added to the SCB 165
6.9 The evolution of total reasoning time as more SWRL filter rules are added to the SCB 166
6.10 The evolution of total reasoning time as more Jena filter rules are added to the SCB 166
6.11 The evolution of total reasoning time as more OWL filter rules are added to the SCB with satisfiability and consistency checks turned off 167
6.12 Average reasoning time for publishing a single message on the SCB, as a function of number of subscribers. 168
6.13 Average reasoning time for publishing a single message on the SCB, as a function of number of message complexity. 169
6.14 Average reasoning time for publishing a single message on the SCB, as a function of number of filter rule complexity. 169

6.15 The evolution of total reasoning time as a function of the number of offered service descriptions, for different percentages of matching descriptions (*mp*) 171

7.1 An overview of the collaborative, AE-driven, hierarchical management architecture 183
7.2 Overview of the internal AE architecture and its interactions with other AEs and managed resources 187
7.3 The context continuum for resource allocation in cloud computing; context is forwarded and aggregated through the AE hierarchy using dynamic context-aware filter rules 192
7.4 The amount of context received by data center AEs for a 2 level data center hierarchy 202
7.5 The amount of context received by data center AEs for a 3 level data center hierarchy 204
7.6 The reasoning time for generating filter rules 206
7.7 The reasoning time for semantically matching filter rules with context 207

A.1 Overview of a hierarchical network management environment. Arrows represent context exchange between the components. 227
A.2 A bus-driven autonomic element. The components connected to the bus are loosely coupled, and can communicate through the exchange of context. 228
A.3 Details of the context manager within a bus-driven autonomic element illustrated in Fig. 2. By exploiting knowledge from the information model, a context authoring process may generate both local and remote filter rules to enable the context exchange. 229
A.4 Sequence diagram illustrating the exchange of context in a federation with two core networks an access network and home network. The sequence diagram illustrates how a change in filter rules is triggered. 231
A.5 Computational overview of the context authoring process. 233
A.6 Overview of the ontological context model, highlighting the main concepts and the interaction with related ontologies such as DENON-ng and the Time ontology. 235
A.7 An example of an instantiation of the context model for a cloud environment scenario. 236
A.8 The translation of the contextual requirements of a generic rule-based system to the context model. 238
A.9 The algorithm for automatically generating the filter rules based on the context model. 241
A.10 Overview of the fuzzification process for illustrative server load values. The fuzzification results in a significant reduction of the number of instances being stored in the ontological context model. 242

A.11 Schematic overview of the use case's topology, including which types of context are generated where. Local devices such as routers and servers generate local monitoring information, whereas AEs generate derived and/or aggregate context by forwarding the maximum of local measurements. 244
A.12 Influence of OWL reasoning on the reasoning time as a function of time. 247
A.13 Influence of the Summarised OWL, SWRL and Summarised SWRL reasoner configuration sets on the reasoning time as a function of the time. 248
A.14 Influence of an increasing time window HWi in the history based pruning algorithm, defining the history that is stored in the context model, on the reasoning time. Different fuzzification configurations are investigated. 249
A.15 Impact of an increasing number of AEs on the reasoning time. A context exchange with 75 AEs, which represents a network topology of hundreds of thousands of nodes, results in a reasoning time of 478 msec. 251

B.1 The system components on management and execution servers . . 261
B.2 Solving overutilization by splitting a node and promoting a child node (grey) to peer status. 264
B.3 Solving underutilization by removing a node and distributing its children (grey) amongst the node's peers. 264
B.4 The origin and destination of the different $applace$ in- and outputs. A single management server, containing the $applace$-function, aggregation and decoupling mechanisms is shown. 268
B.5 The quality of the allocation after subsequent placement calculations ($C_{max} = 20$, $|S| = 50$). Standard errors are shown as well. . 269
B.6 Execution time of the hierarchical and centralized allocation strategies with varying server and application counts ($|A| = |S|$) 270
B.7 Comparison of the average satisfied demand of the different allocation strategies ($|A| = |S|$) 270
B.8 Illustration of the management overhead and performance penalties induced by the hierarchy with a very low branching factor ($C_{max} = 10$) 271
B.9 Illustration of the management overhead and performance penalties with a higher branching factor ($C_{max} = 100$) 272
B.10 Execution time of the hierarchical and centralized allocation strategies with fixed application counts ($|A| = 50$). Execution time of the centralized algorithm was measured up until 1000 servers. . . 273
B.11 The maximum number of applications known per node at different management levels ($C_{max} = 10$) 274

C.1 Architecture of the ACCIO platform using a Semantic Cummunication Bus (SCB) for interaction, collaboration and orchestration . 282
C.2 General concept of the ontology-based Nurse Call Management System (oNCS) with probabilistic priority assessment and profile management 284
C.3 Overview of the most prevalent classes and their relations of the seven core ontologies for the intelligent nurse call system use case. The dashed arrows depict subclass relationships. The blue arrows represent relationships between concepts and individuals. 286
C.4 Sequence diagram of the first three steps of the implemented scenario: Step 1: Configuration, Step 2: Turn on light when nurse enters the room, Step 3: Register a dynamic filter rule. 293
C.5 Average reasoning time needed to publish, filter and forward one event as a function of the number of filter rules and the percentage of filter rules that match with this event, averaged over 30 iterations 295

List of Tables

3.1 The aggregation and satisfaction operators for commonly used QoS types 64
3.2 The list of symbols used throughout this chapter 66
3.3 The QoS classes used in the evaluation scenario 79

6.1 An example of an offered and requested service description to turn on devices and activate servers respectively 162
6.2 A service description of a management function that allows AEs to allocate physical resources to a virtual machine 163

7.1 A service description of a management function that allows AEs to allocate physical resources to a virtual machine 197

A.1 Overview of the available context types and their granularity. The aggregated context types denote the maximum value reported by the corresponding local context types that are controlled by the AE. 245

B.1 Symbols 262

C.1 Amount of data generated by the sensors in a department with 20 rooms, 30 patients and 10 staff members. 294

List of Acronyms

A

ABox	Assertion Component
ACCIO	Ambient-aware provisioning of Continuous Care for Intramural Organizations
A^3DS	Architectural Artefacts for Autonomic Distributed Systems
AE	Autonomic Element
AMD	Advanced Micro Devices
AMS	Autonomic Management System
API	Application Programming Interface
ARPANET	Advanced Research Projects Agency NETwork
AS	Autonomous System
AutoI	Autonomic Internet
AVC	Advanced Video Coding

B

BGP	Border Gateway Protocol

C

CAPEX	CAPital EXpenditures
CASCADAS	Component-ware for Autonomic Situation-aware Communications, and Dynamically Adaptable Services
CBN	Content-Based Networking
CBPMS	Community-Based Policy Management System
CDN	Content Delivery Network
CPU	Central Processing Unit
CSP	Communications Service Provider

D

DAL	Distributed Authorization Language
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Projects Agency
DEN	Directory Enabled Networks
DEN-ng	Directory Enabled Networks - next generation
DENON-ng	Directory Enabled Networks ONtology - next generation
DHT	Distributed Hash Table
DOC	Distributed Orchestration Component
DRM	Domain Relationship Map

E

eTOM	enhanced Telecom Operations Map

F

FAME	Federated, Autonomic Management of End-to-end communication services
FedRR	Federated Resource Reservation algorithm
FIPA	Foundation for Intelligent Physical Agents
FOCALE	Foundation Observation Comparison Action Learning and rEasoning
FP7	Seventh Framework Programme
FRM	Federal Relationship Manager
FWO	Fund for Scientific Research Flanders

G

GB	GigaByte
GiB	GibiByte
Ghz	Gigahertz
GNU	GNU's Not Unix
G-ToPSS	Graph-based Toronto Publish/Subscribe System

H

HAS	HTTP Adaptive Streaming
HCOME	Human-Centered Ontology engineering MEthodology
H_MCOP	Heuristic Multi-Constrained Optimal Path
HTTP	HyperText Transfer Protocol

I

IBBT	Interdisciplinary Institute for Broadband Technology
IBCN	Internet Based Communication Networks and Services
IEEE	Institute of Electrical and Electronics Engineers
IFIP	International Federation for Information Processing
ILP	Integer Linear Programming
INTEC	Department of Information Technology
IOPE	Inputs, outputs, reconditions and effects
IPSec	Internet Protocol Security
IPTV	Internet Protocol TeleVision
ISP	Internet Service Provider
IT	Information Technology
ITIL	Information Technology Infrastructure Library
IWT	Institute for the Promotion of Innovation by Science and Technology in Flanders

K

KBN	Knowledge-Based Networking

L

L-ADS	Language of A^3DS
LFM	Layered Federation Model
LFU	Least Frequently Used
LP	Linear Programming
LRM	Layered Relationship Model
LRU	Least Recently Used

M

MB	MegaByte
Mbps	Megabits per second
MCOP	Multi-Constrained Shortest Path
MIB	Management Information Base
MIN	Optimal Caching Strategy
ms	Milliseconds

N

NIST	National Institute of Standards and Technology
NP	Non-deterministic Polynomial time

O

OIL	Ontology Inference Layer
oNCS	ontology-based Nurse Call management System
OP	Orchestration Plane
OPEX	OPerational EXpenditures
OP-LFU	Optimal-Selection Predictive Least Frequently Used
OSGi	Open Services Gateway initiative framework
OSKMV	Orchestration, Service enablers, Knowledge, Management, Virtualization
OTT	Over-the-Top content
OWL	Web Ontology Language
OWL-S	OWL Web Services Ontology
OWLS-MX	OWL-S Matchmaker

P

P-LFU	Predictive Least Frequently Used
PP-LFU	Perfect Predictive Least Frequently Used
PRoF	Patient Room of the Future

Q

QoE	Quality of Experience

QoS	Quality of Service

R

RAM	Random Access Memory
RCP	Routing Control Platform
RDF	Resource Description Framework
RESERVOIR	REources and SERvices VirtualizatiOn wIthout baRriers
RFID	Radio-Frequency IDentification
RSS	Really Simple Syndication

S

SCB	Semantic Communications Bus
sec	seconds
SERVME	SERViceable Metacomputing Environment
SLA	Service Level Agreement
SLS	Service Level Specification
SNMP	Simple Network Management Protocol
SPARQL	SPARQL Protocol and Resource Description Framework Query Language
SQL	Structured Query Language
SSL	Secure Socket Layer
SVC	Scalable Video Coding
SWRL	Semantic Web Rule Language

T

TAMCRA	Tunable Accuracy Multi-Constraints Routing Algorithm
TBox	Terminological Component
TCBPMS	Trusted Community-Based Policy Management System
TLS	Transport Layer Security
TM	Traceability Map
TSTV	Time-Shifted TeleVision

U

UDDI	Universal Description Discovery and Integration

V

VDM	Vienna Development Methodology
VM	Virtual Machine
VoD	Video on Demand
VoIP	Voice-over-IP
VS	Virtual Space

W

WS-Agreement	Web Services Agreement Specification
WSLA	Web Service Level Agreement
WWW	World Wide Web

X

XML	eXtensible Markup Language

List of Symbols

Chapter 3

$\oplus_q$	Aggregation operator of QoS parameter $q \in \mathcal{Q}$
$\prec_q$	Satisfaction operator of $q \in \mathcal{Q}$
$\mathcal{A}$	Set of access ISPs
$a_r \in \mathcal{A}$	Source ISP of request $r \in \mathcal{R}_a$
av	Access ISP vicinity
av_i	Probability that any access ISP is exactly i hops away from the first placed access ISP
$\mathcal{B}_p$	Set of bit rates offered by $p \in \mathcal{P}$
$b_o \in \mathcal{B}_{p_o}$	Bit rate of content stored in content cache $o \in \mathcal{O}$
$b_r \in \mathcal{B}_{p_r}$	Bit rate associated with request $r \in \mathcal{R}$
$\mathcal{C}$	Set of QoS classes
$\mathcal{C}_t \subseteq \mathcal{C}$	Set of QoS classes offered by $t \in \mathcal{T}$
$c_{t,r} \in \mathcal{C}_t$	QoS class reserved in $t \in \mathcal{T}$ for request $r \in \mathcal{R}$
χ_p	Number of items in the content catalogue of $p \in \mathcal{P}$
$d_o \in \mathcal{S} \cup \mathcal{A} \cup \mathcal{P}$	Domain where content cache $o \in \mathcal{O}$ is deployed
Δ	Total storage cost
$\Delta^{(i)}$	Total storage cost of the solution calculated during iteration i
$\Delta(o)$	Storage cost of the content cache $o \in \mathcal{O}$
$\Delta(\widehat{\Pi})$	Total storage cost of the shared path set $\widehat{\Pi}$
δ_p	Average duration of content offered by $p \in \mathcal{P}$
$\gamma_{c,q}$	QoS guarantee of QoS parameter $q \in \mathcal{Q}$ for QoS class $c \in \mathcal{C}$
$\gamma_{r,q}$	QoS constraint of QoS parameter $q \in \mathcal{Q}$ for request $r \in \mathcal{R}$
$\mathcal{I}$	Set of transit and access ISPs
$\mathcal{I}_i \subseteq \mathcal{I}$	Neighbour set of $i \in \mathcal{I}$

λ_s	Storage cost associated with domain $s \in \mathcal{S}$
$\mathcal{O}$	Set of content caches
$o_r^+ \in \Pi_r$	Successor of $o \in \mathcal{O}$ along the end-to-end path Π_r of $r \in \mathcal{R}$
$o_r^- \in \Pi_r$	Predecessor of $o \in \mathcal{O}$ along the end-to-end path Π_r of $r \in \mathcal{R}$
$\mathcal{P}$	Set of content providers
$p_o \in \mathcal{P}$	Content provider associated with content cache $o \in \mathcal{O}$
$p_r \in \mathcal{P}$	Target content provider of request $r \in \mathcal{R}_p$
$p_{\widehat{\Pi}}$	Boolean decision variable that equals 1 if $\widehat{\Pi}$ is selected and 0 otherwise
$\Phi_{p,b}(\cdot)$	Cumulative popularity distribution of $p \in \mathcal{P}$ for $b \in \mathcal{B}_p$
φ_s	Processing cost on $s \in \mathcal{S}$
$\Pi_r \subseteq O$	End-to-end path associated with $r \in \mathcal{R}$
$\Pi_r^{(i)} \subseteq O$	End-to-end path of $r \in \mathcal{R}$ selected during iteration i
$\widehat{\Pi}_o$	Shared path set of content cache $o \in \mathcal{O}$
$\{\widehat{\Pi}\}^{\text{cnd}}$	Candidate set of shared path sets for the current iteration
$\{\widehat{\Pi}\}^{(i)}$	Set of shared path sets of selected during iteration i
$\pi_{o,r}$	Core Internet path from $d_o \in \mathcal{P} \cup \mathcal{S}$ to $a_r \in \mathcal{A}$
Ψ	Total processing cost
$\Psi^{(i)}$	Total processing cost of the solution calculated during iteration i
$\Psi(\widehat{\Pi})$	Total processing cost of the shared path set $\widehat{\Pi}$
$\mathcal{Q}$	Set of QoS parameters
$\mathcal{R}$	Set of delivery requests
$\mathcal{R}_a \subseteq \mathcal{R}$	Set of requests for $a \in \mathcal{A}$
$\mathcal{R}_o \subseteq \mathcal{R}$	Set of delivery requests for which content will be served from content cache $o \in \mathcal{O}$
$\mathcal{R}_p \subseteq \mathcal{R}$	Set of requests for $p \in \mathcal{P}$
ρ_r	Expected number of simultaneous delivered content items for $r \in \mathcal{R}$
$\mathcal{S}$	Set of storage sites
sc	Storage cost
sv	Storage site vicinity
sv_i	Probability that any storage site is exactly i hops away from its access ISP
σ_o	Cache size of content cache $o \in \mathcal{O}$
σ_o^{aggr}	Upper bound on the aggregated cache size of $o \in \mathcal{O}$

$\mathcal{T}$	Set of transit ISPs
$t_e \subseteq \mathcal{T}$	Gateway of $e \in \mathcal{P} \cup \mathcal{S}$
Θ	Total transmission cost
$\Theta^{(i)}$	Total transmission cost of the solution calculated during iteration i
$\Theta(\widehat{\Pi})$	Total transmission cost of the shared path set $\widehat{\Pi}$
θ_c	Reservation cost associated with QoS class $c \in \mathcal{C}$

Chapter 4

$\mathcal{D}$	Set of popularity distribution models
D_c^{opt}	Optimal popularity model for content item c
P_D	Set of parameters associated with popularity model $D \in \mathcal{D}$
$P_{D,c}^{\text{opt}}$	Optimal parameter values of $D \in \mathcal{D}$ for content item c
R_c	Cumulative request pattern of content item c
θ	Time granularity of the popularity prediction algorithm
W	Prediction window

Chapter 5

B	Size (in bytes) of a single context exchange between AEs in flat architectures
B_l	Size (in bytes) of a single context exchange between an AE at layer l and its parent in hierarchical architectures
C	Maximum number of AEs per cluster in the hierarchical architecture
I	Interval (in seconds) between subsequent context exchanges among AEs
L	Number of layers in the hierarchical architecture
N	Number of neighbours per AE in the flat architecture
$\mathcal{O}_{\text{flat}}$	Total amount of generated context (in bytes) per second in flat architectures
$\mathcal{O}_{\text{hier}}$	Total amount of generated context (in bytes) per second in hierarchical architectures
R	Number of managed resources in the network

$\mathcal{R}_{\text{flat}}$	Maximum amount of context (in bytes) per second that an AE needs to process in flat architectures
$\mathcal{R}_{\text{hier}}$	Maximum amount of context (in bytes) per second that an AE needs to process in hierarchical architectures
S	Average number of executes services per server in the network
T	Number of resource types in the network
$\mathcal{V}_{\text{flat}}$	The number of resources that an AE receives context from in flat architectures
$\mathcal{V}_{\text{hier}}$	The number of resources that an AE receives context from in hierarchical architectures

Chapter 6

$C\,(x)$	SWRL class atom (x is an instance of C)
fp	Number of payloads per filter rule
I_o	Set of inputs in the offered service description
I_r	Set of inputs in the requested service description
mp	Percentage of matching service descriptions
O_o	Set of outputs in the offered service description
O_r	Set of outputs in the requested service description
$P\,(x, y)$	SWRL property atom (x is related to y via P)

Chapter 7

C_i	Number of child AEs governed by AEs at layer i of the hierarchy
M_i	Maximum number of child AEs governed by a single AE at layer i
n	Number of layers in the AE hierarchy
N_i	The amount of context information received by every AE at layer i during normal operations
O_i	Additional context generated at layer i when a single child cluster becomes overloaded
P	Percentage of overloaded servers
R	Number of server resource types that are monitored
S	Number of servers in the database
T_i	Number of AEs at layer i of the hierarchy

V	Average number of virtual machines per server

Appendix B

A	Set of all applications
$C_{\min}$	Number of child nodes causing under-utilization
$C_{\max}$	Number of child nodes causing over-utilization
C_c^r	Amount of currently allocated resources of type r in cluster c
D_a^r	Demand of application a for resource r
$E_{r,c}^{\text{low}}$	Estimated lower bound on available resources of type r in server cluster c
$E_{r,c}^{\max}$	Estimated upper bound on available resources of type r in server cluster c
Γ	Resources considered by the system.
Γ_s	Resources for which the demand is strict
Γ_l	Resources for which the demand is loose
M	Placement matrix
$M_{s,a}^r$	Amount of resource r allocated to server s for application a
M_c	Placement matrix of server cluster c
M_c'	Previous placement matrix of server cluster c
Ra	Resource availability of the various servers
Ra_s^r	Available amount of resource r on server s
$\hat{Ra}_c^r$	Aggregated available amount of resource r on server cluster c
Rd	Resource demand of the applications
$\hat{Rd}$	Aggregated resource demand of the applications
Rd_a^r	Resource demand of application a for resource r
Rd_l	Resource demands for loose resource types.
Rd_s	Resource demands for strict resource types.
S	Set of all servers
σ_a	Resource share of application a
U_c^r	Estimated amount of currently available resources of type r in cluster c
Υ	Set of usable servers

Samenvatting
– Summary in Dutch –

Doorheen de jaren is het internet op verschillende manieren geëvolueerd. Ten eerste is het uitgegroeid van een kleine groep met elkaar verbonden netwerken tot een wereldwijd communicatieplatform bestaande uit meer dan veertigduizend deelnetwerken. Ten tweede hebben de originele statische diensten (bv. e-mail en het wereldwijde web) plaats geruimd voor moderne multimediadiensten met strenge kwaliteitseisen (bv. IP-telefonie en video op verzoek). De kwaliteitseisen van multimediadiensten worden gewoonlijk gespecificeerd in functie van beperkingen op netwerkparameters (bv. maximale vertraging of minimale bandbreedte). Deze factoren hebben sterk bijgedragen tot de steeds groter wordende complexiteit en kost om het internet en de erop aangeboden diensten te beheren. Om deze problemen aan te pakken werden in laatste jaren verschillende vernieuwende aanpakken voorgesteld voor het beheren van grootschalige communicatienetwerken. Zo werd het *netwerkfederatie* paradigma naar voor geschoven om te kunnen voldoen aan de strenge kwaliteitseisen van multimediadiensten. Autonoom netwerkbeheer werd dan weer aangereikt als oplossing voor de steeds stijgende beheerscomplexiteit van communicatienetwerken.

Een netwerkfederatie wordt gedefinieerd als een overeenkomst tussen verschillende organisaties die de daarbij horende communicatienetwerken toelaat om systeembronnen op een gecontroleerde manier met elkaar te delen. Systeembronnen worden beschouwd in de brede zin van het woord, reikend van het reserveren van paden doorheen het netwerk met gegarandeerde bandbreedte tot het aanpassen van de configuratie van specifieke toestellen. Elke deelnemende organisatie behoudt de volledige controle over zijn netwerken. De federatie geeft de partners enkel het recht om bepaalde systeembronnen te gebruiken binnen de grenzen van de overeenkomst. Indien het internet federaties zou ondersteunen, zou dit aanbieders van multimediadiensten de mogelijkheid bieden om paden met kwaliteitsgaranties te reserveren doorheen het netwerk tot bij de gebruikers van hun diensten.

Autonoom netwerkbeheer heeft als doel de beheerscomplexiteit van grootschalige communicatienetwerken te verminderen voor de menselijke beheerder. Dit wordt bereikt door het netwerk te voorzien van zelfbeherende capaciteiten, waardoor het netwerk de mogelijkheid heeft om zichzelf te beheren en configureren binnen de grenzen die door de menselijke beheerder worden opgelegd. Autonoom netwerkbeheer verlicht dus de werkdruk op de beheerder en laat hem of haar toe zich te concentreren op belangrijke beheersbeslissingen.

Dit proefschrift bestaat uit twee belangrijke bijdragen. Vooreerst wordt een vernieuwende methodologie voor het opzetten van netwerkfederaties voorgesteld. Het doel hiervan is om te voldoen aan de strenge kwaliteitseisen van multimediadiensten die worden aangeboden over het internet. Daarnaast worden een aantal architecturale componenten voor de schaalbare samenwerking tussen autonome beheerscomponenten geïntroduceerd.

Het bieden van kwaliteitsgaranties op het internet zal pas mogelijk worden indien de ertoe behorende netwerkdomeinen samenwerken. Dit proefschrift stelt een raamwerk voor om automatisch federaties op te zetten en overeenkomsten te bekomen die deze samenwerking toelaten. Concreet zal het raamwerk de aanbieder van multimediabestanden toelaten om een overeenkomst te sluiten in verband met reservatie van paden met kwaliteitsgaranties naar de gebruikers van de diensten met de deelnetwerken waaruit het internet bestaat. De gebruikers zijn aanbieders van internettoegang, die de multimediabestanden op hun beurt aanbieden aan hun eigen klanten over een beheerd IP netwerk. Daarbovenop kunnen aanbieders van *cloud computing* oplossingen worden toegevoegd aan de netwerkfederaties. Dit geeft de bestandsaanbieders de mogelijkheid om dynamisch opslagruimte te reserveren en bijgevolg *bestandscaches* te ontplooien. Deze *caches* reduceren de hoeveelheid gebruikte bandbreedte en dus de totale kost om de bestanden af te leveren. Omdat de *caches* zich midden in het netwerk bevinden, kunnen ze ook gedeeld worden over verschillende gebruikers. Dit verhoogt hun efficiëntie, zonder de totale opslagkost te verhogen. Er wordt ook een algoritme voorgesteld dat de optimale set van netwerkdomeinen selecteert voor deelname aan de federatie. Daarbovenop berekent dit algoritme de kwaliteitsgaranties en hoeveelheid opslagruimte, die in elk van de gekozen domeinen moet worden gereserveerd, zodat wordt voldaan aan de gevraagde kwaliteitsgaranties en de totale kost wordt geminimaliseerd. Het algoritme werd kwantitatief geëvalueerd om de voordelen van het raamwerk te onderzoeken. De resultaten tonen aan dat de voorgestelde aanpak in staat is om de totale kosten sterk te reduceren ten opzichte van traditionele aanpakken. De grootte van deze reductie hangt wel af van de relatieve opslagkost en afstand van de federatiepartners tot elkaar. Het delen van een *cache* over verschillende gebruikers blijkt daarbovenop de kost nog verder te reduceren voor gebruikers die zich in elkaars nabijheid bevinden.

Het voorgestelde raamwerk voor het opzetten van netwerkfederaties op het internet gebruikt dynamisch ontplooide *caches* om het afleveringsproces van multimediadiensten te optimaliseren. Om de efficiëntie van deze *caches* verder te optimaliseren, wordt er een vernieuwende *cache*-vervangingsstrategie voorgesteld. Deze strategie beslist welke bestanden in het *cache* bewaard zullen worden. De theoretisch optimale strategie zal ervoor kiezen om de bestanden die het populairst zijn in de nabije toekomst te bewaren. Traditionele strategieën gebruiken de populariteit van het verleden als een directe indicator voor de toekomst. In dit proefschrift wordt daarentegen een strategie voorgesteld die, met behulp van machinaal leren, de toekomstige populariteit voorspelt. Concreet wordt het historische aanvraagpatroon van een bestand benaderd door een set van populariteitsmodellen. Met behulp van simulatieresultaten wordt het nut van de voorspellende aanpak on-

derzocht. De resultaten tonen aan dat de voorspellende strategie in theorie een sterke prestatieverbetering kan bekomen.

Een autonoom netwerkbeheerssysteem dat grootschalige gefedereerde netwerken op een efficiënte manier wil beheren, heeft nood aan een groot aantal gedistribueerde zelf-beherende autonome componenten, ook Autonome Elementen (AEs) genoemd. Deze AEs moeten in staat zijn om te communiceren en samen te werken om de globale beheersdoelstellingen van het netwerk te kunnen bereiken. Dit proefschrift stelt een hiërarchische architectuur voor die de interactie tussen AEs bevordert. Zoals de naam doet vermoeden, structureert de architectuur de AEs in een hiërarchie. Deze vorm sluit nauw aan bij hun beheersverantwoordelijkheden. Om de schaalbaarheid te verzekeren wordt vergaarde informatie samengevat en geselecteerd, alvorens ze doorheen de hiërarchie wordt verzonden. De architectuur ondersteunt dus zowel het uitvoeren van gedetailleerde configuraties (onderaan) als optimalisaties op grote schaal (bovenaan) zonder in te boeten op vlak van schaalbaarheid. Er werd ook een analytisch model ontworpen en geanalyseerd om deze beweringen te staven. De resultaten toonden aan dat de architectuur veel beter is uitgerust om om te gaan met de stijgende beheerscomplexiteit van moderne communicatienetwerken dan traditionele gecentraliseerde en gedistribueerde aanpakken.

De voorgestelde architectuur is in staat om schaalbaarheid te behouden door op een intelligente manier informatie te aggregeren en selecteren. Om dit te automatiseren werd de Semantische Communicatie Bus (SCB) ontwikkeld. De SCB laat AEs toe om semantische filterregels te definiëren, om hun interesse in bepaalde types informatie aan te geven. Door gebruik te maken van semantiek kan informatie worden geselecteerd op basis van de betekenis, in plaats van syntactische kenmerken. Daarbovenop zorgt de toegevoegde semantiek ervoor dat AEs informatie op een eenduidige manier kunnen interpreteren. Door de semantische modellen te verdelen overheen verschillende netwerkdomeinen kunnen AEs, die zich in verschillende domeinen bevinden, daarbovenop met elkaar communiceren. Om te kunnen samenwerken moeten de AEs ook in staat zijn om gespecialiseerde beheersfuncties, die worden aangeboden door andere AEs, te vinden en uit te voeren. Om dit te bewerkstelligen voorziet de SCB een algoritme dat met behulp van semantische redeneertechnieken beheersdoelstellingen koppelt aan beheersfunctionaliteit.

Om het nut van de voorgestelde beheersarchitectuur aan te tonen, wordt deze tenslotte toegepast op het beheer van een grootschalige gefedereerde *cloud*. De hiërarchische AE architectuur wordt gecombineerd met een hiërarchische versie van de SCB en omgevormd tot een raamwerk voor het beheren van *clouds*. Een prototype implementatie wordt gebruikt om de theoretisch aangetoonde voordelen in de praktijk te bewijzen. De resultaten tonen aan dat de schaalbaarheid van het raamwerk kan worden gewaarborgd door het aantal lagen in de hiërarchie proportioneel te laten toenemen met de groei van de *cloud* infrastructuur.

Summary

Since its inception, the Internet has evolved significantly along different dimensions. First, it has grown from a small group of inter-connected networks to a global communications substrate consisting of over forty thousand Autonomous Systems (AS), each made up of potentially huge amounts of internal hard- and software resources. Second, the Internet's original static services (e.g., email, web browsing) have been superseded by modern multimedia services with stringent quality requirements (e.g., Voice-over-IP, Video on Demand). The quality requirements of multimedia services are commonly referred to as Quality of Service (QoS), which is expressed in terms of constraints on network parameters (e.g., maximum delay, minimum throughput). These factors contribute to the ever-increasing complexity and cost to manage and configure the Internet and its services. Novel network management paradigms have been suggested in response to these problems. Specifically, *network federations* have been advanced as a method for satisfying the end-to-end QoS requirements of multimedia services, while the *autonomic network management* paradigm has been proposed to encompass the ever-increasing size and complexity of communications networks.

Network federations are defined as persistent cross-organizational agreements that enable the cooperating networks to share capabilities in a controlled way. These capabilities might range from the reservation of bandwidth-guaranteed paths to control over specific device configurations. Each organisation retains full control over the management and configuration of its own network. They merely give their federation partners (possibly constrained) rights to use the capabilities specified in the agreement. Federations would allow providers of multimedia services to automatically negotiate agreements with the core Internet domains on the end-to-end paths towards their consumers, enabling them to reserve QoS-guaranteed paths through the Internet core.

The autonomic network management paradigm aims to reduce the management complexity of large-scale communications networks for the human network operator. This is achieved by introducing self-governing capabilities into the network, which enable it to manage itself within the boundaries of the management policies specified by the operators. The network thus assumes part of human operator's management responsibilities. This reduces the operator's workload and allows him or her to focus on high-level management tasks. In contrast, the network itself performs low-level management and configuration tasks.

The contributions of this dissertation are twofold. First, a novel methodology for the federated delivery of Internet-based multimedia services with stringent

QoS requirements is presented. Second, several architectural components are introduced to facilitate the scalable collaboration between distributed self-governing management components in autonomic network management frameworks.

The end-to-end guarantee of QoS over the Internet can only be achieved through cooperation of the network domains on the end-to-end route through the Internet between the multimedia service provider and its consumers. This dissertation presents a framework to automatically negotiate federation agreements, which facilitate such cooperation. Specifically, the framework sets up federations between the multimedia content provider and a set of intermediary core Internet domains. This facilitates the reservation of QoS-guaranteed end-to-end paths towards the consumers. The consumers are Internet access providers, which deliver the content over a managed IP network to their own customers, a set of end-users. The federation can additionally incorporate cloud providers. This allows the content provider to dynamically reserve storage resources, and deploy content caches. These caches reduce the total bandwidth consumption and thus decrease delivery costs. Additionally, as these caches are deployed inside the network, they can be shared among several consumers. Cache sharing increases the cache's efficiency, without increasing the total storage cost. An algorithm is presented that selects the optimal set of network domains to include into the federation, from the set of all possible network domains. Additionally, the algorithm calculates the QoS guarantees and storage resources that should be reserved in order to minimize the total delivery cost, while satisfying all QoS constraints. The algorithm is quantitatively evaluated, in order to explore the merits of the framework. Results show that the presented approach is capable of significantly reducing delivery costs, compared to traditional end-to-end QoS negotiation mechanisms. The significance of the cost reduction does depend on the relative cost for storing a content item in the cloud and the distance between federation partners. Moreover, cache sharing was shown to further decrease the total delivery costs for consumers that are positioned near each other in the network.

The presented framework for setting up end-to-end network federations uses dynamically deployed content caches to optimize the delivery of multimedia services over the Internet. To further improve the effectiveness of caches, this dissertation presents a novel cache replacement strategy. Such a strategy defines a policy to decide what content to keep in the cache, as it can usually only host a small subset of all available content. Optimally, the strategy should retain the content that will be most popular in the near future. Traditional strategies use past popularity as a direct indicator for the future. In contrast, we propose a predictive cache replacement strategy, which uses machine learning to predict future popularity. Specifically, the content's historical request trace is approximated by a set of popularity models, using a curve fitting algorithm. Based on simulation results, the viability of the predictive approach is studied. It is shown that, in theory, a predictive cache replacement strategy can significantly outperform traditional strategies.

To effectively manage large-scale federated networks in an autonomic fashion, the network will need to contain huge amounts of self-governing management components, also called Autonomic Elements (AEs). These AEs will need

to communicate and cooperate in order to satisfy the network's global management goals. The trend towards inter-domain federations further emphasizes the importance of unambiguous communication between AEs. This dissertation presents a hierarchical architecture to facilitate the intra- and inter-domain interaction between AEs. The architecture structures AEs in a hierarchy, which maps directly to their management responsibilities. To maintain scalability, AEs summarize, filter and aggregate information about the underlying resources. The architecture thus supports both detailed configuration (at the bottom) as well as widespread management optimizations (at the top), without sacrificing scalability. An analytical model is presented and analysed to prove this claim. Results show that the hierarchical architecture is much better equipped to cope with the increasing management complexity of modern communications networks, compared to traditional centralized or flat distributed architectures.

The architecture's scalability stems from its capability to dynamically filter, aggregate and summarize information. To achieve the automatic and intelligent filtering of information, the Semantic Communications Bus (SCB) is proposed. It allows AEs to specify semantic filter rules to indicate their interests in specific types of information. The use of semantics supports filtering based on meaning, rather than syntactic characteristics. Additionally, semantics support the unambiguous interpretation and understanding of the exchanged information. Through a set of shared semantic information models, correct understanding can additionally be guaranteed across the bounds of network domains. In order to effectively collaborate, AEs also need to be able to execute specialized management functions offered by other AEs. A matchmaking algorithm is proposed and incorporated into the SCB to facilitate this. The algorithm uses semantic reasoning techniques to match AE management goals with specialized functionality offered by other AEs. The matchmaker also employs the SCB's shared semantic information models, further guaranteeing unambiguous understanding between AEs.

Finally, to prove the viability of the proposed hierarchical management architecture, it is applied to the management of large cloud computing data centers. A unified cloud management framework, combining the hierarchical AE architecture and a hierarchical version of the SCB, is presented. The previously made claims are substantiated using a prototype implementation of the framework. Results show that scalability can be maintained under an ever-increasing number of managed servers and services, by proportionally increasing the number of layers in the management hierarchy and intelligently filtering superfluous monitoring information.

1 Introduction

"Every generation needs a new revolution."

– Thomas Jefferson (1743–1826)

1.1 The Internet revolution

The origins of the Internet can be traced back to almost fifty years ago [1]. In 1966, the plans for the ARPANET were laid out. It was the first operational packet switched network and is widely considered to have been the basis of the current Internet. A few years later, in 1972, the *open architecture networking* idea was advanced. It allows networks – with diverse underlying technologies – to be connected to one another, and paved the way for the Internet as we know it. Open architecture networking relies on four critical rules:

- Each distinct network had to stand on its own, and no internal changes could be required of any such network before being connected to the Internet.
- Communications would be on a best-effort basis. If a packet did not make it to the final destination, it would quickly be retransmitted from the source.
- Black boxes (later called gateways and routers) would be used to connect the networks. No information would be retained by the gateways about individual flows of packets passing through them.
- There would be no global control at the operations level.

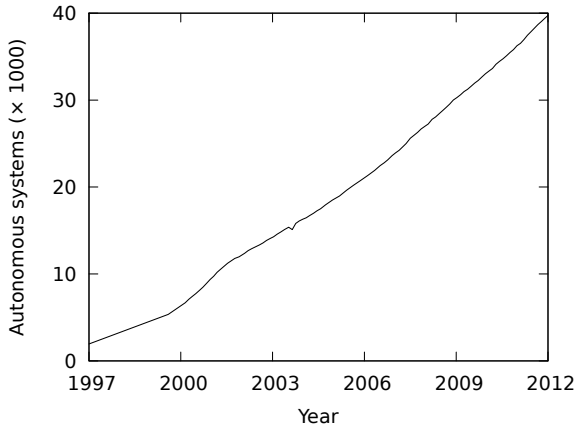


Figure 1.1: The growth of the Internet over the years in terms of the number of Autonomous Systems; In fifteen years time, the Internet has increased twentyfold in size (data obtained from [2])

Since its inception, the original ARPANET has grown into today's global Internet, consisting of thousands of inter-connected networks. Despite this immense increase in size, it still adheres to the original rules of open architecture networking. The networks that make up the Internet are grouped into Autonomous Systems (ASs). Every AS is independently managed by one or a few organizations and adheres to a clearly defined internal routing policy. The Internet's incredible growth is clearly illustrated by Figure 1.1, which depicts the total number of ASs over time [2]. In the beginning of 1997 there were less than 2000 ASs connected to the Internet. Since early 2012, this number has grown to over $40,000$. This evolution is mainly attributed to the Internet's increased penetration, as the number of users has grown from 70 million to over 2000 million during this timespan [3]. In turn, this has caused the Internet's core to become larger (i.e., comprise more ASs) and more complex.

Additionally, the Internet has continuously evolved towards richer and more demanding services. In the early days, it was dominated by email and Usenet traffic. In 1991, the Internet was revolutionized by the introduction of the World Wide Web (WWW). Although some of these early applications are still popular today, the next revolution has begun. A new generation of rich services has emerged, starting with the popularization of peer-to-peer file sharing applications in 1999. Subsequently, Skype introduced the general public to Voice-over-IP (VoIP) in 2003. Since then, video-based multimedia services have come into widespread use. In 2010, global Internet video traffic finally surpassed peer-to-peer traffic since it rose to the top ten years ago [4]. Figure 1.2 plots this evolution in Inter-

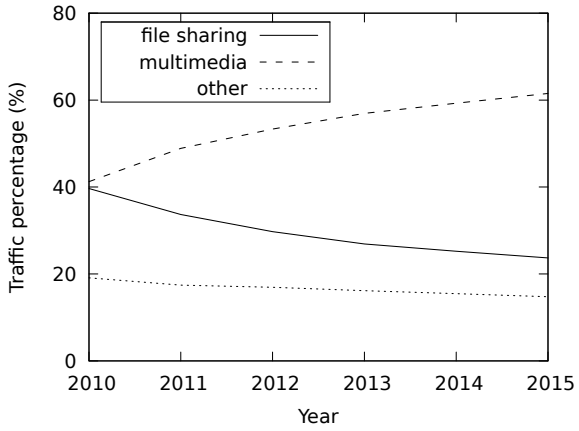


Figure 1.2: Categorization of global Internet traffic in the recent past and as expected in the near future; By 2015, multimedia services are expected to make up over 60% of all Internet traffic (data obtained from [4])

net traffic and presents a forecast for the near future. The file sharing category contains peer-to-peer traffic as well as traffic from web-based file-sharing systems. The multimedia category consists of online gaming, VoIP, video conferencing, Internet television (e.g., Hulu) and video sharing services (e.g., YouTube). Finally, the "other" category contains all other traffic, including but not limited to web, email and instant messaging.

The recent proliferation of novel services across the Internet has been greatly aided by the rise of cloud computing. First thought by many to be a marketing hype, it has since proven its merits. The cloud computing paradigm is based on the concept of utility computing, offering computing resources as a service. This allows service providers to offer their applications across the Internet, without needing to invest in expensive server and network infrastructure resources. Additionally, to meet changes in user demand it supports on-demand resource reservations, reducing the need for over-provisioning.

The Internet has evolved significantly over the years. First, it has grown from a small group of inter-connected networks to a global communications substrate consisting of tens of thousands of Autonomous Systems. Second, the original small selection of offered applications has expanded into a wide variety of rich multimedia services. These evolutions have significantly contributed to the Internet's increased operational and management complexity. Nevertheless, its ground rules and underlying architectural and technical concepts remain largely unchanged.

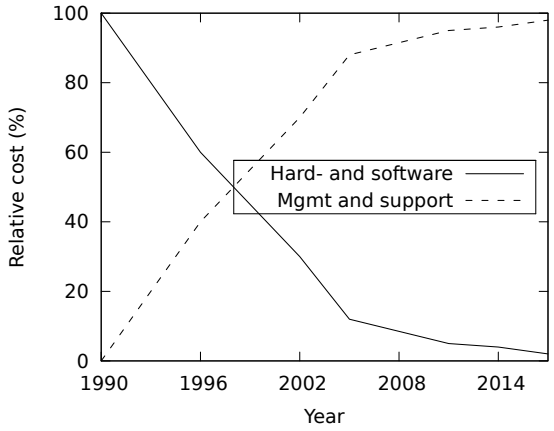


Figure 1.3: Evolution of the relative cost of hard- and software compared to the management thereof; The management cost of computing and communications systems has increased drastically over the years, due to their increased complexity and size (data obtained from [5])

1.2 Problem statement

Despite the Internet's metamorphosis in size and service offerings, there has been a long-term stagnation in its operations and management. The four critical rules of open architecture networking are still in place. The networks that make up the global Internet still stand on their own, with little to no interaction or collaboration among them. Moreover, transmission of data is still packet- and best-effort-based. Nevertheless, the requirements of the Internet's services have significantly changed over the years. Traditionally, the Internet was used mostly for email and the WWW, which can easily handle network hiccups (e.g., increased delay and jitter or reduced bandwidth). In contrast, the quality of modern multimedia services quickly deteriorates under suboptimal network conditions.

The long-term stagnation of the Internet's underlying principles has been the source of several recent problems. The ever-increasing size and complexity of the Internet has caused a shift in the costs associated with maintaining it. Where originally the majority of the costs were associated with the hard- and software infrastructure itself, this has shifted towards the cost of managing and supporting it. Figure 1.3 depicts this evolution and plots the relative hard- and software cost against the management and support cost. The figure shows that the hard- and software cost is expected to become negligible compared to the management cost in the near future [5]. Moreover, due to the Internet's best-effort nature and the lack of cooperation among Autonomous Systems, the stringent end-to-end requirements of modern multimedia services cannot be guaranteed. This has, among

others, been recognised by Lawrence G. Roberts, one of the founding fathers of the ARPANET. Roberts states that the current Internet is incapable of offering the guarantees required by multimedia services [6]. He claims that the gross over-provisioning of the Internet is the only reason that such services can be successfully served today. Obviously, over-provisioning significantly increases costs and does not solve the underlying problems.

To address these concerns, there is a need for novel management approaches. Specifically, the *autonomic network management* paradigm has been proposed to encompass the ever-increasing size and complexity of communications networks, while *network federations* have been advanced as a method for satisfying the end-to-end quality requirements of novel multimedia services. The autonomic networking paradigm is an extension of autonomic computing, proposed by IBM in 2001 [7]. It aims to simplify the ever-increasing management complexity of communications networks by giving network entities the capability to self-govern their behaviour within the constraints of the business goals that the network as a whole seeks to achieve [8]. This is expected to significantly reduce the management complexity of large-scale networked systems for human network operators. They will be able to focus their efforts on high-level decision making, while the network itself governs its low-level configuration. A federation of networks is defined as a persistent cross-organizational agreement that enables the cooperating networks to share capabilities in a controlled way [9]. This implies that these networks have sovereign decision-making power and there is no single governing authority of the federation. Additionally, the agreement should be persistent (but not permanent), which means that it should outlive individual transactions or interactions. Finally, the term capability is used in the widest possible sense. It could range from the reservation of network resources, the exchange of information and even the changing of individual device configurations. By setting up federations, Autonomous Systems can collaborate in order to guarantee the end-to-end quality requirements of multimedia services across the Internet.

In order to alleviate the aforementioned problems, the Future Internet should incorporate characteristics of both autonomic and federated network management. However, applying these principles in practice is challenging and many open issues remain to be solved. This dissertation targets several of them:

- The Future Internet is expected to consist of large-scale network domains, possibly containing many thousands of hard- and software components. Autonomously managing these gargantuan and complex networked systems requires many distributed self-governing management components. In order to achieve their goals, these autonomic components need mechanisms to efficiently communicate and collaborate with one another.

- The network will have to dynamically adapt to changing end-to-end require-

ments of novel multimedia services. To encompass this, network federations should be dynamically and automatically negotiated. This in itself leads to several other issues:

  - The intra-domain communication and collaboration mechanisms should be extended, supporting the exchange of information and execution of capabilities across network domain boundaries. Only then dynamic and automated federations will become possible.
  - The Internet consists of an ever-growing amount of network domains. When setting up a federation, the minimal relevant subset of network domains that should participate needs to be identified. This subset of domains should together be capable of satisfying the end-to-end requirements, while minimizing the associated costs.
  - There is a need for protocols to unambiguously negotiate the shared capabilities and their associated costs, in order to come to mutually beneficial agreements between domains.

- The amount of multimedia traffic on the Internet has steadily increased over the years, and is predicted to increase even faster in the future [4]. To be able to keep up with the ever-growing bandwidth requirements of this type of content, mechanisms need to be deployed throughout the network that significantly reduce end-to-end resource demands (e.g., content caches).

1.3 Definitions & terminology

This section provides a definition of the most important concepts used throughout this dissertation:

- **Autonomic Element**: An autonomic element (AE) is a self-governing network management component. It is responsible for the management and configuration of one or more network resources (e.g., hard- and software components) and/or other AEs. It achieves this through the use of autonomic network management principles, such as autonomic control loops and self-managing algorithms.

- **Autonomic Network Management**: Throughout this dissertation, the definition of autonomic network management asserted by Jennings *et al.* [8] is used: "*Autonomic network management aims to simplify network management processes by automating and distributing the decision making processes involved in optimizing network operation. Its goal is to enable expensive human attention to focus more on business logic and less on low-level device configuration processes.*" This implies that it does not aim to eliminate human intervention, but instead to assist it.

- **Caching**: Caching is defined as storing data, so that future requests for that data can be served more easily. This process is usually transparent for the end-user. Although the technique has been traditionally applied to many kinds of data (e.g., computer instructions, web pages), here it refers to the caching of multimedia content. Storing a subset of popular content in caches closer to the end-users, significantly reduces the amount of data that needs to be transported over the Internet. This has several advantages, including reducing the delay and consumed bandwidth.

- **Cloud Computing**: In its recent rise to fame, several definitions have been proposed for the cloud computing paradigm. In an attempt to converge these different views, the National Institute of Standards and Technology (NIST) recently published a definition. This highly cited publication can be considered the de-facto standard cloud computing vision [10]: "*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*"

- **Network Federation**: The definition of network federations, as formulated by Jennings *et al.* [9] is used throughout this dissertation: "*A persistent organizational agreement that enables multiple autonomous entities to share capabilities in a controlled way.*" This definition has several implications. First, a federation brings together autonomous entities (organizations or individuals) endowed with sovereign decision-making power over the resources they own or control. Hence, there is no single authority for the federation. Second, the federation must exist by the virtue of the agreement of its members. Third, it exists to enable the controlled sharing of capabilities between its members. A capability ranges from the availability of a communication channel to the ability to perform device configuration changes. Controlled sharing refers to the fact that federation members are granted (possibly constrained) access to capabilities they would not otherwise possess. Finally, the federation should be persistent, which does not imply it should be permanent or even last any minimum period of time. Instead, it should outlive individual transactions or interactions between federation members.

- **Publish/Subscribe**: A messaging pattern that only loosely couples the senders of messages (i.e., publishers) with the receivers thereof (i.e., subscribers). Specifically, the publisher does not directly specify the receivers of a message. Instead, subscribers express their interest in specific types of messages

through the definition of subscription or filter rules. The filtering is then performed based on these rules, which could refer to the topic of the message or its content (or a combination of both).

- **Quality of Service**: Quality of Service or QoS is the characterisation of transport requirements of services. In the context of computer networks, it refers to the ability to guarantee a certain level of performance to a data flow. It is usually specified as a constraint on one or more network parameters (e.g., maximum delay, minimum availability). As previously stated, QoS guarantees are especially important for real-time multimedia services, such as streaming video and VoIP.

- **Semantics**: In computer science, semantics refers to the meaning of information, rather than its form (syntax). Concretely, Euzenat and Shvaiko [11] defined semantics as "*providing the rules for interpreting the syntax which do not provide the meaning directly but constrain the possible interpretations of what is declared.*" Semantics thus allow computing systems to interpret information and expressions.

1.4 Research contributions

This thesis aims to incorporate and combine aspects from the autonomic and federated network management paradigms in the management of the Future Internet and its services. The ultimate goal is to improve the Internet's ability to cope with its ever-increasing size and complexity, as well as the increasingly stringent QoS requirements of novel multimedia services. This translates into the following main contributions[1]:

1. A framework for negotiating, configuring and maintaining end-to-end federations of network domains for the delivery of Internet-based multimedia services with stringent QoS requirements.

    - A methodology to negotiate the terms of a federation between the stakeholders involved in the delivery of multimedia services. More specifically, it facilitates the negotiation of the agreement terms (i.e., costs, QoS classes and reserved resources) between the content provider, its customers and intermediary network domains along the delivery route. Additionally, cloud-based storage sites can be incorporated in

[1]The research presented in this dissertation was conducted in collaboration with Steven Latré. He has recently finished his PhD on autonomic Quality of Experience management of multimedia services. His work focussed on enablers in the network for the optimization of Quality of Experience, but not on caching, which is considered here. Additionally, he proposed an algorithm for the automatic generation and adaptation of filter rules, which complements the context dissemination framework introduced in this dissertation.

the federation, allowing caches to be dynamically deployed in order to reduce bandwidth requirements and thus delivery costs.

- An algorithm to determine the minimal set of stakeholders and reserved capabilities (i.e., QoS classes and storage resources) required to satisfy the QoS requested by customers, while minimizing the total delivery costs for the content provider.
- A thorough evaluation of the framework's quantitative advantages. Specifically, the proposed federated approach is compared to more traditional methods for the QoS-aware delivery of multimedia content. Additionally, the merits of dynamically deployed caches are studied under a variety of conditions.

2. A novel cache replacement strategy for multimedia content. Traditional cache replacement strategies directly apply historical information to decide what content to cache. In contrast, the presented strategy uses predicted information about future popularity instead.

   - A generic prediction algorithm that fits a set of popularity models to the content's historical request trace and uses the best fit to perform the actual prediction.
   - An evaluation of the theoretical performance limits of predictive cache replacement. Simulation results are used to determine the effectiveness of predictive caching compared to traditional methods under a variety of assumptions. First, the performance gain achieved assuming perfect predictions is evaluated. Second, the maximum gain when using the curve fitting algorithm as a predictor is studied.

3. A hierarchical network management architecture to support scalable communication and collaboration between AEs, both within and across network domains.

   - An analytical evaluation of the merits of hierarchically structured AEs. The analytical model is used to prove the effectiveness and scalability of hierarchical architectures for managing large scale networks, compared to the traditionally used flat architectures.
   - A substrate to facilitate communication between AEs, called the Semantic Communications Bus (SCB). Self-governing management components require detailed information about their environment in order to be able to effectively configure and manage it. The SCB is responsible for the dissemination of this information. It is augmented with semantics, allowing the SCB to intelligently filter relevant information based on meaning, rather than syntactic characteristics.

- An algorithm for semantic matchmaking and discovery of management services. In addition to exchanging information, AEs interact by instantiating specialized management functionality offered by other AEs. The matchmaking algorithm allows them to specify required functionality using semantically annotated concepts. It then matches these specifications with semantically described management functions. It supports requirements on inputs and outputs, as well as preconditions and effects related to the managed environment.
- A quantitative evaluation to explore the performance impact of semantic reasoning on information dissemination and matchmaking. Several semantic techniques, with varying expressiveness and reasoning capabilities, were evaluated and compared.
- An autonomic, hierarchical framework to manage large scale cloud data centers. Through the use of hierarchies it achieves significantly increased scalability, necessary for managing large-scale clouds. It additionally applies the SCB to a hierarchical scenario. This allows semantically annotated context to be intelligently aggregated, summarized and filtered as it propagates through the management hierarchy, further improving scalability.
- An analytical model and prototype implementation to characterize and evaluate the scalability and performance of the proposed hierarchical cloud management framework.

1.5 Outline of this dissertation

The chapters of this thesis consist of a selected number of publications, written in the context of this PhD. Together, they present a complete and consistent view on the performed work. They offer significant contributions to scientific research related to autonomic and federated network management of large-scale future networks, as outlined above. Chapter 2 provides an overview of state of the art research on federated network management. It explores the trend towards inter-domain collaborative approaches for managing the Future Internet and its services and identifies several challenges and open issues in this field.

Figure 1.4 positions the remaining chapters and appendices of this dissertation within the end-to-end view of the Internet. The figure depicts the network domains involved in our envisioned federated multimedia service delivery architecture. On top of the Internet's infrastructure and services is a management layer, consisting of a huge number of self-governing AEs. Through the SCB, AEs communicate and cooperate both within and across domain boundaries, in order to achieve QoS-guaranteed delivery of novel multimedia services over the Internet.

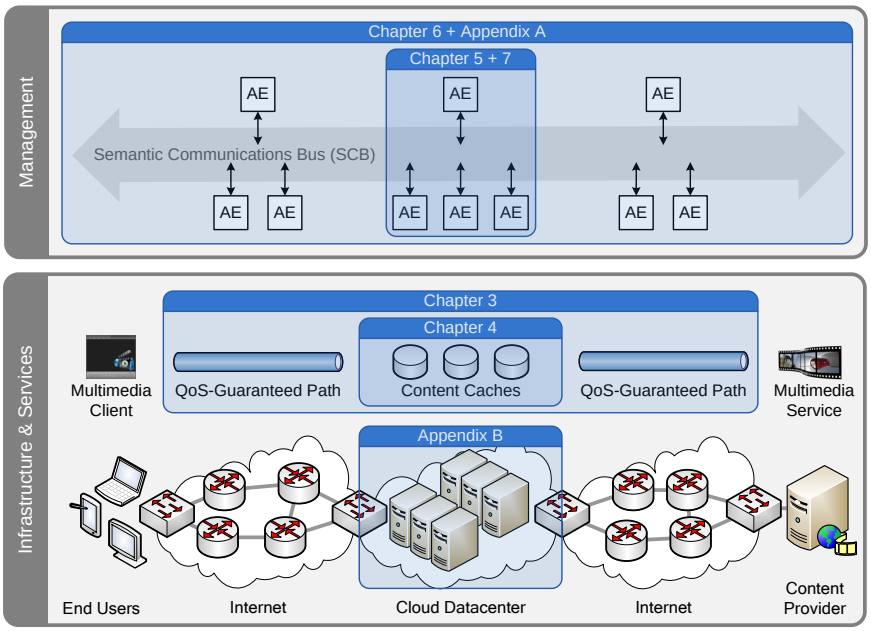


Figure 1.4: Schematic overview of the contributions of this dissertation; The focus of the different chapters and appendices is highlighted

Chapter 3 presents the multimedia service delivery framework, as described above in Contribution 1. The framework automatically negotiates and configures federations between the stakeholders involved in the end-to-end delivery of multimedia services. This allows the content provider to reserve QoS-guaranteed paths through the Internet core. Additionally, by setting up agreements with intermediary cloud providers, caches can be deployed closer to the end-users, reducing bandwidth and minimizing costs. An algorithm is presented that selects the optimal set of stakeholders to include in the federation, based on the location of customers as well as their requested QoS. The algorithm additionally determines the optimal set of capabilities (i.e., QoS classes, bandwidth and storage resources) to reserve within each of the identified stakeholder domains. Its goal is to minimize the content provider's costs, while satisfying customer QoS requirements. Chapter 4 further elaborates on the caching aspect of the architecture proposed in Chapter 3, which is outlined in Contribution 2. The chapter presents an algorithm to predict the future popularity of multimedia content, based on its historical request trace. It employs a curve fitting algorithm to approximate the request trace with a set of popularity models. Subsequently, this popularity prediction algorithm is combined with a novel predictive cache replacement strategy.

Chapters 5, 6 and 7 focus on the interaction between AEs in a federated and

autonomic network management architecture. Together, these chapters form Contribution 3, as outlined above. First, Chapter 5 introduces a hierarchical network management framework. The proposed framework structures self-governing AEs in a management hierarchy. This allows them to communicate and collaborate in a more effective and scalable manner than centralized or non-hierarchical distributed architectures would allow. In addition to the intra-domain hierarchical relationships, the chapter also discusses peer-based inter-domain interactions. Subsequently, Chapter 6 presents the SCB, a semantic communications substrate to facilitate the exchange of information and discovery of management functionality, both within and across network domains. Its semantics allow AEs to interpret the received information and intelligently filter it. Additionally, semantics further aid the correct interpretation and understanding of information exchanged across the bounds of network domains, making the approach ideally suited for federated management scenarios. Finally, Chapter 7 brings together the concepts introduced in Chapters 5 and 6, proposing a hierarchical AE-driven architecture to manage large-scale cloud provider data centers. The hierarchically structured AEs exchange semantically annotated monitoring information through the SCB. In order to guarantee scalability under a growing number of managed servers, the information is automatically aggregated and summarized as the SCB propagates it through the hierarchy.

Filter rules allow AEs to specify the type of information in which they are interested. The SCB presented in Chapter 6 uses these filter rules to match published information with the interests of AEs. Appendix A presents a complementary algorithm to automatically generate such filter rules. Through semantic reasoning, the algorithm dynamically adapts the generated filter rules based on the current state of the managed environment. By extending the SCB with the proposed filter rule generation algorithm, information can be more intelligently filtered, taking into account the dynamically changing interests of AEs. Appendix B focusses on the algorithmic implications of the hierarchical cloud management architecture presented in Chapter 7. Existing algorithms for resource allocation in cloud computing data centers are incompatible with hierarchically structured distributed AEs. The appendix fills this gap by proposing a methodology to transform existing centralized or distributed resource allocation algorithms into hierarchical versions. Appendix C (not depicted in the figure) demonstrates the wider applicability of the SCB and presents a context-aware healthcare platform, centered around the SCB. The platform manages a wide range of physical sensors, as well as several healthcare services. The huge amounts of data generated by the sensors is intelligently and dynamically aggregated and filtered by the SCB, before being sent to the interested services. This platform proves the portability of the SCB in other application domains.

1.6 Publications

The research results obtained during this PhD research have led to several publications in peer reviewed scientific journals and proceedings of both national and international conferences. A complete list is provided below.

1.6.1 A1: Publications in international journals indexed by the ISI Web of Science "Science Citation Index Expanded"

1. **Jeroen Famaey**, Tim Wauters, Filip De Turck, Bart Dhoedt, and Piet Demeester. *Network-Aware Service Placement and Selection Algorithms on Large-scale Overlay Networks.* Published in Computer Communications, Volume 34, Issue 15, Pages 1777–1787, September 2011. doi: 10.1016/j.comcom.2011.03.017.

2. **Jeroen Famaey**, Steven Latré, John Strassner, and Filip De Turck. *Semantic Context Dissemination and Service Matchmaking in Future Network Management.* Published in International Journal of Network Management (IJNM), September 2011. doi: 10.1002/nem.805.

3. **Jeroen Famaey**, Steven Latré, John Strassner, and Filip De Turck. *A Hierarchical Context Dissemination Framework for Managing Federated Clouds.* Published in Journal of Communications and Networks (JCN), Volume 13, Issue 6, Pages 567–582, December 2011. doi: 10.1109/JCN.2011.6157473

4. **Jeroen Famaey**, Steven Latré, Wim Van de Meerssche, Koen De Schepper, Bart De Vleeschauwer, Tim Wauters, and Filip De Turck. *Deadline-Aware Scheduling for Time-Constrained Multimedia Content Delivery.* Submitted to Computer Communications, April 2012,

5. **Jeroen Famaey**, Steven Latré, Tim Wauters, and Filip De Turck. *Federated Resource Management for End-to-End Multimedia Services.* Submitted to Journal of Network and Systems Management (JNSM), April 2012,

6. **Jeroen Famaey**, Tim Wauters, and Filip De Turck. *Towards a Predictive Cache Replacement Strategy for Multimedia Services.* Submitted to Journal of Network and Computer Applications (JNCA), April 2012.

7. **Jeroen Famaey**, Filip De Turck. *Federated Management of the Future Internet: Status and Challenges.* Submitted to International Journal of Network Management (IJNM), April 2012.

8. Steven Latré, **Jeroen Famaey**, and Filip De Turck. *A Novel Context Authoring Process for Federated Autonomic Management.* Submitted to Knowledge and Information Systems (KAIS), June 2012.

1.6.2 P1: Conference proceedings indexed by ISI Web of Science "Conference Proceedings Citation Index - Science"

1. **Jeroen Famaey**, Tim Wauters, Filip De Turck, Bart Dhoedt, and Piet Demeester. *Dynamic Overlay Node Activation Algorithms for Large-scale Service Deployments.* In proceedings of the 19th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM), Samos, Greece, Pages 14–27, September 2008. doi: 10.1007/978-3-540-87353-2_2.

2. **Jeroen Famaey**, Wouter De Cock, Tim Wauters, Filip De Turck, Bart Dhoedt, and Piet Demeester. *A Latency-Aware Algorithm for Dynamic Service Placement in Large-Scale Overlays.* In proceedings of the 11th IFIP/IEEE International Symposium on Integrated Network Management (IM), New York, USA, Pages 414–421, June 2009. doi: 10.1109/INM.2009.5188843.

3. **Jeroen Famaey**, Bart De Vleeschauwer, Tim Wauters, Filip De Turck, Bart Dhoedt, and Piet Demeester. *Dynamic QoE Optimisation for Streaming Content in Large-Scale Future Networks.* In proceedings of the 1st IFIP/IEEE International Workshop on Management of the Future Internet (ManFI), New York, USA, Pages 128–134, June 2009. doi: 10.1109/INMW.2009.5195948.

4. **Jeroen Famaey**, Wim Van de Meerssche, Steven Latré, Stijn Melis, Tim Wauters, Filip De Turck, Koen De Schepper, Bart De Vleeschauwer, and Rafael Huysegems. *Towards Intelligent Scheduling of Multimedia Content in Future Access Networks.* In proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS), Osaka, Japan, Pages 821–824, April 2010. doi: 10.1109/NOMS.2010.5488363.

5. **Jeroen Famaey**, Steven Latré, John Strassner, and Filip De Turck. *An Ontology-Driven Semantic Bus for Autonomic Communication Elements.* In proceedings of the 5th International Workshop on Modelling Autonomic Communication Environments (MACE), Niagara Falls, Canada, Pages 37–50, October 2010. doi: 10.1007/978-3-642-16836-9_4.

6. Hendrik Moens, **Jeroen Famaey**, Steven Latré, Bart Dhoedt, and Filip De Turck. *Design and Evaluation of a Hierarchical Application Placement Algorithm in Large Scale Clouds.* In proceedings of

the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM), Dublin, Ireland, Pages 137–144, May 2011. doi: 10.1109/INM.2011.5990684.

7. **Jeroen Famaey**, Tim Wauters, and Filip De Turck. *On the Merits of Popularity Prediction in Multimedia Content Caching.* In proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM), Dublin, Ireland, Pages 17–24, May 2011. doi: 10.1109/INM.2011.5990669.

8. Koen De Schepper, Bart De Vleeschauwer, Chris Hawinkel, Werner Van Leekwijck, **Jeroen Famaey**, Wim Van de Meerssche, and Filip De Turck. *Shared Content Addressing Protocol (SCAP): Optimizing Multimedia Content Distribution at the Transport Layer.* In proceedings of the 13th IEEE/IFIP Network Operations and Management Symposium (NOMS), Maui, Hawaii, USA, April 2012.

9. **Jeroen Famaey**, Steven Latré, Tim Wauters, and Filip De Turck. *FedRR: A Federated Resource Reservation Algorithm for Multimedia Services.* In proceedings of the 13th IEEE/IFIP Network Operations and Management Symposium (NOMS), Maui, Hawaii, USA, April 2012.

1.6.3 C1: Other international conference proceedings

1. **Jeroen Famaey**, Tim Wauters, Filip De Turck, Bart Dhoedt, and Piet Demeester. *Towards Efficient Service Placement and Server Selection for Large-scale Deployments.* In proceedings of the 4th Advanced International Conference on Telecommunications (AICT), Athens, Greece, Pages 13–18, June 2008. doi: 10.1109/AICT.2008.46.

2. **Jeroen Famaey**, Jef Donders, Tim Wauters, Frédéric Iterbeke, Niels Sluijs, Bart De Vleeschauwer, Filip De Turck, Piet Demeester, and Rudy Stoop. *Comparative Study of Peer-to-Peer Architectures for Scalable Resource Discovery.* In proceedings of the 1st International Conference on Advances in P2P Systems (AP2PS), Sliema, Malta, Pages 27–33, October 2009. doi: 10.1109/AP2PS.2009.12.

3. **Jeroen Famaey**, Steven Latré, John Strassner, and Filip De Turck. *A Hierarchical Approach to Autonomic Network Management.* In proceedings of the 2nd IFIP/IEEE International Workshop on Management of the Future Internet (ManFI), Osaka, Japan, Pages 225–232, April 2010. doi: 10.1109/NOMSW.2010.5486571.

4. **Jeroen Famaey**, Steven Latré, Tim Wauters, and Filip De Turck. *An SLA-Driven Framework for Dynamic Multimedia Content Delivery Federations.* In proceedings of the 5th International Workshop on Distributed Autonomous Network Management Systems (DANMS), Maui, Hawaii, USA, April 2012.

1.6.4 Other publications

1. **Jeroen Famaey**, Bart Dhoedt, and Filip De Turck. *A Hierarchical Approach to Autonomic Service Management.* In proceedings of the 10th FirW PhD Symposium, Ghent, Belgium, Pages 76–77, December 2009.

References

[1] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. S. Wolff. *The past and future history of the internet*. Communications of the ACM, 40(2):102–108, 1997. doi:10.1145/253671.253741.

[2] T. Bates, P. Smith, and G. Huston. *CIDR and ASN assignment report*. http://cidr-report.org. Last accessed: 8 March 2012.

[3] Internet World Stats. *Internet growth statistics*. http://www.internetworldstats.com/emarketing.htm. Last accessed: 24 March 2012.

[4] Cisco Systems. *Cisco visual networking index: Forecast and methodology, 2010–2015*. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf, 2011. Last accessed: 8 March 2012.

[5] N. Agoulmine, editor. *Autonomic network management principles: From concepts to applications*. Elsevier, 2011. doi:10.1016/B978-0-12-382190-4.00001-2.

[6] L. G. Roberts. *A radical new router*. IEEE Spectrum, 46(7):34–39, 2009. doi:10.1109/MSPEC.2009.5109450.

[7] J. Kephart and D. Chess. *The vision of autonomic computing*. Computer, 36(1):41–50, 2003. doi:10.1109/MC.2003.1160055.

[8] B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M. Ó Foghlú, W. Donnelly, and J. Strassner. *Towards autonomic management of communications networks*. IEEE Communications Magazine, 45(10):112–121, 2007. doi:10.1109/MCOM.2007.4342833.

[9] B. Jennings, K. C. Feeney, R. Brennan, S. Balasubramaniam, D. Botvich, and S. van der Meer. *Federating autonomic network management systems for flexible control of end-to-end communications services*. In N. Agoulmine, editor, Autonomic network management principles: From concepts to applications, pages 101–118. Elsevier, 2011. doi:10.1016/B978-0-12-382190-4.00001-2.

[10] P. Mell and T. Grance. *The NIST definition of cloud computing*. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2011. Last accessed: 10 March 2012.

[11] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verslag, 2007.

2

Federated management of the Future Internet: status and challenges

J. Famaey, and F. De Turck

Submitted to International Journal of Network Management

The Internet's original static services have been superseded by rich multimedia services with stringent end-to-end QoS requirements. Additionally, there has been a trend from simple applications offered by a single provider, towards service compositions, managed across the bounds of multiple domains. It is widely accepted that the end-to-end requirements of multimedia and composed services cannot be satisfied by the current Internet, which does not support inter-domain collaboration. The network federation paradigm was advanced to address these limitations. It envisions the automatic negotiation and management of dynamic agreements between network domains, allowing them to collaborate to achieve goals they cannot achieve alone. This chapter presents an overview of state of the art research in the area of federated network management. Specifically, existing definitions are compared and aligned. Moreover, the most important efforts towards an architecture for a federated Future Internet are discussed. Finally, we have identified several important research challenges that need to be tackled before the federated Future Internet vision can be fully achieved. For each of these challenges, existing research efforts are surveyed and remaining open issues identified. The remaining chapters of this dissertation address several of the challenges identified here.

2.1 Introduction

Since its inception, the Internet has evolved from a traditional packet-switched communication network towards a service-oriented delivery platform. Its original static and best-effort services, such as email and the World Wide Web (WWW), have been superseded by rich and complex services with stringent end-to-end requirements. Specifically, multimedia services, such as Internet-Protocol television (IPTV) and Voice-over-IP (VoIP), have recently grown to become the most prevalent source of traffic on the Internet [1]. This type of services requires strong guarantees on the end-to-end Quality of Service (QoS), in order to satisfy end-user quality requirements. In parallel, the Internet's services are evolving from simple applications, offered by a single provider, towards complex end-to-end service compositions managed by multiple providers across administrative domains.

The trend towards complex end-to-end services with stringent requirements strongly contributes to the need for coordination and collaboration across independent network domains. In the current Internet, coordination across management domains exists only on a very limited scale, consisting of long-term and static collaborations with manually negotiated contracts [2]. Additionally, their scope is limited primarily to the participation in end-to-end routing protocols, network peering arrangements and the exchange of limited management information (e.g., for charging or billing purposes). Nevertheless, such static agreements are inadequate in light of ever-changing end-user needs, network dynamics and evolving service requirements. In order to address this issue, Future Internet research has spawned the *federated network management* paradigm. Federated network management supports the coordination, interaction and collaboration of independently managed network domains through automatically negotiated and managed agreements. It aims to facilitate the delivery of value-added end-to-end services across the Internet.

Although the ideas behind federated network management have existed for some time, several open issues and challenges remain to be solved. This chapter discusses the most important challenges associated with federated management of the Future Internet, surveys the current state of the art in research and identifies the remaining open issues. More specifically, the remainder of this chapter is structured as follows. Section 2.2 analyses the plethora of existing definitions of network federations and attempts to merge them into a unified definition encompassing different views on the topic. Several architectures have been proposed to incorporate federations into the design of the Future Internet. An overview of the most important federated Future Internet architectures is given in Section 2.3. Subsequently, Section 2.4 presents the important challenges and evaluates the status of current research efforts concerning them. Finally, the chapter is concluded in Section 2.5.

2.2 Definition

Even in the narrow context of communications networks, the term *federation* has been defined in many ways over the years. This section lists some well known definitions from literature and identifies the common characteristics in order to align them. Originally, the term federation stems from political jargon. The Oxford English dictionary defines it as follows:

> *"The formation of a political unity out of a number of separate states, provinces, or colonies, so that each retains the management of its internal affairs"*

The adoption of the term in the context of communications networks has given rise to a plethora of definitions, derived from the original political definition. Panlab [3], a federated European test-bed facility, defines a federation as follows:

> *"A model for the establishment of a large scale and diverse infrastructure for the communication technologies, services, and applications and can generally be seen as an interconnection of two or more independent administrative domains for the creation of a richer environment and for the increased multilateral benefits of the users of the individual domains".*

Additionally, several definitions have been advanced in the context of network management specifically. Serrano *et al.* [4, 5] define a federation as:

> *"A set of domains that are governed by either a single central authority or a set of distributed collaborating governing authorities in which each domain has a set of limited powers regarding their own local interests"*

Finally, Jennings, Feeney *et al.* [2, 6, 7] have come up with an alternative definition:

> *"A persistent organizational agreement that enables multiple autonomous entities to share capabilities in a controlled way"*

The presented definitions consider a federation to be an agreement between a set of independent entities or network domains, that retain the responsibility over their internal management. As explicitly stated by Serrano [5], the federation is either governed by a central authority or by the independent entities themselves in a distributed manner. The agreement pertains to the (possibly restricted) sharing of a set of capabilities between the federation partners. Jennings and Feeney [2, 7] clarified that the term *capability* should be interpreted broadly, and might range from the usage of network infrastructure to the configuration of a specific device

or software component. Finally, the federation should be persistent, which means that it should outlive individual interactions and transactions. Note that this does not imply that it should be in any way permanent.

2.3 Architectures and models

This section describes state of the art Future Internet management architectures and models that incorporate support for federations among independent network domains. Specifically, two important architectural models towards federated management of the Future Internet are explored: the *Layered Federation Model* from the FAME research cluster, as well as the *Autonomic Internet architecture* from the European AutoI project.

2.3.1 Layered Federation Model

The Layered Federation Model (LFM) [2, 7] was proposed within the context of the Federated, Autonomic End-to-End Communications Services Strategic Research Cluster (FAME)[1], a project funded by Science Foundation Ireland. The LFM is a general purpose high-level conceptual model of the components of a federal agreement. It captures and reflects the factors that may vary across federal arrangements and models their evolving, dynamic nature. Figure 2.1 depicts the model's six layers.

Each layer of the model represents an aspect of a federal agreement. Each layer builds upon the underlying layers and cross-layer interactions may occur. Additionally, in some agreements, there may be empty layers. Specifically, the model is composed of the following layers:

- **Trusted communication layer**: In order to facilitate the communication between independent management domains, a communication channel must be configured that satisfies the security and trust requirements of both parties. They must agree on communication protocols and security mechanisms. This is the most fundamental layer of the model, as all higher level agreements and interactions make use of it.

- **Federal relationship definition layer**: This layer supports the definition and transmission of the basic rules that govern the relationships between federal partners. This provides a generic methodology to negotiate on the rules concerning membership of a federation and sharing of capabilities.

- **Shared semantic layer**: The goal of a federation is to share capabilities among network domains, in order to obtain some added advantage for all

[1] http://www.fame.ie

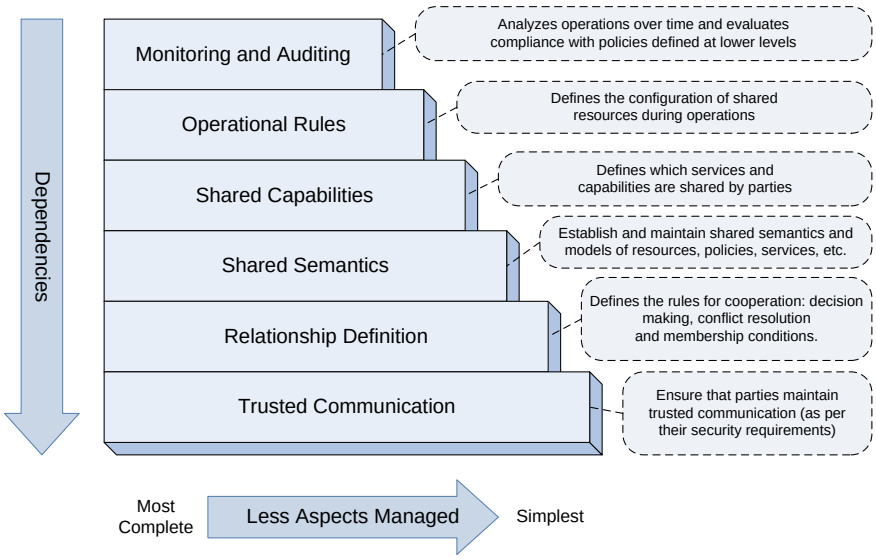


Figure 2.1: Layered Federation Model (from [7])

participating parties. However, the involved parties will usually have their own mechanisms for addressing and describing their internal capabilities. The goal of this layer is to align these diverging mechanisms and descriptions by providing a mapping between the internally used semantics.

- **Shared capabilities layer**: On top of a sufficiently secure communications channel, a relationship agreement and a semantic mapping to facilitate mutual understanding, the actual sharing of capabilities can be supported. This consists of operations to add or remove capabilities to a shared pool, as well as discovery mechanisms to allow other parties to find the capabilities available for use at any particular time.

- **Operational rule layer**: An extension of the capability sharing layer that allows federation partners to view and configure the capabilities shared by others.

- **Monitoring and Auditing layer**: Although the layers of the LFM are expected to manage their own auditing, reporting and compliance assurance, this layer adds additional facilities. It supports long term monitoring and auditing through aggregation, as this might be required by some federal agreements.

In order to add support for the abstract LFM in actual network management systems, Feeney *et al.* proposed the Federal Relationship Manager (FRM) [7].

It is designed to interconnect existing network management systems, through the implementation of several aspects of the LFM. In order to reduce the complexity and cost to deploy the FRM, it minimizes the necessity for common technologies, protocols, models and processes within the participating network domains. The goal of the FRM is to adopt the set of common technical aspects that an organization must adopt in order to manage and maintain federal relationships. It incorporates two components, a semantic ontology mapping framework and a Community Based Policy Management System (CBPMS). The ontology mapping framework enables the efficient and effective creation and management of mappings between domains in order to increase understanding of shared capabilities across federations. The CBPMS provides secure authority management capabilities that are policy language and information-model neutral.

A further application of the LFM was introduced by Brennan *et al.* [8], in the form of the multi-domain relationship management architecture. The LFM is renamed the Layered Relationship Model (LRM) as it is extended to support hierarchical (i.e., domain compositions) in addition to peer-to-peer (i.e., domain federations) relationships. The multi-domain relationship management architecture is an IT architecture to manage the federation of next-generation communications service providers (CSP). It consists of three components: the domain relationship map (DRM), trusted community-based policy management system (TCBPMS) and relationship traceability map (TM) tool chain. The DRM models the federal relationship from the perspective of an individual participating domain. It provides an instantiation of the operational rules, shared capabilities, shared semantics and relationship definition layers of the LRM. The TCBPMS is an extension of the previously discussed FRM's CBPMS, which also incorporates an instantiation of the LRM's trusted communication layer. The relationship TM tool chain automatically generates TMs, which document the relationship between interacting software components within and across network domains. This architecture thus incorporates the features of the FRM, as well as several novel aspects.

2.3.2 Autonomic Internet Architecture

The autonomic Internet (AutoI) architecture was proposed within the context of the European FP7 AutoI project [9]. The architecture's aim is to support self-managing virtual resources that can span across heterogeneous networks. Although the project's focus is on autonomic and self-managing next-generation service-aware networks, it incorporates the federation aspect by supporting the management and sharing of resources across the bounds of administrative domains. The AutoI architecture is composed of five layers, the OSKMV planes [10]:

- **Virtualization Plane**: Virtualizes physical resources in order to support on-the-fly migration and reconfiguration of network resources.

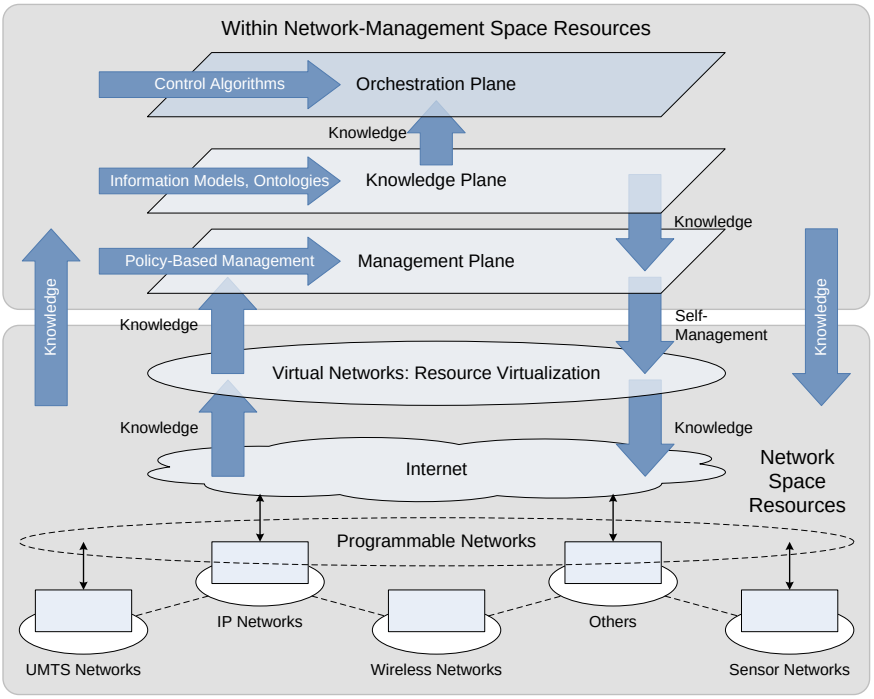


Figure 2.2: Autonomic Internet (AutoI) Architecture (from [9])

- **Management Plane**: Deals with the creation and management of individual autonomic control loops. These loops are realized by Autonomic Management Systems (AMSs), which represent organisational or administrative boundaries.

- **Service Enablers Plane**: Responsible for the discovery, deployment, and composition of services.

- **Knowledge Plane**: A fully distributed information service, responsible for the timely dissemination of information to the other planes. Its inferencing capabilities allow it to derive new knowledge from the gathered information.

- **Orchestration Plane**: Orchestrates the interactions between network domains, AMSs and services.

Figure 2.2 depicts an overview of the architecture and the OSKMV planes. As the AutoI architecture's ability to federate networks stems from its Orchestration Plane (OP) [10], the remainder of this section focusses on the OP and its capability to federate AMSs.

The OP governs the behaviour of the system in response to changing context and in accordance with applicable business goals and policies. It supervises all other planes' behaviour, in order to preserve integrity across the architecture. Its self-governing behaviour is achieved through a set of control algorithms, which can be plugged into it. The OP hosts one or more AMSs. Every AMS represents an independent administrative domain and is responsible for its own internal management. Specifically, the OP enables the federation of AMSs, negotiation of policies, distribution of management tasks and monitoring of AMS behaviour.

The OP is made up of a set of Distributed Orchestration Components (DOCs). A DOC is responsible for a single orchestration domain, which is in itself made up of multiple AMSs. It enables the AMSs of the orchestration domain to communicate and collaborate with each other. Additionally, DOCs can collaborate with each other, in order to provide end-to-end QoS. The DOCs thus serve as the facilitators of inter-domain federations. A DOC performs several tasks that together implement the ability of configure and manage federations. Specifically, these tasks are:

- **Distribution**: This component enables management tasks to be split across AMSs and executed concurrently, both within and across different network domains.

- **Negotiation**: The DOC enables the AMSs to negotiate their business objectives, in order to align them and achieve to a common set of goals for the federation. Two negotiation protocols are currently supported; coalition formation and bargaining.

- **Federation**: Allows a set of independent domains to be combined into a larger virtual domain, with a set of converged high-level goals (obtained through negotiation). The negotiation process additionally aligns the internal domain Service Level Agreements (SLAs) and policies with the high-level federation-wide goals.

- **Governance**: AMSs are self-governing entities, that may decide to change their internal policies or SLAs. This might trigger incompatibilities between internal and federal policies and goals. The DOC's governance component monitors this, and takes appropriate action if such incompatibilities arise (e.g., it might trigger a re-negotiation of the federal agreement).

2.3.3 Summary and comparison

The presented architectures aim to achieve a similar vision; to enable the negotiation, configuration and management of dynamic network federations in the Future Internet. However, their approach and emphasis differs significantly. The LFM

and its instantiations, such as the FRM, focus on the semantic interoperability and compatibility of models and information between independently managed network domains. To achieve this, they employ a semantic ontology mapping framework, which aligns the semantics of information models and context information in order to facilitate unambiguous communication. On the other hand, the AutoI architecture's OP concerns itself with the alignment and compatibility of intra-domain SLAs and policies with the federation-wide goals and high-level policies. Through a set of monitoring processes and SLA negotiation protocols it ensures this compatibility throughout the life-cycle of federations. We believe that, in order to achieve the vision of a Future Internet that supports dynamically adapting network federations, aspects of both architectures will need to be incorporated, guaranteeing both semantic interoperability as well as policy alignment. Additionally, several other challenges will need to be tackled, which will be identified and discussed throughout the next section.

2.4 Status and challenges

The architectures discussed in the previous section conceptually describe how federated network management could be incorporated into the Future Internet. However, concrete algorithms, protocols and solutions are needed to implement the described architectural components. In this section, we identify the, in our opinion, most important challenges that need to be tackled before the federated network management vision can be fully achieved. Additionally, state of the art research that addresses these challenges is evaluated and the remaining open issues are discussed. The following technical challenges are considered:

1. **Security and trust**: Partners in a federation exchange (possibly sensitive) information about their internal management and operations, and allow external parties to access and modify their internal resources. Additionally, the federated Future Internet is expected to support federation agreements without any form of centralized authority. As such, there is a need for decentralized security and trust mechanisms capable of operating in a fully distributed setting without any centralized governing authority.

2. **Semantic interoperability**: The internal semantic representations of models and context information might differ significantly across interconnected network domains. To allow independently managed network domains to cooperate in dynamic and automatically managed federations, these (possibly incompatible) internal representations need to be aligned. This indicates the necessity for semantic mapping techniques that translate between differing internal semantics in order to facilitate unambiguous understanding across network domains.

3. **Agreement negotiation**: Before a federations can be set up, the participating parties need to come to an agreement about the associated costs, benefits, shared capabilities, goals and federation-wide policies. As the involved network domains might have different expectations, requirements and internal (possibly conflicting) policies, their views and goals need to be aligned through the use of negotiation protocols. Once an acceptable compromise has been achieved, the federal agreement can be finalized.

4. **Resource discovery and matchmaking**: The translation of high-level federation goals into specific capabilities, resources and configurations postulates the need for matchmaking and discovery mechanisms. The abstract capabilities and configurations must be mapped unto actual physical or virtual resources that offer the required functionality. This is achieved through scalable and distributed mechanisms that allow shared capabilities and resources, which satisfy specific requirements, to be discovered.

5. **End-to-end resource configuration**: The goal of setting up network federations is to provide some sort of added value to service consumers, which the individual federation partners cannot offer by themselves. To achieve this, capabilities and resources are shared among them. The federation's high-level goals must thus be translated into concrete capability and resource configurations.

6. **Management coordination**: In a federation of network domains, resources and capabilities of the individual domains cooperate in order to satisfy one or more federation-wide goals. The management of the shared capabilities must be coordinated across the participating network domains in order to ensure their cooperation in achieving the expected service benefits. This necessitates the end-to-end monitoring of their state and performance, as well as scalable communications mechanisms that allow federated domains to exchange information about internal resources and policies.

Figure 2.3 depicts the identified challenges and shows where they are positioned within the envisioned federated network management architecture of the Future Internet. The figure denotes the vertical relationships (between the different architectural components within a domain) as well as the horizontal relationships (between same architectural components across domains). Throughout the remainder of this section, the challenges are discussed in more detail.

2.4.1 Security and trust

The network domains participating in a federation interact in different ways. They exchange sensitive information about their internal management and operations,

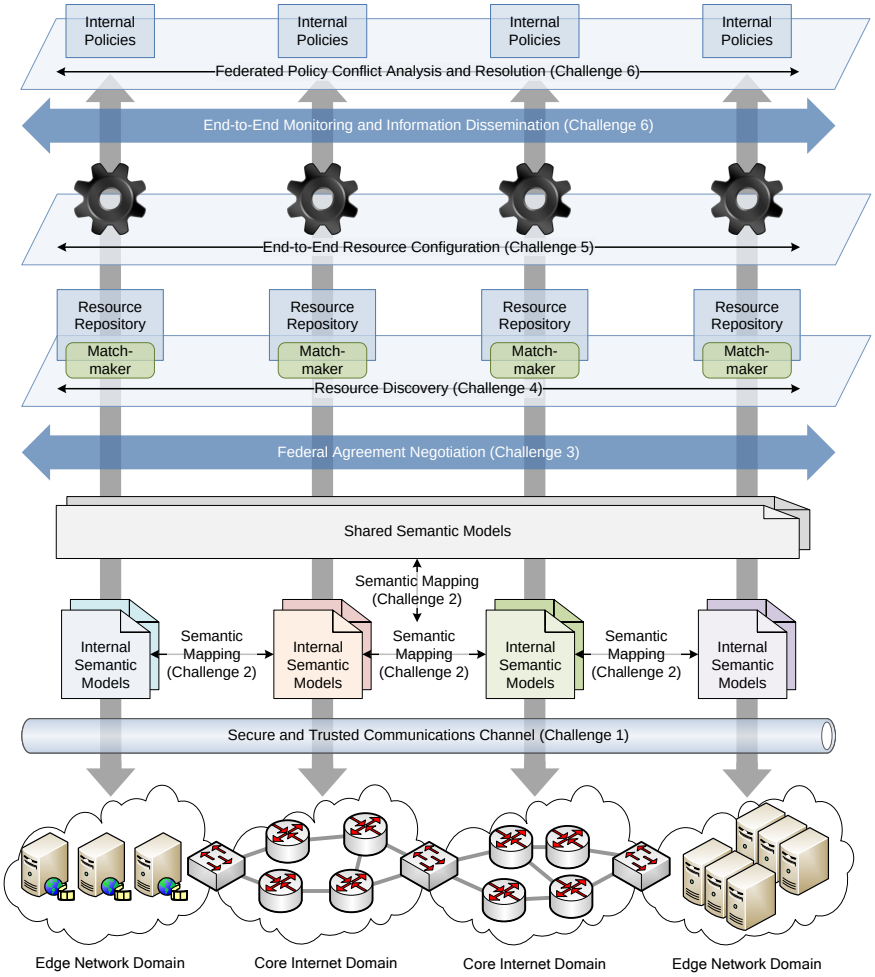


Figure 2.3: The identified challenges and their relationship to each other and the underlying network

and allow external parties to access and modify their internal resources. To ensure the integrity of the federation's participants and prevent malicious tempering with internal resources and information, these interactions should be sufficiently secure and trust should be guaranteed. This challenge thus consists of two parts. First, secure communications channels should be provided as to guarantee secrecy, integrity and authentication of exchanged messages [8]. This can be achieved through public key authentication and encryption (e.g., IPSec, SSL, TLS). As this is a well researched topic, this aspect will not be considered further. Second, trust should be guaranteed between communicating parties in order to prevent the ma-

licious or accidental propagation of fraudulent information and policies. In a centralized system, trust is relatively easily enforceable, since all security policies are centrally managed by a secure and trusted authority. However, in a fully distributed environment, such as a distributed federation of networks, several trust issues arise.

In a federation, the authorization to use internal resources and capabilities is given to external parties. In turn, these authorizations might be delegated to other parties. This gives rise to a specific trust problem called *authorization subterfuge*, which is highly relevant in the context of network federations. Foley and Zhou [11] define it as the problem where "*delegation chains that are used to prove authorization may not actually reflect the original intention of all the participants in the chain*." Specifically, authorization subterfuge is caused by incompetence, confusion or dishonesty. Through incompetence, a delegated authorization credential might be ambiguously defined, given the recipient more access rights than intended. Confusion refers to unintended side effects, caused by authorization policies of which the delegator has no knowledge. Finally, dishonesty means providing third parties unauthorized access by intentionally exploiting incompetence and confusion or denying accountability by claiming to be incompetent or confused. Zhou and Foley [12] propose the Distributed Authorization Language (DAL) to overcome these issues. They argue that existing frameworks for guaranteeing security and trust in distributed systems rely on a centralized security administrator and do not consider authorization subterfuge. The DAL language, however, is the first step towards a fully distributed framework, without centralized control, that is subterfuge-safe.

The FRM framework (cf. Section 2.3.1) is based on a trust model for secure delegation of capabilities in federated systems [13], referred to as the *capability authority model*. This model describes the capabilities provided by a domain and how authority to invoke and manage them is distributed to domains though ownership relations and delegation. It supports two modes; non-trusted and trusted. In non-trusted mode, every domain keeps track of the domains that have been given access rights to specific capabilities. The usage of capabilities always follows the chain of delegation. However, following the chain of delegation might become a performance bottleneck in large-scale scenarios with many embedded federal agreements. The trusted mode can be used to overcome this. It allows delegated capabilities to be directly invoked by any domain that holds its access rights. Through guaranteed uniqueness of permission identifiers, delegation subterfuge is prevented.

Another problem that arises in the context of distributed security and trust is the interoperability of heterogeneous access control mechanisms. Although the generic semantic interoperability problem is discussed further in Section 2.4.2, we consider here the specific interoperability problems arising in the context of access control. Traditionally, the interoperation problem of access policies is solved

by enforcing a common vocabulary for policy attributes [14]. Martínez-García *et al.* [15] argue that this cannot be assumed in independently designed and managed systems. They chose another approach to tackle the problem, which converts policy attributes from one representation to another. This allows access control mechanisms to interoperate without the need for a shared vocabulary. Concretely, a fuzzy set theory approach is proposed, which determines the relationship between attributes using membership functions. The approach is extended to the interoperability problem between more than two domains, where the number of possible mappings increases exponentially. This is alleviated by chaining attribute conversion mappings together. Every domain maintains mappings for a small subset of trusted other domains. Creating a mapping to another, unknown, domain can be done by first combining the mapping of a known domain, with one or more other mappings maintained by other domains.

Traditional frameworks and mechanisms for security and trust in heterogeneous and distributed environments were based on a trusted central management component that maintains and checks access rights and security policies. Additionally, they do not consider federation-specific issues, such as delegation subterfuge and interoperability. Recently, some advances have been made towards fully distributed trust management systems, without the need for a central managing entity and built-in methods to detect and overcome delegation subterfuge. These research efforts should be combined to provide a framework for configuring and maintaining dynamic federal trust relationships, capable of interoperating with heterogeneous intra-domain access control systems.

2.4.2 Semantic interoperability

Automatically configuring and managing dynamic federations among network domains requires autonomic agents located within these independently managed networks to interact and communicate. They must be capable of discovering and configuring shared resources and negotiate mutually beneficial agreements. This is only possible if these autonomic agents are capable of understanding each other and the information they exchange. Unambiguous understanding between autonomic entities is generally achieved through the use of shared semantic models [16]. This facilitates the autonomic understanding and interpretation of policies [17], context information [18], services [19] and shared resources [7]. However, it is infeasible to assume the same set of semantic models can be shared among all network domains in a federated Future Internet. This necessitates the need for mapping the semantics used internally by the different network domains [2, 7]. The importance of this challenge was first indicated by Jennings *et al.* [6] in 2009. It is also reflected in the LFM (cf. Section 2.3.1), where the *shared semantic layer* is responsible for mapping the diverse internal semantic models unto a standard-

ized semantic language or unto one another (if such a standardized language is missing).

The creation of mappings between semantic models has been the topic of much research in the recent past [20, 21], and is often referred to as *ontology mapping*. According to Choi, Song and Han [21], ontology mapping techniques can be classified into three categories; (1) mapping a global into a set of local ontologies, (2) mapping between local ontologies and (3) mapping on ontology merging and alignment. Semantic mapping in the context of network federations is mainly related to the second category. In the context of this category, Choi defines ontology mapping as "*the process that transforms the source ontology entities into the target ontology entities based on semantic relation*." It is most useful for highly dynamic, open and distributed environments, such as network federations in the Future Internet. Traditional ontology mapping tools rely on significant intervention of domain experts or knowledge engineers to aid in the mapping process [22–24]. This results in constrained and static mappings, which makes these approaches ill-suited for dynamic and automatically configured and managed network federations [6]. Besana *et al.* [25] propose an algorithm specifically focussed on mapping diverse and dynamic ontologies with only partially overlapping knowledge domains. They claim that it is infeasible to create all possible mappings in advance, due to the huge amount of possible combinations. Additionally, mapping complete ontologies at runtime is a computationally expensive task. Although they note these problems in the context of semantic peer-to-peer networks, they also arise in the context of network federations, where many network domains can potentially interact with one another. To solve these problems, Besana proposes a novel algorithm that maps only those ontological concepts that are relevant for the interaction at hand. Specifically, the mapping framework dynamically maps concepts when they are first encountered during an interaction. This mapping process is iterative and consists of three steps; hypothesis generation, filtering and selection. Although the framework aims to automate the entire mapping process eventually, the described algorithm does not offer a solution for all steps, necessitating the intervention of human experts.

The semantic interoperability problem has also been studied in the more specific context of Future Internet network management. Strassner, Serrano *et al.* introduced the inference plane [26, 27] an evolution of Clark's knowledge plane [28]. The original knowledge plane does not consider the heterogeneity of technologies, devices and information models in independently managed network domains. The inference plane extends the knowledge plane to be able to cope with this through semantic interoperability. Specifically, they propose the use of a *lingua franca*, based on a set of common information models and ontologies. This set of common models can then serve as a lexicon to translate the different semantic models into a mutually understandable format. Although this approach does not require

administrative domains to change their internally used semantics and models, it does obligate them to be able to translate their internal models into the globally agreed upon *lingua franca*. Wong *et al.* [29] proposed a semi-automatic ontology mapping algorithm for the communication between network domains. Although their approach is capable of automatically mapping different ontologies unto one another, the algorithm has several parameters that need to be manually configured by domain experts. As the configuration of these parameters depends on the nature of the interaction, human intervention is required whenever the context within which the network domain interacts changes. Finally, the FRM (cf. Section 2.3.1) incorporates a semantic mapping framework [7, 30], based on the ontology mapping approach presented by O'Sullivan *et al.* [31]. When a new federal relationship is created, the FRM attempts to re-use and adapt known mappings to facilitate the interoperability between the domains taking part in the relationship. The original mappings that are used as a basis for this process could be created through manual or (semi-)automated processes. They argue that re-use avoids unnecessary redundancy and prevents an explosion in the number of created and deployed mappings. Additionally, this allows the FRM to efficiently deal with dynamism of data and schemata, which they argue has been ignored in research to date on semantic interoperability.

In conclusion, we found that existing methods for the facilitation of semantic interoperability require significant intervention by human experts. This obviously hinders the automatic and dynamic configuration and management of network federations. However, it has been argued that the fully automatic generation of mappings between semantic models is difficult, if not impossible, due to the uncertainty related to matching two ontologies or other semantic models [32]. To accomplish the vision of fully automated federation life-cycle management, human interventions in the creation of mappings should be performed in an offline manner. This can be achieved by employing a common set of information models, for which mappings can be created in advance [26]. This would allow network domains to employ internal semantics for intra-domain management, while the shared semantic models would serve as a *lingua franca* to facilitate semantic interoperability in inter-domain affairs. This has the added advantage of preventing an explosion in the number of mappings, which would have to be created if no common shared models exist. Finally, most existing semantic mapping approaches are concerned with mapping entire models unto one another, which is a computationally expensive operation. In order to guarantee scalability, semantic mapping approaches should be able to determine the relevant subset of models, based on the federation's context. Some promising early work on this topic [25] was performed in the context of peer-to-peer networks.

2.4.3 Agreement negotiation

As stated in the definitions presented in Section 2.2, a network federation is characterised by an agreement between the participating parties. This agreement formally stipulates the rights and obligations of the involved network domains, which usually pertain to a set of shared resources or capabilities. Additionally, the agreements should specify the revenue sharing strategy, which determines how monetary gains, if any, are split across the participants. Today, the negotiation of federal agreements between network domains is a manual and time-consuming process, where business managers, lawyers, and network operators define the business, legal and technical aspects that the participating parties must adhere to. In the Future Internet, where federations will be dynamically and automatically initiated based on changing needs and requirements, this manual process should be (at least partly) automated. We envision an agreement negotiation mechanism that is automatically executed by autonomic agents. Nevertheless, they perform these negotiations within the bounds specified by high-level business, legal and technical policies, defined by human managers and operators. As such, humans are no longer directly involved in these negotiations, but can influence and govern them through the definition of high-level policies.

Automated negotiation protocols have been most commonly proposed in the context of SLA negotiation. An SLA is a formal agreement between a service provider and its customer. It specifies the terms under which a service is delivered, such as the quality, availability, or QoS guarantees. Several standardization efforts exist for the negotiation of Web Service SLAs. The Web Service Level Agreement (WSLA) specification was first proposed by IBM in 2001 [33]. It addresses the specification, creation and monitoring of SLAs. Concretely, the SLAs specify the obligations of the service provider, in terms of IT-level service parameter guarantees (e.g., availability, response time and throughput). Additionally, WSLA specifies the measures that should be taken in case the provider fails to meet its obligations. Although WSLA is equipped to incorporate an automated negotiation protocol, the actual protocol is outside its scope. More recently, the Web Services Agreement (WS-Agreement) [34] specification was introduced by the Open Grid Forum. It is a Web Service protocol for establishing agreements between parties, and has goals similar to WSLA. The agreements themselves are specified in an XML-based language. WS-Agreement incorporates three main components; a schema for specifying agreements, a schema for specifying agreement templates and a set of operations for managing their life-cycles (i.e., creation, expiration and monitoring). In contrast to WSLA, WS-Agreement does propose its own negotiation protocol [35]. It allows two parties to negotiate on the terms of an agreement. If a compromise is reached, the negotiation results in the creation of an actual agreement using the WS-Agreement specification. Hudert et al. [36] extended the WS-Agreement specification, adding support for multilateral, in additional to

bilateral, negotiations. However, they do not propose an actual multilateral negotiation protocol.

The composition of resources or services positioned within independent managed domains is especially relevant within the context of Grid and Cloud Computing. As such, the automatic negotiation of SLAs has been a topic of interest within these fields for several years. Hasselmeyer *et al.* [37] presented a framework for SLA management in Grids, which incorporates a simple one-phase *discrete-offer-protocol*. It lets the customer send a request for an offer to the service provider. The provider then decides whether to reply with an offer or not (based on its available WS-Agreement SLA templates). If it replies with an offer, the customer can either accept or reject. The protocol thus supports bilateral negotiations, without the possibility for compromise (i.e., the customer can only accept or reject the initial offer, not make a counter-offer). Recently, Parkin *et al.* [38] extended the framework with a more elaborate SLA negotiation protocol. The protocol is a multi-round re-negotiation protocol. The multi-round aspect means that if the initial offer is not accepted, one or more additional offers can be made in an attempt to reach a suitable compromise. Additionally, the protocol supports re-negotiation of existing SLAs, to accommodate changing requirements and business goals. Like its predecessor, the protocol focusses on bilateral negotiations, between a service provider and its customer.

Yan *et al.* [39] propose a generic SLA negotiation protocol, that supports multiple service providers. Specifically, it allows a single customer to negotiate the provisioning of a complex service, of which the individual components are offered by different service providers. The consumer is represented by a set of agents who negotiate with the individual service providers. A coordinating agent makes sure that the individually negotiated SLAs together satisfy the customer's end-to-end QoS requirements. The agents use the *FIPA iterated contract net interaction protocol* [40]. It is a bilateral one-to-many agent negotiation protocol. A single agent (the initiator) requests an offer from a set of other agents (participants). The participants may reply with an offer, or refuse. The initiator then iteratively repeats this process with the remaining participants, until a suitable offer is made or all offers have been refused. As such, Yan's framework does not actually support multilateral negotiation, but rather transforms it into a set of coordinated bilateral negotiations.

The importance of agreement negotiation, within the context of a federated Future Internet, was first recognised by the AutoI project consortium. AutoI's OP (cf. Section 2.3.2) incorporates support for the negotiation of agreements between AMSs [10]. Rubio-Loyola *et al.* [41] propose an algorithm to negotiate service provider coalitions, for AutoI's OP. The algorithm is based on an electronic marketplace, where every service provider publishes its service offerings and associated guarantees. The algorithm acts as a centralized manager, that coordinates the

negotiations on behalf of the customer. The algorithm thus supports the negotiation of multilateral agreements, but requires a trusted central management entity. More recently, Chai *et al.* [42] proposed an alternative negotiation protocol for the AutoI OP. The protocol supports bilateral negotiation between two AMSs and additionally assumes the OP plays a coordinating role. Specifically, it is based on the concept of alternating offers. The two negotiating parties take turns in making an offer. The other party accepts, rejects, or opts out. If the offer is rejected, the other party must make a counter-offer, otherwise the negotiation ends. The probability that a participant accepts an offer is based on a utility function, its patience (which decreases over time) and estimated risk. The utility is a function of the amount of requested resources, the agreements duration, the expected benefits, and the current offer.

The envisioned Future Internet should support federations consisting of many (i.e., more than two) independent network domains, without any need for centralized control or management. Existing agreement negotiation protocols are usually designed for bilateral negotiations [35, 37, 38, 40, 42], which is inconsistent with our vision of large-scale federations. Recently, some protocols for the automated negotiation of multilateral agreements have been presented [36, 41]. They, however, expect a centralized trusted management component that coordinates the negotiation process. Before multi-party federations without any form of centralized control can become a reality, the gap towards an automated, multilateral, fully distributed agreement negotiation protocol must thus be filled. Additionally, existing protocols that support iterative negotiations are usually based on a set of mathematical utility functions that model their satisfaction as a function of the current offer and time. These functions need somehow be mapped unto the human specified business, legal and technical high-level policies that should constrain and guide the negotiation process. This aspect has not been discussed in state of the art research and remains an open issue.

2.4.4 Resource discovery and matchmaking

A federation of network domains is set up in order to achieve a common goal that the federation partners cannot achieve alone. However, before the federation agreement can be negotiated, the initiator must determine the set of network domains and shared resources/capabilities that can achieve this goal. As the Internet consists of many thousands of network domains and many more shareable resources and capabilities, scalable and distributed resource discovery mechanisms are needed that are capable of mapping generic federation goals into concrete physical and virtual resources and capabilities. This challenge thus consists of two closely related topics: (1) the actual discovery of resource and capability descriptions, and (2) matching or mapping goals unto those descriptions. The first topic

is referred to as resource discovery, while the second is called matchmaking.

Scalable resource discovery has been a well researched topic in the areas of Grid and, more recently, Cloud Computing. It has become especially relevant within these areas since the introduction of federated grids and clouds. Ranjan *et al.* stated that a resource discovery mechanisms for global, or federated, grids should should be scalable, fault tolerant and impart a limited overhead on the underlying network [43]. Traditional resource discovery approaches for federated Grids used centralized or hierarchical resource indexing services. Especially the centralized, but also the proposed hierarchical methods are prone to central points of failure and scale poorly to a large number of resources. As a solution, peer-to-peer-based resource indexing and discovery protocols were proposed. Early attempts used unstructured peer-to-peer networks in combination with flooding to broadcast the set of available resources [44, 45]. However, flooding-based peer-to-peer protocols are known to scale poorly in terms of generated network overhead [43]. In an attempt to reduce network overhead, solutions based on structured peer-to-peer networks were proposed [45, 46]. They often use a Distributed Hash Table (DHT) [47, 48], as an underlying routing substrate. In a DHT, data is stored as (key, value) pairs, which can be looked up in a logarithmic number of hops. Resource discovery mechanisms based on DHTs are usually based on mapping a d-dimensional logical key to the 1-dimensional DHT key-space. Every dimension then corresponds to a specific attribute of a grid or cloud resource. In a computational grid, these attributes would be for example CPU, memory, bandwidth and cost. Although this is a viable solution for grids and clouds, where the resource types and attributes are limited and known at design time, this information is unknown in the context of generic network domain federations, where a huge number of different types of resources and capabilities, with widely varying attributes, are available for sharing. Heine *et al.* [49] solve this problem by using an ontology to represent resources and their attributes. This approach allows new attributes and resource types to be added at runtime. However, the d-dimensional queries are split up into d 1-dimensional queries and merged after the results have been returned. This causes potentially huge amounts of useless information to be propagated through the network. Additionally, their approach only supports the equality operator, and not more complex comparison operators. Pipan [50] identified some further drawbacks of DHT-based resource discovery, such as it lack of adaptability in highly dynamic scenarios (i.e., where resources and their attributes change often) and its difficulty to handle rich resource descriptions with many attributes. He proposes a novel overlay network, called TRIPOD, which combines the advantages and reduces the disadvantages of existing structured overlays. Specifically, it is capable of finding resources based on proximity, efficiently processes complex queries and handles dynamics in resource characteristics well.

Matchmaking is most often defined in the context of services, and is concerned

with determining the set of services that match a given set of requirements. Traditional methods employ keyword-based matching. However, this has been shown to lead to low matching precision, due to lack of semantics [51]. More recent algorithms employ semantic service descriptions to improve matching precision and make them machine-understandable. These novel semantic matchmaking algorithms are also more suitable for use in federations, as semantics are an important mechanism to guarantee interoperability between network domains. Several semantic matchmaking algorithms have been proposed for web services. Most early work focussed on matching inputs and outputs [52]. However, more recent algorithms have started taking into account preconditions and effects. Preconditions model the state the environment must be in before the service is executed, while effects define the changes that the service will have on the environment. These algorithms are based on various semantic techniques, such as description logics [51], SWRL rules [18, 53] and SPARQL queries [54]. Existing work on service matchmaking focusses heavily on matching software services. However, resources and capabilities in a generic network federation should be interpreted more broadly, and could for example be physical server resources, network paths, device configurations or software components. As such, semantic description methodologies and matchmaking algorithms need to be adapted to encompass this broad range of discoverable components.

Jennings, Feeney *et al.* [2, 7] recognised the importance of service discovery for network federations in their LFM (cf. Section 2.3.1). The shared capabilities layer is, among other things, responsible for discovering the capabilities that are available at any particular time. Additionally, the FRM [7] contains an instantiation of this functionality, in the form of the "Capability Publication and Discovery" component. It uses an authenticated SPARQL endpoint to find capabilities based on RDF descriptions. The RDF documents describe the web service entry points of the actual underlying capabilities. Although this approach is useful for finding suitable capabilities offered by a known federation partner or candidate. It does not support capability discovery from an unknown source. The authors explicitly stated that this problem is outside the scope of their work.

Resource discovery research to date has mostly focussed on Grid and Cloud Computing scenarios. Although there are some parallels with the envisioned federated Internet scenario, there are also several differences that make a direct application of existing methods difficult. In Grid and Cloud Computing, there are a limited number of resource types and attributes. State of the art resource discovery methods thus assume a limited set of resource types and attributes, that are additionally known at design time. In contrast, a huge variation in resource types and attributes is expected to be encountered in the federated Future Internet. Work to date on matchmaking has focussed on matching objectives to software services. However, the shared resources in a federation of networks could also refer

to, for example, hardware components, network capabilities, or device interfaces. As such, existing description methodologies, as well as matchmaking algorithms, need to be adapted to encompass this. In the context of generic network federations, the discovery of shareable capabilities within a known network domain, and linking them to federation goals has been studied to some extent. However, it is assumed that the candidate domains to include within a federation are known. Determining this set of candidates is an important open question that remains to be solved.

2.4.5 End-to-end resource configuration

The negotiated high-level federation goals need to be mapped unto concrete resource and capability configurations. In the envisioned service-driven Future Internet, such goals usually relate to the added value of delivered end-to-end services. An important driver for federations is the provisioning of end-to-end QoS across a set of independently managed network domains, which cannot be guaranteed in the current best-effort Internet. Network domains will need to cooperate in order to provision end-to-end paths that satisfy bandwidth and other network parameter requirements. Several evolutionary and revolutionary methods have been proposed to extend the Internet with QoS reservation capabilities, through the federation of Internet routing domains.

Evolutionary approaches usually propose extensions to the de-facto standard inter-domain routing protocol BGP (Border Gateway Protocol). Kumar and Saraph [55] propose such an evolutionary solution, based on the Routing Control Platform (RCP). They propose the *Alliance Network* model, which allows Autonomous Systems (AS) to join into federations in order to provide end-to-end QoS for end-users. The QoS-guaranteed paths through the participating ASs are identified and configured using the Virtual Space (VS) routing algorithm [56]. However, their model assumes prior agreements on revenue sharing and information exchange have been negotiated. Due to its semi-static nature, the model therefore does not scale up to the global Internet.

Pouyllau and Douville [57] identified several shortcomings associated with extending BGP to support inter-domain QoS-guaranteed routing, such as scalability and confidentiality problems. To alleviate this, they propose a revolutionary approach, based on the negotiation of SLAs. Every intermediary network domain, referred to as a carrier, offers a set of Service Level Specifications (SLSs) (i.e., the shared capabilities). Every SLS is related to the reservation of a QoS-guaranteed path through the associated carrier domain. The proposed algorithm maps the customer's QoS requirements to a chain of SLSs that together form a QoS-guaranteed end-to-end path. The carriers associated with the selected SLSs subsequently negotiate to form a federation. The composition problem is modelled using the

game theory approach and solved using an algorithm based on Q-learning [58]. Their approach, however, assumes that the federation is negotiated and configured with the help of a third party, which has complete knowledge about the capabilities of each candidate carrier.

Recently, we presented the FedRR algorithm [59]. Its goal is to set up network federations to support end-to-end QoS-guaranteed paths across multiple network domains. It identifies the network domains that need to be included within the federation, as well as the capabilities (i.e., QoS classes and network paths) that need to be shared and reserved within each identified domain. Additionally, it allows cloud providers to be included within the federations supporting the dynamic deployment of content caches inside the network. In line with to Pouyllau's approach, it assumes the set of candidate federation partners, as well as their capabilities, are known by a central governing entity.

Work to date on the transformation of federation goals into specific capability configurations has heavily focussed on specific scenarios (e.g., end-to-end QoS), where the goals, and thus the required capabilities, are known at design time. This circumvents the need to dynamically map goals to specific shared capabilities. However, in order to support dynamic, automatically configured and managed federations with varying goals and requirements, there is need for more intelligent translation mechanisms. They should be able to determine a set of candidate federation partners based on their offered capabilities and the specific requirements of services and end-users. Additionally, existing federated resource configuration methods assume the complete set of network domains and capabilities is known. However, in a network as large as the Internet, consisting of tens of thousands of independently managed network domains as well as millions of soft- and hardware resources, this is an infeasible assumption. To guarantee scalability of federation management architectures, resource configuration will thus need to be combined with scalable mechanisms to discover and match shared capabilities, as discussed in Section 2.4.4. Finally, it is often assumed that a central governing entity oversees the configuration and coordination of resources and capabilities. However, this assumption is inconsistent with the vision that the Future Internet should support fully distributed federations, without the need for a central governing body [2]. To support this vision, resource configuration algorithms and processes need to be adapted to operate in a fully distributed environment, without centralized control and management.

2.4.6 Management coordination

The network domains participating in a federation are expected to collaborate in a coordinated fashion in order to achieve the federation-wide goals. To achieve this, there is a need for distributed management coordination mechanisms that govern

the behaviour of individual participating network domains, and monitor the end-to-end state of federated resources and services. Specifically, it should be possible to detect and solve conflicts between federation-wide goals and policies on one hand, and intra-domain policies on the other. Additionally, scalable information dissemination substrates are needed in order to facilitate the end-to-end monitoring of internal domain states.

The dissemination of aggregated monitoring information and context is necessary in order to guarantee the continuous satisfaction of federation-wide goals. Additionally, to ensure inter-domain understanding and interoperability, the exchanged information should be semantically annotated. The publish-subscribe paradigm is well suited to offer these functionalities. It allows interested parties to subscribe to specific types of events. When an event that matches the subscription is published, it is routed accordingly. The paradigm has been successfully applied to the dissemination of semantic information in large-scale networked environments under the banner of *knowledge based networking* (KBN) [60]. It is an extension of content based networking (CBN) [61], which involves the forwarding of events across a network based on subscription filters based on the semantics of the (meta-)data of the event's contents. KBN extends this and states that the semantics of messages play an important part in the matching of publications to subscriptions. To this end, the Sienna publish-subscribe system, originally devised for CBN, was extended with more expressive semantics for the specification of subscriptions [62] to satisfy the KBN vision. Messages in the original Siena take the form of a set of typed attributes (i.e., name-value pairs). Filters specify constraints on the values of those attributes. The filtering process is thus purely based on the syntactical form of messages. Siena additionally supports patterns, which allows matching on combinations of messages. Carzaniga *et al.* [61] additionally propose a set of efficient and scalable routing strategies to forward messages from publishers to interested subscribers. The KBN extension [62], proposed by Keeney *et al.*, adds limited support for semantic messages and filters. Specifically, three new attribute types are added: ontological properties, concepts and individuals. Through basic ontological reasoning, filtering can be done based on semantic equivalence, super- and subconcept relationships, and property relationships. Throughout the years, several other semantic publish-subscribe frameworks have been proposed. They have varying degrees of semantics and inferencing power, ranging from simple RDF graph matching [63, 64], to full-fledged OWL [18, 65] and SWRL-based reasoning [18]. However, it is difficult to find a good balance between expressive and inferencing capabilities on one hand, and routing performance and scalability on the other. Increasing the expressiveness of the messages and filters, will generally lead to significantly reduced scalability.

An aspect of context dissemination that has been mostly ignored, is the actual specification of subscriptions or filter rules. In the envisioned Future Internet,

where networks are autonomously managed and federations are automatically created, the generation and adaptation of subscriptions and filter rules needs to be automated as well. This process should take into account the requirements and goals of a specific federation, as well as changes in the state of the environment and requirements. Latré *et al.* [66] first identified this problem, and proposed an algorithm to generate semantic filter rules, using an OWL-based reasoning approach. The algorithm takes into account the dynamic requirements of autonomic management components, as well as the changing state of the managed environment. Based on these inputs it generates and adapts filter rules for use in semantic context dissemination frameworks.

Another important aspect of the end-to-end coordination of federations, is the alignment and compatibility of intra-domain with federation-wide policies and goals. This is an important focal point of the AutoI OP (cf. Section 2.3.2) [10]. The AutoI DOC component hosts a set of behaviours, which describe the specific orchestration tasks it performs. The *governance behaviour* is concerned with the alignment and compatibility of goals and policies. Network domains might independently change their internal policies or requirements, which might lead to federation-wide inconsistencies. Specifically, the DOC performs three tasks to detect and correct such inconsistencies. First, it monitors the management actions performed within domains and verifies the alignment between internal configurations and federation goals. Second, if a conflict is detected, the DOC informs the management components of the offending domains. Third, if necessary the DOC will trigger a renegotiation of the federal agreement in order to ensure continued smooth operation of the network. The analysis and resolution of conflicts between policies is a complex topic in itself. Research to date has mostly focussed on the policy conflict analysis within single network domains. In an attempt to extend this research topic to federations, Barron *et al.* extended the DEN-ng policy model to support federation-wide policies [67]. More recently, they outlined a novel policy conflict analysis algorithm for federal management policies [68] based on this extended model.

An important aspect of the federation of network domains is the end-to-end coordination of management behaviour, as well as the alignment of internal and federation-wide goals and policies. To guarantee inter-domain coordination, there is a need for scalable mechanisms to exchange and correlate semantic monitoring information. Existing work on semantic context dissemination either offers good scalability with limited expressiveness or the other way around. It is widely believed good routing performance and scalability cannot be combined with extensive expressive power. The combination of both aspects still requires further study. Additionally, relatively little research has been done on the topic of automatic generation of subscriptions and filter rules. In order to achieve the fully automated configuration and management of federations, this topic should be further

explored. Another important aspect of management coordination is the detection and resolution of local and federal policy conflicts. Research in the area of policy conflict analysis between local and federal policies is very limited. Additionally, the complex topic of conflict resolution has not been addressed at all in the context of federations.

2.5 Conclusion

This chapter presents an in-depth survey of state of the art research on federated management of the Future Internet. The chapter offers several distinct contributions. First, the plethora of definitions of the term *network federation* introduced throughout the years were compared and aligned. Specifically, we attempted to come to a unified vision of federated network management, by combining the important aspects of these existing views. Second, an overview was given of the two most influential Future Internet architectures that include support for federations of network domains; the Layered Federation Model (LFM) and the Autonomic Internet (AutoI) architecture. Third, several important challenges related to the envisioned federated Future Internet were identified and discussed. The state of the art research that addresses these challenges was thoroughly evaluated and remaining gaps were identified. This led to several pertinent conclusions, of which we consider the following to be the most important:

- High-level federation goals should be mapped to specific capability and resource configurations within the participating domains. However, work to date on this topic has focussed on very specific scenarios where the goals, and consequently the mapping to resources, are known at design time. This circumvents the need for dynamic algorithms that map generic federation goals unto specific resource configurations. However, we believe such generic algorithms are needed in order to support generic federations of network domains, with widely varying goals and tasks.

- Additionally, existing end-to-end resource configuration frameworks often assume the existence of a centralized management entity that coordinates the configuration effort. This is incompatible with the vision of fully distributed federations without the need for centralized governance or control.

- The automatic configuration and management of federations requires significant communication and collaboration between automated management components. To allow these components to interact with mutual understanding, the internal semantics of the associated network domains need to be aligned. Existing methods to facilitate semantic interoperability (e.g., based on ontology mapping) rely on significant at-runtime interventions by domain

experts. By adopting a standardized set of shared information models and semantics, the semantic mapping process can instead be done in an offline manner. This would negate the need for at-runtime human intervention, but would require all involved network domains to adopt standardized models and translate their internal models and semantics in advance.

- Existing methods to discovery resources across the bounds of administrative domains were designed specifically for Grid and Cloud Computing scenarios. In such scenarios, the types of resources and their attributes are limited and known at design time. Existing resources discovery methods exploit this assumption and can therefore not be directly applied to a generic federation scenario. Novel resource discovery algorithms and protocols thus need to be designed, capable of handling a huge number of different resources types and attributes, not necessarily known at design time.

- Moreover, the network domains in which discoverable resources reside are assumed to be known. In a large-scale scenario with many thousands of potentially collaborating network domains, this assumption is infeasible. There is thus need not only for efficient resource discovery mechanisms in known domains, but also scalable techniques to identify and select the set of potential federation partners among a huge number of available network domains.

- In order to successfully configure a federation of networks, they should be able to negotiate a federal agreement. Existing protocols for agreement negotiation often support only bilateral negotiations. The few protocols that do support simultaneous multilateral negotiations, rely on a centralized trusted component that governs the negotiation process. As federations are expected to possibly contain a large number of network domains without any centralized governing entity, such existing agreement negotiation protocols cannot be directly applied. Novel protocols that support fully distributed multilateral negotiations, need to be devised.

- An important aspect of federation life-cycle management is the alignment of internal and federation-wide policies and goals. Most research on the topic of policy conflict analysis and resolution considers only single-domain scenarios. Although some research has been done in the area of conflict analysis of federation policies, the subject of conflict resolution in federated scenarios is yet to be addressed.

The remainder of this dissertation addresses several of the challenges and open issues identified above. The *semantic interoperability* challenge (cf. Section 2.4.2) is considered in Chapters 6 and 7. They propose the SCB, which facilitates the

communication between autonomic management components across the bounds of network domains, using semantic models. Semantic interoperability is ensured by way of a minimal subset of these models, which is shared across the domains. Chapter 3 considers the *agreement negotiation* challenge (cf. Section 2.4.3), as it presents a framework and methodology for the negotiation of federation-wide SLAs. However, the actual agreement negotiation protocols are outside its scope. The *resource discovery and matchmaking* challenge (cf. Section 2.4.4) is tackled in Chapter 6, as it proposes a semantic matchmaking algorithm. Although it is applied to the matching of Autonomic Element (AE) goals with specialized management services, its extensible semantic models allow it to be extended to the matching of generic resources and capabilities. The *end-to-end resource configuration* challenge (cf. Section 2.4.5) is addressed in Chapter 3, which proposes an algorithm to map federation goals (in the form of end-to-end QoS requirements) unto network domains and shared resources. Finally, the *management coordination* challenge (cf. Section 2.4.6) is tackled in Chapters 5, 6 and 7. Chapter 5 introduces a scalable architecture for the intra- and inter-domain coordination and governance of AEs. The SCB presented in Chapters 6 and 7 facilitates the intelligent dissemination of semantically enriched context information, to be used for the coordination of AEs, both within and across network domain boundaries.

References

[1] Cisco Systems. *Cisco visual networking index: Forecast and methodology, 2010–2015*. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf, 2011. Last accessed: 8 March 2012.

[2] B. Jennings, K. Feeney, R. Brennan, S. Balasubramaniam, D. Botvich, and S. van der Meer. *Federated autonomic network management systems for flexible control of end-to-end communications services*. In Autonomic Network Management Principles: From Concepts to Applications, pages 101–118. Elsevier, 2011. doi:10.1016/B978-0-12-382190-4.00005-X.

[3] S. Wahle, B. Harjoc, K. Campowsky, T. Magedanz, and A. Gavras. *Pan-european testbed and experimental facility federation – architecture refinement and implementation*. International Journal of Communication Networks and Distributed Systems, 5(1):67–87, 2010. doi:10.1504/IJCNDS.2010.033968.

[4] M. Serrano, S. van Der Meer, V. Holum, J. Murphy, and J. Strassner. *Federation, a matter of autonomic management in the future internet*. In Network Operations and Management Symposium (NOMS), pages 845–849, 2010. doi:10.1109/NOMS.2010.5488357.

[5] M. Serrano, S. Davy, M. Johnsson, W. Donnelly, and A. Galis. *Review and designs of federated management in future internet architectures*. In The Future Internet, pages 51–66. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-20898-0.

[6] B. Jennings, R. Brennan, W. Donnelly, S. N. Foley, D. Lewis, D. O'Sullivan, J. Strassner, and S. van der Meer. *Challenges for federated, autonomic network management in the future internet*. In International Symposium on Integrated Network Management (IM), pages 87–92, 2009. doi:10.1109/INMW.2009.5195942.

[7] K. Feeney, R. Brennan, J. Keeney, H. Thomas, D. Lewis, A. Boran, and D. O'Sullivan. *Enabling decentralised management through federation*. Computer Networks, 54(16):2825–2839, 2010. doi:10.1016/j.comnet.2010.07.006.

[8] R. Brennan, K. Feeney, J. Keeney, D. O'Sullivan, J. Fleck, S. Foley, and S. van der Meer. *Multidomain IT architectures for next-generation communications service providers*. IEEE Communications Magazine, 48(8):110–117, 2010. doi:10.1109/MCOM.2010.5534595.

[9] A. Galis, S. Denazis, A. Bassi, P. Giacomin, A. Berl, A. Fischer, H. de Meer, J. Strassner, S. Davy, D. Macedo, G. Pujolle, J. R. Loyola, J. Serrat, L. Lefevre, and A. Cheniour. *Management architecture and systems for future internet networks*. In Towards the Future Internet – A European Research Perspective, pages 112–122. IOS Press, 2009. doi:10.3233/978-1-60750-007-0-112.

[10] D. Macedo, Z. Movahedi, J. Rubio-Loyola, A. Astorga, G. Koumoutsos, and G. Pujolle. *The AutoI approach for the orchestration of autonomic networks*. Annals of Telecommunications, 66(3):243–255, 2011. doi:10.1007/s12243-010-0187-x.

[11] S. Foley and Z. Hongbin. *Authorization subterfuge by delegation in decentralized networks*. In 13th International Conference on Security Protocols, pages 97–102, 2005. doi:10.1007/978-3-540-77156-2_12.

[12] H. Zhou and S. Foley. *A framework for establishing decentralized secure coalitions*. In 19th IEEE Computer Security Foundations Workshop, pages 270–282, 2006. doi:10.1109/CSFW.2006.5.

[13] K. Feeney, R. Brennan, and S. Foley. *A trust model for capability delegation in federated policy systems*. In 6th International Conference on Risk and Security of Internet and Systems (CRiSIS), pages 1–8, 2011. doi:10.1109/CRiSIS.2011.6061828.

[14] L. Gong and X. Qian. *Computational issues in secure interoperation*. IEEE Transactions on Software Engineering, 22(1):43–52, 1996. doi:10.1109/32.481533.

[15] C. Martínez-García, G. Navarro-Arribas, S. Foley, V. Torra, and J. Borrell. *Flexible secure inter-domain interoperability through attribute conversion*. Information Sciences, 181(15):3491–3507, 2011. doi:10.1016/j.ins.2011.04.023.

[16] J. Strassner, N. Agoumine, and E. Lehtihet. *FOCALE – a novel autonomic networking architecture*. International Transactions on Systems Science and Applications, 3(1):64–79, 2007.

[17] J. Strassner, J. Neuman de Souza, S. van der Meer, S. Davy, K. Barrett, D. Raymer, and S. Samudrala. *The design of a new policy model to support ontology-driven reasoning for autonomic networking*. Journal of Network and Systems Management, 17(1):5–32, 2009. doi:10.1007/s10922-009-9119-3.

[18] J. Famaey, S. Latré, J. Strassner, and F. De Turck. *Semantic context dissemination and service matchmaking in future network management*. International Journal of Network Management, 2011. doi:10.1002/nem.805.

[19] D. Martin, M. Burstein, D. McDermott, S. McIlraith, M. Paolucci, K. Sycara, D. L. McGuinness, E. Sirin, and N. Srinivasan. *Bringing semantics to web services with OWL-S*. World Wide Web, 10(3):243–277, 2007. doi:10.1007/s11280-007-0033-x.

[20] Y. Kalfoglou and M. Schorlemmer. *Ontology mapping: The state of the art*. The Knowledge Engineering Review, 18(1):1–31, 2003. doi:10.1017/S0269888903000651.

[21] N. Choi, I.-Y. Song, and H. Han. *A survey on ontology mapping*. ACM SIGMOD Record, 35(3):34–41, 2006. doi:10.1145/1168092.1168097.

[22] P. Mitra and G. Wiederhold. *Resolving terminological heterogeneity in ontologies*. In 15th European Conference on Artificial Intelligence (ECAI), 2002.

[23] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. *Learning to match ontologies on the Semantic Web*. International Journal on Very Large Data Bases, 12(4):303–319, 2003. doi:10.1007/s00778-003-0104-2.

[24] M. Ehrig and S. Staab. *QOM – quick ontology mapping*. In Third International Semantic Web Conference (ISWC), pages 683–697, 2004. doi:10.1007/978-3-540-30475-3_47.

[25] P. Besana, D. Robertson, and M. Rovatsos. *Exploiting interaction contexts in P2P ontology mapping*. In 2nd International Workshop on Peer to Peer Knowledge Management, 2005.

[26] J. Strassner, M. Ó'Foghlú, W. Donnelly, and N. Agoulmine. *Beyond the knowledge plane: An inference plane to support the next generation Internet*. In First International Global Information Infrastructure Symposium (GIIS), pages 112–119, 2007. doi:10.1109/GIIS.2007.4404176.

[27] M. Serrano, J. Strassner, and M. Ó'Foghlú. *A formal approach for the inference plane supporting integrated management tasks in the Future Internet*. In IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 120–127, 2009. doi:10.1109/INMW.2009.5195947.

[28] D. D. Clark, C. Partridge, C. J. Ramming, and J. T. Wroclawski. *A knowledge plae for the Internet*. In ACM Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM), 2003. doi:10.1145/863955.863957.

[29] A. K. Y. Wong, P. Ray, N. Parameswaran, and J. Strassner. *Ontology mapping for the interoperability problem in network management*. IEEE Journal on Selected Areas in Communications, 23(10):2058–2068, 2005. doi:10.1109/JSAC.2005.854130.

[30] R. Brennan, K. Feeney, B. Walsh, H. Thomas, and D. O'Sullivan. *Explicit federal relationship management to support semantic integration*. In 1st IFIP/IEEE Workshop on Managing Federations and Cooperative Management, pages 1148–1155, 2011. doi:10.1109/INM.2011.5990575.

[31] D. O'Sullivan, V. Wade, and D. Lewis. *Understanding as we roam*. IEEE Internet Computing, 11(2):26–33, 2007. doi:10.1109/MIC.2007.50.

[32] J. Keeney, D. Lewis, D. O'Sullivan, A. Roelens, V. Wade, A. Boran, and R. Richardson. *Runtime semantic interoperability for gathering ontology-based network context*. In 10th IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 56–65, 2006. doi:10.1109/NOMS.2006.1687538.

[33] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. *Web service level agreement (WSLA) language specification*. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf, 2003.

[34] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. *Web services agreement specification (WS-Agreement)*. http://www.ogf.org/documents/GFD.193.pdf, 2011.

[35] D. Battré, F. Brazier, K. Clark, M. Oey, A. Papaspyrou, P. Wieder, and W. Ziegler. *WS-Agreement negotiation version 1.0*. http://www.ogf.org/documents/GFD.192.pdf, 2011.

[36] S. Hudert, H. Ludwig, and G. Wirtz. *Negotiating SLAs – an approach for a generic negotiation framework for WS-Agreement*. Journal of Grid Computing, 7(2):225–246, 2009. doi:10.1007/s10723-009-9118-3.

[37] P. Hasselmeyer, H. Mersch, B. Koller, H.-N. Quyen, L. Schubert, and P. Wieder. *Implementing an SLA negotiation framework*. In Expanding the Knowledge Economy: Issues, Applications, Case Studies (eChallenges), pages 154–161, 2007.

[38] M. Parkin, P. Hasselmeyer, B. Koller, and P. Wieder. *An SLA re-negotiation protocol*. In 2nd Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop, 2008.

[39] J. Yan, R. Kowalczyk, J. Lin, M. B. Chhetri, S. K. Goh, and J. Zhang. *Autonomous service level agreement negotiation for service composition provision*. Future Generation Computer Systems, 23:748–759, 2007. doi:10.1016/j.future.2007.02.004.

[40] Foundation for Intelligent Physical Agents (FIPA). *Fipa iterated contract net interaction protocol specification*. http://www.fipa.org/specs/fipa00030/SC00030H.pdf, 2002.

[41] J. Rubio-Loyola, C. Merida-Campos, S. Willmott, A. Astorga, J. Serrat, and A. Galis. *Service coalitions for future internet services*. In IEEE International Conference on Communications (ICC), 2009. doi:10.1109/ICC.2009.5199454.

[42] W. K. Chai, A. Galis, M. Charalambides, and G. Pavlou. *Federation of Future Internet networks*. In IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 209–216, 2010. doi:10.1109/NOMSW.2010.5486573.

[43] R. Ranjan, A. Harwood, and R. Buyya. *Peer-to-peer-based resource discovery in global grids: A tutorial*. IEEE Communications Surveys, 10(2):6–33, 2008. doi:10.1109/COMST.2008.4564477.

[44] A. Iamnitchi, I. Foster, and D. Nurmi. *A peer-to-peer approach to resource location in grid environments*. In 11th IEEE International Symposium on High Performance Distributed Computing (HPDC), page 419, 2002. doi:10.1109/HPDC.2002.1029949.

[45] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi. *Peer-to-peer resource discovery in grids: Models and systems*. Future Generation Computer Systems, 23(7):864–878, 2007. doi:10.1016/j.future.2006.12.003.

[46] S. Basu, S. Banerjee, P. Sharma, and S.-J. Lee. *NodeWiz: peer-to-peer resource discovery for grids*. In IEEE International Symposium on Cluster Computing and the Grid (CCGrid), pages 213–220, 2005. doi:10.1109/CCGRID.2005.1558557.

[47] A. Rowstron and P. Druschel. *Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems*. In IFIP/ACM International Conference on Distributed Systems and Platforms, pages 329–350, 2001. doi:10.1007/3-540-45518-3_18.

[48] I. Stoica, P. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan. *Chord: a scalable peer-to-peer lookup protocol for internet*

applications. IEEE/ACM Transactions on Networking, 11(1):17–32, 2003. doi:10.1109/TNET.2002.808407.

[49] F. Heine, M. Hovestadt, and O. Kao. *Towards ontology-driven P2P grid resource discovery*. In Fifth IEEE/ACM International Workshop on Grid Computing, pages 76–83, 2004. doi:10.1109/GRID.2004.61.

[50] G. Pipan. *Use of the TRIPOD overlay network for resource discovery*. Future Generation Computer Systems, 26(8):1257–1270, 2010. doi:10.1016/j.future.2010.02.002.

[51] G. Shen, Z. Huang, Y. Zhang, X. Zhu, and J. Yang. *A semantic model for matchmaking of web services based on description logics*. Fundamenta Informaticae, 96(1):211–226, 2009. doi:10.3233/FI-2009-175.

[52] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. *Semantic matching of web services capabilities*. In First International Semantic Web Conference (ISWC), pages 333–347, 2002. doi:10.1007/3-540-48005-6_26.

[53] A. B. Bener, V. Ozadali, and E. S. Ilhan. *Semantic matchmaker with precondition and effect matching using SWRL*. Expert Systems and Applications, 36(5):9371–9377, 2009. doi:10.1016/j.eswa.2009.01.010.

[54] M. L. Sbodio, D. Martin, and C. Moulin. *Discovering semantic web services using SPARQL and intelligent agents*. Web Semantics: Science, Services and Agents on the World Wide Web, 8(4):310–328, 2010. doi:10.1016/j.websem.2010.05.002.

[55] N. Kumar and G. Saraph. *End-to-end QoS in interdomain routing*. In International Conference on Networking and Services (ICNS), pages 82–88, 2006. doi:10.1109/ICNS.2006.45.

[56] G. Saraph and P. Singh. *New scheme for IP routing and traffic engineering*. In Workshop on High Performance Switching and Routing (HPSR), pages 227–232, 2003. doi:10.1109/HPSR.2003.1226709.

[57] H. Pouyllau and R. Douville. *End-to-end QoS negotiation in network federations*. In 12th IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 173–176, 2010. doi:10.1109/NOMSW.2010.5486578.

[58] H. Pouyllau and G. Carofiglio. *Inter-carrier SLA negotiation using Q-learning*. Telecommunication Systems, 2011. doi:10.1007/s11235-011-9505-5.

[59] J. Famaey, S. Latré, T. Wauters, and F. De Turck. *FedRR: A federated resource reservation algorithm for multimedia services*. In 13th IEEE/IFIP Network Operations and Management Symposium (NOMS), 2012.

[60] D. Jones, J. Keeney, D. Lewis, and D. O'Sullivan. *Knowledge-based networking*. In Second International Conference on Distributed Event-Based Systems, 2008. doi:10.1145/1385989.1386034.

[61] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. *Design and evaluation of a wide-area event notification service*. ACM Transactions on Computer Systems, 19(3), 2001. doi:10.1145/380749.380767.

[62] J. Keeney, D. Roblek, D. Jones, D. Lewis, and D. O'Sullivan. *Extending siena to support more expressive and flexible subscriptions*. In Second International Conference on Distributed Event-Based Systems (DEBS), pages 35–46, 2008. doi:10.1145/1385989.1385995.

[63] J. Wang, B. Jin, J. Li, and D. Shao. *A semantic-aware publish/subscribe system with RDF patterns*. In 28th Annual International Computer Software and Applications Conference (COMPSAC), pages 141–146, 2004. doi:10.1109/CMPSAC.2004.1342818.

[64] M. Petrovic, H. Liu, and H.-A. Jacobsen. *G-ToPSS: Fast filtering of graph-based metadata*. In 14th international conference on World Wide Web (WWW), pages 539–547, 2005. doi:10.1145/1060745.1060824.

[65] H. Li and G. Jiang. *Semantic message oriented middleware for publish/-subscribe networks*. In Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense III, volume 5403, pages 124–133, 2004. doi:10.1117/12.548172.

[66] S. Latré, S. van der Meer, F. De Turck, J. Strassner, and J. Won-Ki Hong. *Ontological generation of filter rules for context exchange in autonomic multimedia networks*. In 12th IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 575–582, 2010. doi:10.1109/NOMS.2010.5488448.

[67] J. Barron, S. Davy, B. Jennings, and J. Strassner. *A policy authoring process and den-ng model extension for federation governance*. In 5th International Workshop on Modelling Autonomic Communication Environments (MACE), pages 73–86, 2010. doi:10.1007/978-3-642-16836-9_7.

[68] J. Barron, S. Davy, and B. Jennings. *Conflict analysis during authoring of management policies for federations*. In 1st IFIP/IEEE Workshop on Managing Federations and Cooperative Management, pages 1180–1187, 2011. doi:10.1109/INM.2011.5990579.

3

Federated resource management for end-to-end multimedia services

J. Famaey, S. Latré, T. Wauters, and F. De Turck

Submitted to Journal of Network and Systems Management

Multimedia services have recently grown to become the prime source of traffic on the Internet. Due to the Internet's best-effort nature and lack of inter-domain collaboration, it is ill-equipped to guarantee the stringent QoS requirements of modern multimedia services. Chapter 2 discussed the recent trend towards a federated Internet, where network domains cooperate to deliver value-added services. This chapter applies the federation concept and proposes a framework to negotiate and configure federation agreements in order to deliver multimedia services with end-to-end QoS guarantees. Specifically, content providers negotiate with the core Internet domains on the path towards its customers, in order to reserve QoS-guaranteed end-to-end paths. Additionally, cloud providers can be incorporated into the end-to-end federations, allowing the content provider to dynamically deploy caches inside the network. An algorithm is presented that selects the optimal set of network domains and capabilities to include in the federation. It minimizes the total delivery cost, while satisfying the customers' QoS requirements. Chapter 4 further discusses the efficiency of the content caches, while Chapter 7 focusses on the management aspects of the clouds that host them. Finally, Chapters 5 and 6 elaborate on managing the federation, once it has been established.

3.1 Introduction

The increasing availability of bandwidth have converted modern communication networks into popular platforms for the delivery of multimedia services, such as Time-Shifted TeleVision (TSTV) and Video on Demand (VoD). Many Internet Service Providers (ISPs) have started offering these IPTV-based services (e.g., Verizon, Comcast). Additionally, so called Over-The-Top (OTT) content providers have started offering VoD and live television services to users across the Internet (e.g., Hulu, Netflix). Both approaches have their advantages and disadvantages. As ISPs offer their multimedia services to users within their own managed IP network, they are able to give guarantees concerning the delivered Quality of Service (QoS). However, the operator needs to compose its own content catalogue, acquiring licenses from copyright owners. Moreover, it can serve this content only to a limited number of potential customers. Therefore, the operator will have to choose between limiting costs by offering only recent and popular content, or satisfying all its customers by providing an extensive catalogue that caters all tastes. On the other hand, OTT content providers can offer their services to a potentially huge number of users across the Internet. However, as the current Internet is a best-effort network, they cannot offer any QoS guarantees. Additionally, the content is delivered over the ISP's infrastructure, without it being involved in the control or distribution of the content. This causes the ISP network to become more heavily loaded, while it does not share in the revenue [1].

In this chapter, we propose an approach to overcome the shortcomings of the two described multimedia content delivery methods. It allows users to consume the wide selection of content offered by the plethora of content providers on the Internet, while additionally supporting end-to-end QoS guarantees. More specifically, a framework is proposed for setting up end-to-end network federations (also called alliances). A federation is defined as a collaboration between network domains, in order to deliver a combined service, or set of services [2]. In the proposed approach, federations are set up between a content provider, a set of intermediary core Internet transit ISPs and an access ISP. This allows the content provider to offer its multimedia content across the Internet to the end-users of the access ISP with guaranteed end-to-end QoS. The intermediary core Internet domains share in the revenues in return for offering bandwidth- and QoS-guaranteed paths through their networks. This chapter thus considers a Future Internet scenario that supports the provisioning of QoS in the Internet core, which is not possible in the current best-effort Internet. The necessity of providing such guarantees in the Future Internet, has been argued by many Future Internet research efforts [3–6]. The content provider does not directly deal with end-users. For scalability reasons, it instead deals with access ISPs, which act on behalf of a set of end-users. The access ISPs specify their QoS requirements in a Service Level Agreement (SLA), on the terms

of which it negotiates with the content provider. The advent of cloud computing has given rise to the possibility of dynamic on-demand reservation of computational and storage resources. This allows content caches to be deployed on-the-fly across the Internet. Throughout this chapter we call such domains, where storage resources can be dynamically reserved by the content provider, storage sites. These storage sites can thus cooperate in the content delivery federations, trading storage resources for part of the revenues. Additionally, in-network deployed content caches can be shared among several access ISPs. This significantly increases their efficiency, as it allows more content to be served without increasing the amount of reserved storage resources. The use of storage sites and cache sharing is a novel aspect of our approach compared to existing end-to-end QoS reservation mechanisms, which consider only direct QoS-constrained paths between the content provider and consumer.

The process of setting up content delivery federations is facilitated through a mathematical model of the problem. The model defines the stakeholders that can take part in these federations, the end-to-end QoS constraints that need to be satisfied and the cost functions associated with delivering the requested multimedia content. Additionally, this chapter presents an algorithm for calculating the optimal content delivery paths based on this model. The algorithm minimizes the total delivery cost for the content provider by identifying a set of suitable stakeholders (i.e., transit ISPs and storage sites) that should be included in the delivery federation. Additionally, it calculates the amount of QoS and storage resources that need to be reserved within each of the identified stakeholder domains. Finally, it guarantees the QoS requirements specified in the access ISP SLAs.

A quantitative evaluation, based on a VoD scenario, is performed to validate the presented algorithm and proposed novel content delivery approach. This evaluation has several goals. First, performance and scalability of the algorithm are characterized. Second, the use of intermediary storage sites is compared to an approach that only employs direct end-to-end delivery paths between the content provider and its customers. Third, the merits of cache sharing are evaluated under a variety of conditions. Finally, we explore the potential benefits of deploying caches in the access network in addition to in-network caches.

The remainder of this chapter is structured as follows. Section 3.2 describes related work in the context of end-to-end QoS and SLA negotiation. A detailed description of the framework is provided in Section 3.3. It defines the stakeholders involved in the content delivery federations, as well as the interactions that take place between them. Section 3.4 goes into more detail about setting up the end-to-end delivery paths and formally models the end-to-end content delivery federation problem solved in this chapter. Section 3.5 presents an algorithm to solve the presented problem. Subsequently, its merits are validated based on evaluation results of a VoD scenario in Section 3.6. Section 3.7 concludes the chapter.

3.2 Related work

The end-to-end QoS reservation problem can be divided into several sub-problems; finding QoS-constrained shortest paths, negotiating SLAs and managing them. All of these aspects have been actively researched within the network management community. The remainder of this section gives an overview of the most important research efforts on each of these related problems and explains the differences with the work presented in this chapter. However, first the novelty compared to our own previous work is discussed.

The work presented in this chapter is based on our earlier work [7, 8]. The first paper [7] focused on solving a static version of the problem presented here. It assumed the path through the Internet core is fixed and considered only a single content cache shared among all customers. Instead, this chapter presents an algorithm capable of finding the optimal path through the Internet core. Additionally, it is capable of constructing more complex delivery trees that consist of multiple content caches, possibly shared among only a subset of all customers. The second paper focused on the SLA management side, and explains how to incorporate the solution of the algorithm into the SLA negotiation process [8].

3.2.1 Multi-constrained optimal path problem

Finding QoS-constrained cheapest paths through a network is equivalent to the multi-constrained optimal path (MCOP) graph problem [9]. This problem has long been known to be NP-complete [10]. Its goal is to find the shortest (i.e., optimal) path in a graph, subject to multiple edge constraints. Throughout the years, many algorithms and heuristics have been proposed to solve this problem [10–15]. Chen and Nahrstedt proposed an approximation heuristic that attempts to find a feasible solution [11]. However, it cannot guarantee to find a path, even if one exists. The TAMCRA algorithm also finds feasible paths without optimization [12]. If its K parameter is chosen high enough, it has a higher chance of finding the actual optimal path. However, this significantly increases its execution time. The H_MCOP heuristic proposed by Korkmaz and Krunz is a fast approximating heuristic that easily finds feasible solutions if the constraint bounds are loose [10]. However, it often does not yield near-optimal solutions. Several algorithms have also been proposed that do find the optimal, or near-optimal solution. The limited path heuristic is based on an extended version of the Bellman-Ford algorithm [14]. Instead of keeping track of all possible paths from source to destination, it stores only a subset. This reduces its execution time, at the cost of optimality. Liu and Ramakrishnan proposed the optimal A*Prune algorithm [13]. It is an adaptation of the A* searching strategy, combined with a pruning strategy that discards candidate paths that cannot satisfy the constraints. Its runtime is exponential, but a polynomial-time heuristic called BA*Prune is also presented.

Finally, Xiao and Boutaba identified several shortcomings with existing MCOP algorithms, making them unsuitable for use in their proposed dynamic service provisioning framework [15]. To alleviate this, they propose a novel fast running heuristic that utilizes a two-step Dijkstra process. First, Dijkstra's shortest path algorithm is employed in reverse in order to determine whether or not there is a feasible path from every node within the network. Second, a normal Dijkstra is used to find the cost-minimizing path that satisfies the constraints. The algorithm has, in the worst case, five times the runtime of Dijkstra's shortest path algorithm. However, it successfully finds the optimal solution in some specific cases where H_MCOP and TAMCRA fail.

3.2.2 End-to-end Quality of Service

The theoretical work on solving the end-to-end QoS-constrained cheapest path problem is complemented by more practical frameworks for setting up end-to-end QoS-constrained federations. Yan *et al.* presented a multi-agent approach to negotiate QoS for the provisioning of service compositions [16]. Every agent is responsible for provisioning a single service component within the composition. A coordinating agent makes sure the total offered QoS satisfies the requested amount. Pouyllau *et al.* proposed a novel algorithm for setting up end-to-end network federations for the delivery of QoS-constrained services [9]. They assume the set of candidate paths has already been found, using an existing MCOP algorithm. They then formulate the problem of selecting the optimal QoS-constrained path as a game theoretic problem. More recently, they have proposed a reinforcement learning algorithm, based on Q-learning and Markov Decision Processes, to solve the previously formulated game [6]. Their goal is to maximize long-term revenues, while performing real-time treatment of customers' requests. Amigo *et al.* focus on the economics of end-to-end QoS-aware federations [1]. They argue that the current service business model of the Internet is unbalanced, where not all intermediary network domains receive their fair share of revenues. To alleviate these economic concerns, they propose an end-to-end bandwidth allocation framework. It allows transit network domains to collaborate in order to offer bandwidth guaranteed end-to-end pipes through the Internet.

In addition to algorithms that find QoS-constrained optimal paths, there is also a need for QoS-aware routing protocols in order to support end-to-end QoS on the Internet. Several evolutionary approaches have been proposed to support end-to-end QoS on top of the current best-effort Internet. Kumar *et al.* presented the Alliance network model [3]. It allows interconnected Autonomous Systems (AS) to form an alliance or federation, which enables optimal inter-domain path selection and QoS guarantees. Additionally, it is compatible with the Border Gateway Protocol (BGP) and can thus co-exist with the current best-effort Internet. Xiangjiang

et al. presented a similar approach, also based on BGP [4].

Our work differs from existing work on end-to-end QoS provisioning, in the sense that we aim to solve an extension of the MCOP problem. In addition to finding QoS-constrained shortest paths, our goal is to set up end-to-end delivery federations that additionally include intermediary content caches. As such, existing MCOP algorithms cannot be directly applied to this extended problem. Nevertheless, they are incorporated into our novel algorithm to solve a subset of the extended problem.

Recently, Balasubramaniam *et al.* proposed an integrated architecture combining service lifecycle management and dynamic end-to-end routing [17]. The proposed management framework is inspired by biological systems, and aims to autonomously configure services and the underlying routing system in order to guarantee the ever-changing end-to-end QoS requirements of a large number of heterogeneous services. Although their research has similar goals to ours, they focus on service lifecycle management and dynamic, distributed routing. In contrast, our work centers around long-term cost minimization through intelligent federation composition and resource management.

3.2.3 SLA negotiation and management

In order to successfully set up a federation, an agreement should be negotiated between the participants. To achieve this, SLA negotiation protocols can be employed. Several frameworks and architectures have been proposed to support SLA negotiation between federation partners. Yuanming *et al.* developed a framework for the negotiation of SLAs between service providers, network operators and content providers [18]. More recently, the SLA-based SERViceable Metacomputing Environment (SERVME) was proposed [19]. It consists of a framework and accompanying SLA model, which guide the SLA negotiation process, match providers based on QoS requirements and perform on-demand resource provisioning. In addition to frameworks that support the negotiation and management of SLAs, several protocols that allow the actual SLA terms to be negotiated have been proposed. The Web Services Agreement Specification (WS-Agreement) is a Web Services protocol for establishing agreement between two parties, such as a service provider and consumer [20]. It consists of an XML-based language for specifying SLAs, as well as a protocol for negotiating its terms [21]. Hudert *et al.* extended these ideas with a framework built around WS-Agreement that supports multilateral in addition to bilateral negotiations [22]. Hasselmeyer *et al.* proposed a Discrete-Offer-Protocol that allows the service provider to make a single offer, which the consumer can accept or reject [23]. More recently they proposed a more elaborate protocol [24]. It supports multi-round negotiations, as well as re-negotiating the terms of an SLA already in place as the requirements of par-

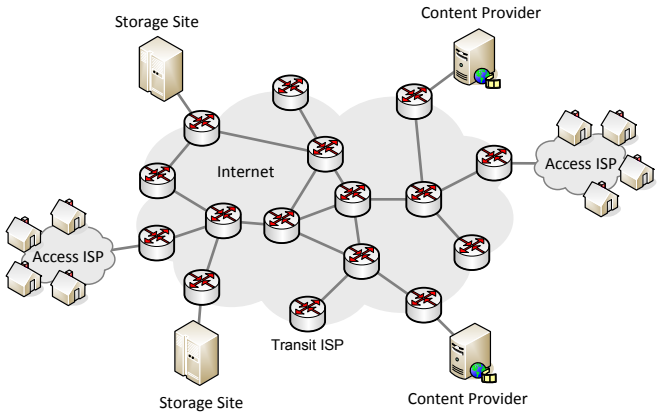


Figure 3.1: An overview of the stakeholders involved in the end-to-end content delivery process: content providers, storage sites, access ISPs and transit ISPs

ticipants change. The actual SLA negotiation protocol is outside the scope of our work. Instead we focus on determining the costs associated with complying to the terms of an SLA. This calculated cost can then be taken into account during the subsequent negotiation process.

3.3 Federated content delivery framework

The goal of the envisioned framework is to facilitate the end-to-end delivery of multimedia content across the Internet. To achieve this, it allows content providers and access ISPs to set up federations with transit ISPs and storage sites along the end-to-end path connecting them. Cooperating in a federation holds advantages for all involved stakeholders. On one hand, it allows the content provider to guarantee QoS across the Internet. On the other, it entitles the intermediary network domains to a share of the content provider's revenue. The remainder of this section identifies the different stakeholders involved in these federations and describes how they interact to set them up. Sections 3.4 and 3.5 further elaborate on the algorithmic details of the federation set-up process.

3.3.1 Stakeholders

Figure 3.1 positions the different stakeholders throughout the network. There are four types of stakeholders involved in the content delivery federations: content providers, access ISPs, transit ISPs, and storage sites. The content providers and storage sites are positioned at the edge of the Internet, connected to the remainder of the network through a single transit ISP. The *content provider* locally hosts a

set of multimedia content items. It aims to sell these to interested access ISPs. Traditionally, the *access ISP* provided Internet access to a set of end-users. Nowadays they often offer novel multimedia services, including TSTV and VoD. Our framework builds on this and considers the access ISPs the direct customers of the content provider. In line with current advances, the access ISP is assumed to offer multimedia services to the end-users. However, in contrast to current approaches, it does not manage its own content catalogue, but rather obtains content via one or more content providers, over the Internet. The Internet core consists of many *transit ISPs*, together forming a network of networks. They connect the different edge domains to the Internet, and are responsible for routing network traffic from source to destination. In the current Internet, end-to-end routing is static and best-effort. However, future Internet research has advanced the idea of on-demand end-to-end QoS provisioning within the Internet core [3, 4, 6]. As such, transit ISPs can be included in the end-to-end federations, providing QoS guarantees in return for a share of the content provider's revenue. A set of *storage sites* is spread across the Internet. They enable on-the-fly provisioning of storage resources. This allows content providers to dynamically deploy content caches across the Internet. Note that this approach differs from the traditional use of Content Delivery Networks (CDNs) for the distribution of content. A traditional CDN retains full control of its resources and decides where to cache what content. Instead, we envision an approach that allows the content provider to manage its leased storage site resources, which is more in line with a cloud-based leasing model. Nevertheless, our approach can be applied to the CDN use-case, by letting the CDN take control of the federation set up procedure, instead of the content provider.

In summary, the presented framework thus combines characteristics and advantages of the two described existing multimedia content delivery approaches. First, it allows content providers to offer their content to end-users across the Internet, as is the case in the OTT scenario. However, in contrast to OTT content delivery, access ISPs are still involved in the delivery process and can share in the revenue. Thus allowing access ISPs to offer a plethora of multimedia services, as they currently do, without needing to maintain their own content catalogue. Additionally, by including transit ISPs and storage sites in the delivery federations, QoS guarantees can be offered and transmission costs can be decreased, further reducing disadvantages of OTT content delivery.

3.3.2 Interactions

The stakeholders involved in the content delivery process interact in several ways to set up QoS-guaranteed end-to-end paths across the Internet. The federation set-up process is initiated by the access ISP when it decides it wants to offer the content of a specific content provider to the end-users connected to its network.

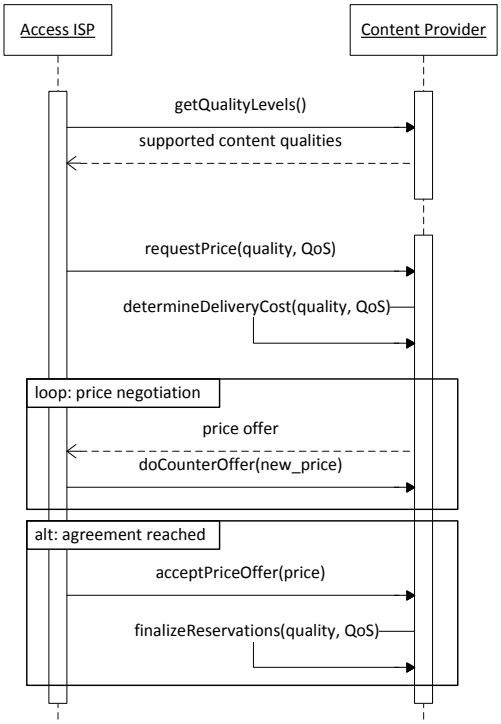


Figure 3.2: A sequence diagram detailing the negotiation process between the content provider and an access ISP customer

Figure 3.2 depicts the negotiation process from the content provider to an access ISP in detail. First, the ISP requests the list of quality levels offered by the content provider. The content provider replies with information about each quality level. Depending on the type of multimedia content the offered information may differ. It could include for example spatial resolution, temporal resolution, and bit-rate. The access ISP can use this plethora of information to determine which quality levels its end-users will be interested in (based on their device capabilities and expectations). From the content provider's perspective the bit-rate is the important attribute, as it determines the amount of bandwidth resources that need to be reserved and consequently the cost associated with delivering the content. Subsequently, the access ISP requests an initial price offer for delivering content with a specific quality and specific QoS requirements (e.g., delay, jitter, packet loss, availability). The content provider uses its price models and associated algorithms to determine the costs associated with delivering its content to the access ISP. To calculate this cost, it additionally needs to find a cost-minimizing end-to-end path and identify the network domains along this path. This is a complex problem in

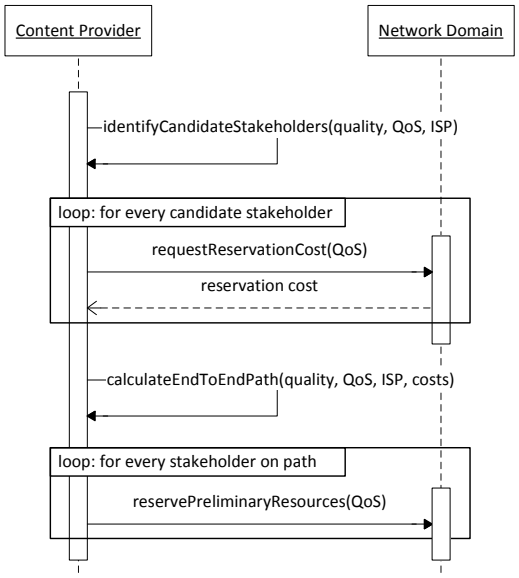


Figure 3.3: A sequence diagram detailing the interactions involved in finding an end-to-end delivery path from the content provider to an access ISP

itself for which we offer a solution in Sections 3.4 and 3.5. The content provider can then formulate an initial price offer to the ISP, based on its calculated delivery costs and the expected profit. This negotiation can end in either agreement or disagreement. If an agreement is reached, the SLA negotiation process is finalized and the ISP can start offering the content provider's multimedia content to its end-users. This finalized SLA contains the negotiated price, content quality, QoS requirements and the time-frame over which it is valid. The negotiated SLA is only valid for a certain period of time to take into account changes in delivery costs. After the time-frame has expired, a new SLA should be negotiated taking into account current parameters and costs. If no agreement is reached during the negotiation procedure, the federation is not finalized. It is then up to the access ISP to start again, either by contacting another content provider or requesting less stringent QoS.

As previously stated, the content provider needs to determine the costs associated with delivering its content across the Internet towards its federated access ISPs. Additionally, it needs to reserve the necessary resources in order to satisfy its obligations captured in the negotiated SLA. This introduces a set of additional interactions between the content provider on one hand and the transit ISPs and storage sites along the end-to-end path on the other hand. Figure 3.3 depicts the interactions involved in finding an end-to-end delivery path between the content provider and an access ISP. In order to calculate the costs involved in delivering its

content to a specific access ISP, the content provider first needs to identify the network domains the content will pass through. These domains can either be transit ISPs or storage sites. In case of the former, the content provider requests the cost per bandwidth unit for reserving a path through the domain with the requested QoS. In case of the latter, it asks the cost for reserving storage resources. This cost information is then employed by the content provider to calculate the cost-minimizing end-to-end path to the access ISP that satisfies the requested QoS. Finally, it performs a preliminary lock on the necessary resources within the domains along this path, to make sure it can satisfy the offer that will be made to the access ISP. These preliminary reservations are finalized as soon as the content provider and access ISP come to an agreement. If they do not, the locks are released.

3.4 End-to-end content delivery federations

The previous section gave an overview of the stakeholders involved in the envisioned content delivery approach, as well as how they interact and set up federations. Before the content provider can successfully initiate an end-to-end federation, it must first identify the set of stakeholders it wishes to involve. The set of stakeholders involved in the end-to-end delivery of multimedia content from the content provider to one of its access ISP customers takes the form of an end-to-end path through the Internet, consisting of transit ISPs and storage sites. This section further elaborates on the problem of finding such an end-to-end path between a content provider and its access ISP customers. Every path should satisfy the QoS constraints requested by the associated access ISP. Additionally, of all possible combinations of paths, the cost minimizing set should be selected. First, the problem domain is mathematically modelled. Second, a formal problem formulation is presented. Subsequently, Section 3.5 presents a heuristic to solve this problem.

3.4.1 Notations & assumptions

Let us consider an Internet topology, consisting of an interconnected set of ISPs $\mathcal{I}$. Every ISP $i \in \mathcal{I}$ is connected to a set of neighbours $\mathcal{I}_i$. There are two types of ISPs: transit ISPs $\mathcal{T}$ and access ISPs $\mathcal{A}$. Additionally, there are several types of edge network domains involved in the content delivery process: a set of content providers $\mathcal{P}$ and a set of storage sites $\mathcal{S}$. Every content provider and storage site $e \in \mathcal{P} \cup \mathcal{S}$ is connected to the Internet through its gateway $t_e \in \mathcal{T}$.

In the considered scenario, it is possible to reserve QoS-guaranteed paths through the Internet core. To this end, every transit ISP $t \in \mathcal{T}$ offers a set of QoS classes $\mathcal{C}_t \subseteq \mathcal{C}$. A QoS class $c \in \mathcal{C}$ offers a QoS guarantee $\gamma_{c,q}$ for every QoS parameter $q \in \mathcal{Q}$ (e.g., delay, packet loss, jitter, availability), and has an associated transmission cost θ_c per unit of content. Concretely, a QoS guarantee specifies a constraint

Table 3.1: The aggregation and satisfaction operators for commonly used QoS types

QoS type	Aggregation operator	Satisfaction operator
Delay	$\sum$	$\leq$
Throughput	$\min$	$\geq$
Packet loss	$\prod$	$\leq$
Availability	$\prod$	$\geq$

on the associated QoS parameter (e.g., delay ≤ 100ms, availability $\geq 99.999\%$). Different QoS parameters are aggregated in different ways. For example, the total end-to-end delay is the sum of the individual link delays, while the total availability is the product of all link availabilities. As such, we define the end-to-end QoS aggregation operator of a QoS parameter $q \in \mathcal{Q}$ as $\oplus_q$. It calculates the aggregated value of the QoS parameter based on the individual link values. Similarly, the satisfaction operator to determine if a QoS value satisfies a requirement depends on the QoS parameter. For example, delay is satisfied if the guaranteed value is below the constraint, while availability is satisfied if it is above. As such, we define the satisfaction operator of a QoS parameter q as $\prec_q$. Specifically, QoS value a satisfies constraint b if $a \prec_q b$ equals true. Table 3.1 summarizes the aggregation and satisfaction operators for some commonly used QoS types.

Every content provider $p \in \mathcal{P}$ offers a content catalogue, available in a set of bit rates $\mathcal{B}_p$. The content catalogue is additionally characterised by some statistical information, which is necessary to calculate the total delivery cost. This information consists of the weighted average content duration δ_p (with the weight proportional to the item's popularity), the number of items in the catalogue χ_p and for every bit rate $b \in \mathcal{B}_p$ a cumulative popularity distribution function $\Phi_{p,b}(\cdot)$. This function takes as input an integer x, and returns as output the probability that an end-user requests one of the x most popular items in the content catalogue. For example, $\Phi_{p,b}(100) = 0.9$, means that 90% of all requests for content with bit rate b received by content provider p are for its 100 most popular items.

The negotiation of SLAs is governed by a set of delivery requests $\mathcal{R}$. These requests stipulate the demands of access ISPs pertaining to the content they want to receive from a specific content provider. Note that these requests are not requests for a single content item, but rather requests for an end-to-end delivery path, over which multiple content items can be sent with specific QoS guarantees. The requests associated with a content provider $p \in \mathcal{P}$ are defined as $\mathcal{R}_p \subseteq \mathcal{R}$, while the set of requests originating from an access ISP $a \in \mathcal{A}$ is defined as $\mathcal{R}_a \subseteq \mathcal{R}$. A request $r \in \mathcal{R}$ originates from an access ISP $a_r \in \mathcal{A}$, which sends it to a content provider $p_r \in \mathcal{P}$. Furthermore, it contains a requested bit rate $b_r \in \mathcal{B}_{p_r}$, an expected number of simultaneous delivered content items ρ_r and an end-to-end QoS constraint $\gamma_{r,q}$ for every QoS parameter $q \in \mathcal{Q}$.

Storage sites support the on-demand reservation of storage resources. This

allows the content providers to deploy dynamic caches throughout the Internet. Several caches, associated with different requests, may be deployed within a single storage site. In order to support cache sharing, a single cache may also be associated with multiple requests. Note that caches can also be deployed within access ISP domains. However, their available resources are expected to be limited compared to those of dedicated storage sites, and a small, static cache is thus associated with them. For genericness, we also assume every content provider hosts a content cache that stores its entire content catalogue. A set of content caches $\mathcal{O}$ is deployed within domains across the Internet. A content cache $o \in \mathcal{O}$ is characterised by the domain $d_o \in \mathcal{S} \cup \mathcal{A} \cup \mathcal{P}$ in which it is deployed, its cache size σ_o and the set of requests $\mathcal{R}_o \subseteq \mathcal{R}$ for which it is used. The cache size is expressed in terms of number of cached objects. It is assumed that all requests $r \in \mathcal{R}_o$ request the same bit-rate $b_o \in \mathcal{B}$ and target the same content provider $p_o \in \mathcal{P}$. A storage cost λ_s and processing cost φ_s are associated with every storage site $s \in \mathcal{S}$ per unit of content. The storage cost is paid for cached content only, while the processing cost is paid for every content request that it serves. It is assumed there are no costs associated with the caches deployed in the access ISP and content provider domains. As such, the incentive for the access ISP to host a cache is a reduction in the price it pays to the content provider rather than an actual share of the revenues.

For every request $r \in \mathcal{R}$, an associated end-to-end delivery path $\Pi_r = \langle o_1, ..., o_n \rangle$ is set up, with $o_i \in \mathcal{O}$ for $i \in [1, n]$, $d_{o_1} = p_r$, $d_{o_n} = a_r$ and if $n > 2$, $d_{o_i} \in \mathcal{S}$ for $i \in [2, n-1]$. In other words, the path consists of a set of content caches, with the source cache deployed within the content provider domain, the target cache within the access ISP domain and any remaining intermediary caches within storage sites. The successor and predecessor of $o \in \Pi_r$ along a path Π_r are respectively defined as o_r^+ and o_r^-. When an end-users requests a content item, the associated content request is sent to the content caches along this path in reverse order (i.e., first to o_n, then to o_{n-1}, ...). The request is forwarded until a cache is encountered that locally stores the requested content item. As the cache associated with the content provider stores all content, every request is eventually answered. The content itself is not sent via the end-to-end delivery path, but via a direct path through the Internet core. As such, a core Internet path $\pi_{o,r} = \langle g_1, ..., g_m \rangle$ is associated with every cache $o \in \Pi_r \setminus \{o_n\}$, with $g_1 = t_{d_o}$, $g_n \in \mathcal{I}_{a_r}$ and $g_i \in \mathcal{T}$ for $i \in [1, m]$. A core Internet path thus consists of only transit ISPs. Its first transit ISP is the gateway of d_o, while its last is a neighbour of a_r. For every transit ISP $t \in \pi$ along the core Internet path $\pi_{o,r}$, a QoS class $c_{t,r} \in \mathcal{C}_t$ is reserved.

In summary, Table 3.2 lists the symbols introduced throughout this section.

Table 3.2: The list of symbols used throughout this chapter

Symbol	Explanation
$\oplus_q$	aggregation operator of QoS parameter $q \in \mathcal{Q}$
$\prec_q$	satisfaction operator of $q \in \mathcal{Q}$
$\mathcal{A}$	access ISPs
$a_r \in \mathcal{A}$	source ISP of request $r \in \mathcal{R}_a$
$\mathcal{B}_p$	bit rates offered by $p \in \mathcal{P}$
$b_o \in \mathcal{B}_{p_o}$	bit rate of content stored in content cache $o \in \mathcal{O}$
$b_r \in \mathcal{B}_{p_r}$	bit rate associated with request $r \in \mathcal{R}$
$\mathcal{C}$	QoS classes
$\mathcal{C}_t \subseteq \mathcal{C}$	QoS classes offered by $t \in \mathcal{T}$
$c_{t,r} \in \mathcal{C}_t$	QoS class reserved in $t \in \mathcal{T}$ for request $r \in \mathcal{R}$
χ_p	number of items in the content catalogue of $p \in \mathcal{P}$
$d_o \in \mathcal{S} \cup \mathcal{A} \cup \mathcal{P}$	domain where content cache $o \in \mathcal{O}$ is deployed
δ_p	average duration of content offered by $p \in \mathcal{P}$
$\gamma_{c,q}$	QoS guarantee of QoS parameter $q \in \mathcal{Q}$ for QoS class $c \in \mathcal{C}$
$\gamma_{r,q}$	QoS constraint of QoS parameter $q \in \mathcal{Q}$ for request $r \in \mathcal{R}$
$\mathcal{I}$	transit and access ISPs
$\mathcal{I}_i \subseteq \mathcal{I}$	the neighbours of $i \in \mathcal{I}$
λ_s	storage cost associated with domain $s \in \mathcal{S}$
$\mathcal{O}$	content caches
$o_r^+ \in \Pi_r$	successor of $o \in \mathcal{O}$ along the end-to-end path Π_r of $r \in \mathcal{R}$
$o_r^- \in \Pi_r$	predecessor of $o \in \mathcal{O}$ along the end-to-end path Π_r of $r \in \mathcal{R}$
$\mathcal{P}$	content providers
$p_o \in \mathcal{P}$	content provider associated with content cache $o \in \mathcal{O}$
$p_r \in \mathcal{P}$	target content provider of request $r \in \mathcal{R}_p$
$\Phi_{p,b}(\cdot)$	cumulative popularity distribution of $p \in \mathcal{P}$ for $b \in \mathcal{B}_p$
φ_s	processing cost on $s \in \mathcal{S}$
$\Pi_r \subseteq O$	end-to-end path associated with $r \in \mathcal{R}$
$\pi_{o,r}$	core Internet path from $d_o \in \mathcal{P} \cup \mathcal{S}$ to $a_r \in \mathcal{A}$
$\mathcal{Q}$	QoS parameters
$\mathcal{R}$	delivery requests
$\mathcal{R}_a \subseteq R$	requests for $a \in \mathcal{A}$
$\mathcal{R}_o \subseteq R$	requests for which content will be served from cache $o \in \mathcal{O}$
$\mathcal{R}_p \subseteq R$	requests for $p \in \mathcal{P}$
ρ_r	number of simultaneous delivered content items for $r \in \mathcal{R}$
$\mathcal{S}$	storage sites
σ_o	cache size of content cache $o \in \mathcal{O}$
$\mathcal{T}$	transit ISPs
$\sqcup_e \subseteq \mathcal{T}$	gateway of $e \in \mathcal{P} \cup \mathcal{S}$
θ_c	reservation cost associated with QoS class $c \in \mathcal{C}$

3.4.2 Problem formulation

As stated, the problem consists of finding the cost minimizing end-to-end delivery path Π_r for every request $r \in \mathcal{R}$. Additionally, for every $o \in \Pi_r \setminus \{o_n\}$ a path $\pi_{o,r}$ through the Internet core from $d_o \in \mathcal{S} \cup \{p_r\}$ to $a_r \in \mathcal{A}$ that satisfies the request's QoS constraints needs to be found. Formally, $\pi_{o,r}$ should therefore satisfy the following constraints:

$$\forall r \in \mathcal{R}, \forall q \in \mathcal{Q}, \forall o \in \Pi_r \setminus \{o_n\} : \bigoplus_{t \in \pi_{o,r}} \gamma_{c_{t,r},q} \prec_q \gamma_{r,q} \tag{3.1}$$

The total cost of Π_r consists of three components: transmission cost, storage cost and processing cost. The storage and processing cost are related to the storage sites, while the transmission cost is related to the reservation of QoS in the transit ISP domains. The total storage cost of all content caches is calculated as follows:

$$\Delta = \sum_{o \in \mathcal{O}} \sigma_o \times \lambda_{s_o} \times b_o \times \delta_{p_o} \tag{3.2}$$

The processing and transmission costs both depend on the number of content items that are served from a specific cache. Additionally, these costs are influenced by other content caches on the path towards the access ISPs. More specifically, the cache deployed nearest the access ISP will serve the most popular content, the second nearest then serves the most popular content not served by the first, and so on. However, as content caches can be shared, they may belong to multiple end-to-end delivery paths and thus have multiple child caches. This makes it difficult to determine the exact number of popular items that are served downstream. As such, we use the minimum of all direct child caches as a lower bound. First, we calculate the aggregated cache size of a content cache $o \in \mathcal{O}$, which is defined as the lower bound on the number of content items that are served by o or any of the caches on the delivery paths from o towards the access ISPs it serves. The aggregated cache size of o is then defined as follows:

$$\sigma_o^{\text{aggr}} = \sigma_o + \min_{r \in \mathcal{R}_o} \sigma_{o_r^+}^{aggr} \tag{3.3}$$

If $d_o \in \mathcal{A}$, this equation is trivially reduced to $\sigma_o^{\text{aggr}} = \sigma_o$, as in such a case o has no successors. For any request $r \in \mathcal{R}$, the cumulative popularity distribution function $\Phi_{p_r,b_r}(\cdot)$ and the aggregated cache size can then be used to calculate the percentage of requests answered by every $o \in \Pi_r$:

$$\Phi_{p_r,b_r}(\sigma_o^{\text{aggr}}) - \Phi_{p_r,b_r}\left(\sigma_{o_r^+}^{\text{aggr}}\right) \tag{3.4}$$

This allows us to calculate the total processing cost as follows:

$$\Psi = \sum_{o \in \mathcal{O}} \sum_{r \in \mathcal{R}_o} \left(\Phi_{p_r,b_r}(\sigma_o^{\text{aggr}}) - \Phi_{p_r,b_r}\left(\sigma_{o_r^+}^{\text{aggr}}\right)\right) \times b_r \times \varphi_{d_o} \times \rho_r \tag{3.5}$$

And finally the transmission cost:

$$\Theta = \sum_{o \in \mathcal{O}} \sum_{r \in \mathcal{R}_o} \sum_{t \in \pi_{o,r}} \left(\Phi_{p_r,b_r} \left(\sigma_o^{\text{aggr}} \right) - \Phi_{p_r,b_r} \left(\sigma_{o_r^+}^{\text{aggr}} \right) \right) \times b_r \times \theta_{c_{t,r}} \times \rho_r \quad (3.6)$$

In summary, the goal of the content delivery federation problem is to minimize $\Delta + \Psi + \Theta$, while satisfying the constraints specified in Eq. 3.1.

3.5 End-to-end resource reservation algorithm

This section presents a heuristic to solve the problem described in Section 3.4.2. The sub-problem of finding QoS constrained paths through the Internet core is equivalent to the MCOP problem [9]. This has been shown to be NP-complete [10]. Therefore, the federated content delivery problem presented in this chapter is also NP-complete. We propose a heuristic that starts from a sub-optimal feasible solution and iteratively improves it. The initial solution consists of the minimum cost QoS-constrained Internet core paths directly from the content provider to the access ISP. Subsequently, storage sites are iteratively included in the end-to-end paths and cache sharing opportunities are identified in order to further reduce the total delivery cost. Figure 3.4 depicts the steps of the devised algorithm in more detail. The sections in which the steps are described are denoted in parentheses. The remainder of this section formally describes the different steps of the heuristic. The final part of this section tackles some deployment considerations that were previously ignored.

3.5.1 Finding QoS-constrained core paths

Both the initial set-up step and the subsequent iterative improvement steps of the presented heuristic rely on finding QoS-constrained minimum cost paths through the Internet core. As such, we consider this sub-problem first before focusing on the actual steps of the heuristic. As stated, this problem is equivalent to the MCOP problem, which finds the shortest path in a graph subject to one or more constraints. Several optimal algorithms and sub-optimal heuristics have been proposed in literature to solve this problem [25, 26]. In our implementation we chose the *A*Dijkstra* variant of the *A*Prune* algorithm [13]. It is capable of both finding the optimal solution (in exponential time) or an approximation (in polynomial time). Nevertheless, any other algorithm that solves the MCOP problem could be used instead.

To find the minimum cost core Internet path $\pi_{o,r}$ for a content cache $o \in \mathcal{O}$ and a request $r \in \mathcal{R}$, the MCOP algorithm takes as input a source domain $t_{d_o} \in \mathcal{T}$ (i.e., the gateway of the domain in which o is deployed), a target domain $a_r \in \mathcal{A}$, a QoS constraints $\gamma_{r,q}$ for every $q \in \mathcal{Q}$ and a graph G representing the network

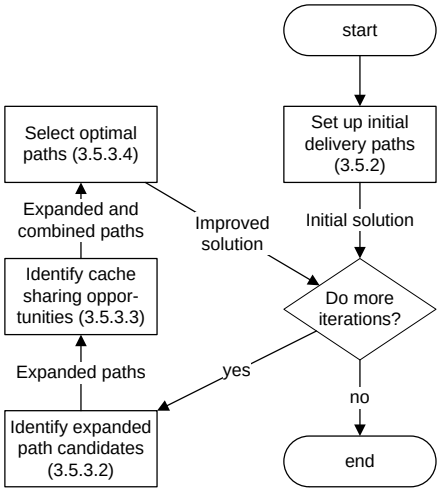


Figure 3.4: Flowchart depicting the steps and flow of the resource reservation algorithm; The sections in which the steps are described are denoted in parentheses

topology. The set of vertices of G equals the set of ISPs $\mathcal{I}$. Every $t \in \mathcal{T}$ has one outgoing edge for every QoS class $c \in \mathcal{C}_t$ to all of its neighbours $n \in \mathcal{I}_t$. Every $a \in \mathcal{A}$ thus only has incoming edges. The cost of an edge equals the cost θ_c of the associated QoS class c, while the weights equal the QoS values $\gamma_{c,q}$ of c for all QoS parameters $q \in \mathcal{Q}$. As a solution, the MCOP algorithm returns a path $\pi_{o,r}$ of vertices and edges. This path is guaranteed to satisfy the constraints specified in Eq. 3.1 for r. If the algorithm is optimal, the path is also guaranteed to have the minimum aggregated cost of all feasible paths. As every edge in G is associated with exactly one QoS class $c \in \mathcal{C}$, the reserved QoS classes $c_{t,r}$ for every $t \in \pi_{o,r}$ can be unambiguously derived from the edges selected by the algorithm.

3.5.2 Setting up the initial delivery paths

As the federated content delivery problem is NP-complete, it is impractical to expect to find the optimal solution within a feasible time-frame. Therefore, we propose a heuristic that iteratively improves its solution. This section describes the first iteration, while the subsequent iterative improvement process is discussed in the next section. To be able to quickly set-up content delivery federations, this initial step has a considerably lower computational complexity than the improvement steps. As a trade-off, the initial solution is less cost-efficient. However, the initial federation can be set-up using the algorithm's initial solution and refinements can be done over time as better solutions are discovered.

For every $r \in \mathcal{R}$, the initial iteration creates a trivial end-to-end path $\Pi_r =$

$\langle o_{p_r}, o_{a_r} \rangle$, consisting of the content caches deployed in the content provider p_r and access ISP a_r domains associated with r. Using the method described above, the minimum-cost QoS constrained core Internet path $\pi_{o_{p_r},r}$ from p_r to a_r is then calculated. In this initial solution, all content is thus requested directly from the content provider.

3.5.3 Iteratively improving the delivery paths

The initial solution calculated above does not include any storage sites. Including storage sites, and deploying dynamic content caches within them, can nevertheless significantly reduce transmission costs. The improvement process expands existing end-to-end paths by adding intermediary content caches. As previously stated, cache sharing allows multiple access ISPs to benefit from a single cache, while sharing the storage cost among them. This allows the total delivery cost of the solution to be further reduced. As such, the improvement process additionally identifies cache sharing opportunities and deploys shared caches when appropriate. Note that cache sharing can only be done among requests that share the same content provider $p \in \mathcal{P}$ and bit-rate $b \in \mathcal{B}_p$. As such, the remainder of this section assumes all requests are directed at the same content provider p and request the same bit-rate b. The steps can then be repeated in the same fashion for other content providers and bit-rates to solve the entire problem.

Iteration (i) takes as input an end-to-end path $\Pi_r^{(i-1)}$ for every $r \in \mathcal{R}$ calculated during iteration $(i-1)$. The goal is to find an end-to-end path $\Pi_r^{(i)}$ for every $r \in \mathcal{R}$, while satisfying the following inequality:

$$\Delta^{(i)} + \Psi^{(i)} + \Theta^{(i)} \leq \Delta^{(i-1)} + \Psi^{(i-1)} + \Theta^{(i-1)} \tag{3.7}$$

In other words, the total cost of the solution found in iteration (i) should be less than or equal to the total cost of the solution found in iteration $(i-1)$. The improvement process consists of several steps. First, for every path, the set of storage sites that could potentially reduce its total cost, is identified. Second, cache sharing opportunities are identified. Third, from all the candidates, the optimal set of delivery paths is selected. In order to calculate the optimal cost of an end-to-end path, we need to be able to determine the optimal cache size of all caches along the path. As such, the remainder of this section first presents a method for determining the optimal cache size. Subsequently, the three sub-steps are discussed in more detail.

3.5.3.1 Determining optimal cache sizes

In this section we present a method for calculating the optimal size of the content caches in an end-to-end path Π_r. As caches can be shared, the optimal cache sizes should be calculated simultaneously for all paths that share at least one cache.

The presented method thus calculates the optimal cache sizes, given a set of overlapping end-to-end paths $\{\Pi_r\}_{r\in\mathcal{R}}$. The problem can be formulated as a Linear Programming (LP) problem, which can be solved with standard LP solving algorithms such as the simplex method [27]. An LP formulation consists of decision variables, constraints and an objective function. The decision variables represent the unknowns, which in this case are the cache sizes. As such, a decision variable $\sigma_o \in [0, \chi_p]$ is associated with every content cache $o \in \{\Pi_r\}_{r\in\mathcal{R}}$. As the cache sizes of content providers and access ISPs are assumed to be static, those decision variables are set to a predefined constant value. The objective function minimizes the total cost (i.e., $\Delta + \Psi + \Theta$). As cache size is an integer variable, the problem becomes an Integer Linear Program (ILP). However, this makes solving it NP-complete. We therefore allow the cache size decision variables to take on any floating point value within the specified range. This relaxed LP problem can be solved in polynomial time. Subsequently, the calculated floating point cache sizes are rounded to the nearest integer value.

3.5.3.2 Identifying expanded path candidates

As a first step in the improvement process, the paths of the previous iteration are expanded with new storage sites. An end-to-end path Π_r of a request $r \in \mathcal{R}$ takes the form $\langle o_1, ..., o_n\rangle$, with o_1 the cache deployed in $p \in \mathcal{P}$ and o_n the cache deployed in a_r. At the start of the expansion process, all paths that share the same o_2 (i.e., the cache deployed in the successor domain of p) are grouped together. This means that all paths that were expanded with a shared cache in the previous iteration, will also be expanded with a shared cache in this iteration. We denote such a group of paths, with the top shared cache $o = o_2$, as $\widehat{\Pi}_o$ and call it a *sharing path set*. During iteration (i), the set of shared path sets $\{\widehat{\Pi}_o\}^{(i-1)}$ constructed during iteration $(i-1)$ is expanded with every $s \in \mathcal{S}$. An expanded shared path set is created by inserting a new shared cache o', deployed in s (i.e., $d_{o'} = s$), right after the root cache into all paths of the existing shared path set. The set $\{\widehat{\Pi}\}^{\text{cnd}}$ contains all the candidate shared path sets that are created during iteration (i). It is initialized to contain all path sets in $\{\widehat{\Pi}_o\}^{(i-1)}$. As described below, new candidate path sets are then iteratively added to it.

From here on, let us consider a single shared path set $\widehat{\Pi}_o \in \{\widehat{\Pi}_o\}^{(i-1)}$ and explain the expansion process for it. The process can subsequently be repeated for all other shared path sets. Concretely, the set is expanded with all $s \in \mathcal{S}$, that are not yet part of any path in the set. This means that an end-to-end path can never contain two different content caches deployed within the same storage site. The expansion of a path $\Pi = \langle o_1, o, ..., o_n\rangle \in \widehat{\Pi}_o$ with a content cache o', results in a path $\Pi' = \langle o_1, o', o, ..., o_n\rangle$. Performing this expansion for all paths in the set results in the new path set $\widehat{\Pi}_{o'}$. Subsequently, the cost-minimizing core Internet paths (cf., Section 3.5.1) and cache sizes (cf. Section 3.5.3.1) can be calculated for

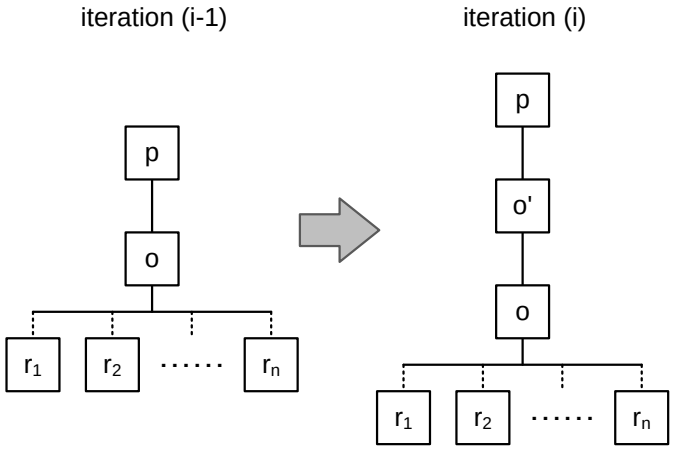


Figure 3.5: A graphical example of how expanded path set candidates are created; The path set $\widehat{\Pi}_o$ is transformed into a new path set $\widehat{\Pi}_{o'}$; The dotted lines represent paths containing zero or more content caches

this new set. This, in turn, allows the total delivery cost of the set to be calculated (cf. Section 3.4.2). If the total delivery cost of $\widehat{\Pi}_{o'}$ is lower than that of $\widehat{\Pi}_o$, then this set becomes the optimal candidate set to replace $\widehat{\Pi}_o$. If the total delivery cost of $\widehat{\Pi}_{o'}$, minus the storage cost of o' is lower than that of $\widehat{\Pi}_o$, then the new set could potentially improve delivery costs if o' is shared across multiple shared path sets. If neither of these conditions is met, the new set cannot lower costs compared to iteration $(i-1)$ and it is discarded. In the former two cases, $\widehat{\Pi}_{o'}$ is added to the set $\{\widehat{\Pi}\}^{\text{cnd}}$ of candidate path sets for iteration (i).

Figure 3.5 further clarifies the creation of expanded path set candidates, using a graphical example. On the left is a shared path set $\widehat{\Pi}_o$, which contains a path for the requests $\{r_1, ..., r_n\} \in \mathcal{R}$ and is represented as by tree. On the right is the same shared path set, after it has been extended with a new cache o'.

3.5.3.3 Identifying cache sharing opportunities

The end of the previous step results in a set of candidate expanded shared path sets $\{\widehat{\Pi}\}^{\text{cnd}}$. However, some of the candidate shared path sets could be combined into a single set, as their paths have the same storage site $s \in \mathcal{S}$ as a successor of $p \in \mathcal{P}$. The goal of this step is to identify all such compatible path sets, and create new path sets that combine them. Let us consider $\{\widehat{\Pi}\}_s \subseteq \{\widehat{\Pi}\}^{\text{cnd}}$ that contains all $\widehat{\Pi}_o \in \{\widehat{\Pi}\}^{\text{cnd}}$ for which $d_o = s$, with $s \in \mathcal{S}$ and $o \in \mathcal{O}$. As every path $\Pi \in \{\widehat{\Pi}\}_s$ shares s as the successor of p, they can be combined into a single shared path set. To achieve this, a new content cache $o' \in \mathcal{O}$, with $d_{o'} = s$, is constructed. Subsequently, every path $\Pi = \langle o_1, o, o_2, ..., o_n \rangle \in \{\widehat{\Pi}\}_s$, with

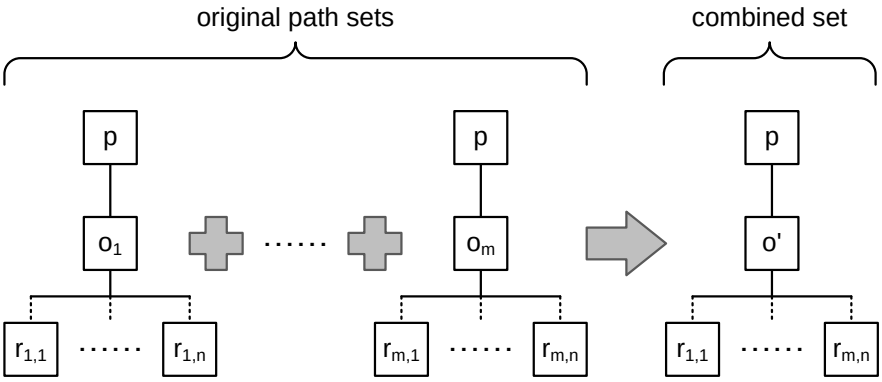


Figure 3.6: A graphical example of how a group of path sets can be combined into a single path set with a shared cache; The original set of path sets $\{\widehat{\Pi}\}_s$ is transformed into a new combined path set $\widehat{\Pi}_{o'}$; The dotted lines represent paths containing zero or more content caches

$d_o = s$, is transformed into $\Pi' = \langle o_1, o', o_2, ..., o_n \rangle$ and added to the new shared path set $\widehat{\Pi}_{o'}$. Subsequently, $\widehat{\Pi}_{o'}$ is added to the set $\{\widehat{\Pi}\}^{\text{cnd}}$ of candidate path sets. This process can then be repeated for all $s \in \mathcal{S}$ in order to identify all combined shared path sets.

Figure 3.6 further clarifies the process of combining multiple shared path sets into a new combined path set with a shared cache. It shows a group of m shared path sets $\{\widehat{\Pi}\}_s = \{\widehat{\Pi}_{o_1}, ..., \widehat{\Pi}_{o_m}\}$, with $d_{o_1} = ... = d_{o_m} = s$. They are combined into a new shared path set $\widehat{\Pi}_{o'}$, where also $d_{o'} = s$.

The set $\{\widehat{\Pi}\}^{\text{cnd}}$ now contains all fully combined shared path sets, that combine *all* path sets with the same s as direct successor of p. However, the partially combined shared path sets that only contain a subset of the shared path sets with the same s can also be added to $\{\widehat{\Pi}\}^{\text{cnd}}$. This is done as follows. Consider the set $\{\widehat{\Pi}\}^{\text{cnd}}_r \subseteq \{\widehat{\Pi}\}^{\text{cnd}}$ that contains all the shared path sets $\widehat{\Pi}$ that contain an end-to-end path Π_r for request $r \in \mathcal{R}$. Such a set is either a direct expansion of a shared path set of iteration $(i-1)$ (cf. Section 3.5.3.2) or a combination of several such expanded sets (cf. Section 3.5.3.3). In the former case, no new paths are derived from it. In the latter case, the shared path set $\widehat{\Pi}$ consists of the union of several other, non-combined, shared path sets $\{\widehat{\Pi}_1, ..., \widehat{\Pi}_n\}$. A new set $\widehat{\Pi}'$, which combines the same non-combined paths sets $\{\widehat{\Pi}_1, ..., \widehat{\Pi}_n\}$ except for the one containing Π_r, is then constructed. Subsequently, $\widehat{\Pi}'$ is also added to $\{\widehat{\Pi}\}^{\text{cnd}}$.

The total number of candidate path sets in $\{\widehat{\Pi}\}^{\text{cnd}}$ can grow exponentially with the number of requests and storage sites. In order to reduce complexity of the problem, a subset of candidate paths can be filtered out. The algorithm uses a

method that retains K path sets for every $r \in \mathcal{R}$. As the same path set can be retained for multiple requests, the total remaining number of path sets is therefore less than or equal to $|\mathcal{R}| \times K$. For every request r, the K path sets that contain a path Π_r with the lowest delivery cost for Π_r are retained.

3.5.3.4 Selecting optimal paths

The first two steps of the iterative improvement process generate a set of shared path sets $\{\widehat{\Pi}\}^{\text{cnd}}$. It potentially contains many path sets $\widehat{\Pi}$ that contains a path Π_r for request $r \in \mathcal{R}$. However, the final solution returned by iteration (i) should contain exactly one end-to-end path Π_r for every request r. The final step of the improvement process thus selects the optimal combination of path sets, such that exactly one path is selected for every request. This is an optimization problem, that can be formulated as an ILP formulation. The formulation consists of a boolean decision variable $p_{\widehat{\Pi}} \in \{0, 1\}$ for every $\widehat{\Pi} \in \{\widehat{\Pi}\}^{\text{cnd}}$. This variable denotes whether or not the associated path set will be selected for the final solution. There is a single additional constraint, stipulating that for every request $r \in \mathcal{R}$ only one path set can be selected that contains a path Π_r for r:

$$\forall r \in \mathcal{R} : \sum_{\widehat{\Pi} \in \{\widehat{\Pi}\}^{\text{cnd}}_r} p_{\widehat{\Pi}} = 1 \tag{3.8}$$

The objective function is once again to minimize the total delivery cost of the solution. Let us define $\Delta(\widehat{\Pi})$, $\Psi(\widehat{\Pi})$ and $\Theta(\widehat{\Pi})$ as the total storage, processing and transmission costs of shared path set $\widehat{\Pi}$. The objective can then be formulated as follows:

$$\min \sum_{\widehat{\Pi} \in \{\widehat{\Pi}\}^{\text{cnd}}} p_{\widehat{\Pi}} \times \left(\Delta\left(\widehat{\Pi}\right) + \Psi\left(\widehat{\Pi}\right) + \Theta\left(\widehat{\Pi}\right) \right) \tag{3.9}$$

To calculate the value of the above objective function, the delivery cost of every $\widehat{\Pi} \in \{\widehat{\Pi}\}^{\text{cnd}}$ needs to be known. This is a costly operation, as for every $\widehat{\Pi}$ the LP formulation described in Section 3.5.3.1 needs to be solved. However, the cost of the non-combined expanded path sets (as calculated in Section 3.5.3.2) is known, as it needs to be calculated to determine whether or not the path set is a valid candidate. The number of paths in this set is much smaller and limited by $|S| \times |R|$. To avoid having to calculate the cost of all candidate shared path sets, we propose a method to estimate the total delivery cost of $\widehat{\Pi}$, based on the cost of the non-combined path sets of which it is composed. Every combined shared path set $\widehat{\Pi}_o$ equals the union of multiple non-combined shared path sets $\{\widehat{\Pi}_{o_1}, ..., \widehat{\Pi}_{o_n}\}$. The total delivery cost of $\widehat{\Pi}_o$ can then be estimated as follows:

$$\sum_{\widehat{\Pi}_{o_i} \in \{\widehat{\Pi}_{o_1}, ..., \widehat{\Pi}_{o_n}\}} \Delta\left(\widehat{\Pi}_{o_i}\right) + \Psi\left(\widehat{\Pi}_{o_i}\right) + \Theta\left(\widehat{\Pi}_{o_i}\right) - \frac{(n-1)}{n} \times \Delta\left(o_i\right) \tag{3.10}$$

With $\Delta\left(o_i\right)$ the storage cost associated with content cache $o_i \in \mathcal{O}$.

3.5.4 Additional considerations

3.5.4.1 Scalable video coding

Throughout this section it was assumed that the different bit-rate versions of the same multimedia content item are unrelated. As such, the calculation of end-to-end delivery paths can be done separately for each bit-rate, without taking other bit-rates into account. This is a valid assumption for traditionally encoded content, such as AVC (Advanced Video Coding) video. Recently, Scalable Video Coding (SVC) has garnered a lot of attention [28]. It allows video content to be encoded as a set of layers, that can be combined to increase quality. The base layer can thus be decoded separately, resulting in a low quality video. If, however, it is decoded together with one or more enhancement layers, video quality increases. Obviously, when content is encoded using SVC the assumption that different bit-rate versions of the same content item are unrelated no longer holds.

Our presented model and algorithm can be easily adapted to support SVC encoded content as follows. When an access ISP requests a new QoS-aware end-to-end delivery path, it no longer creates a single request $r \in \mathcal{R}$. Instead, it creates a set of requests, one for each layer of the SVC encoded content it wants to receive. For example, assume a content provider offers its content catalogue encoded using SVC in three layers with bit-rates b_1, b_2 and b_3. If an access ISP wants to set up a delivery path for the lowest quality version, it sends the content provider a request r with bit-rate $b_r = b_1$. If however it wants to request the highest quality, it sends three requests, r_1, r_2 and r_3. Their bit-rates are respectively $b_{r_1} = b_1$, $b_{r_2} = b_2$ and $b_{r_2} = b_2$. The advantage of SVC-based content is that different quality requests now partially overlap, resulting in more cache sharing opportunities and thus decreased delivery costs.

3.5.4.2 Segment-based content delivery

The presented model and algorithm assumes the delivered content items to be monolithic units of data. However, it has been shown in practice that multimedia content often has a high internal popularity skew [29]. For example, in many video-based multimedia services, the beginning of videos is often much more popular than the end. As such, it has been shown that splitting content into temporal segments can significantly increase delivery and caching performance [30]. Although the presented model is based on content as the unit of data, it can be adapted to support temporal content segments. The only required change is in the statistical information provided by the content provider. Specifically, the weighted average content duration should be calculated on a per-segment basis, instead of per-content item, the number of items in the catalogue should be replaced with the total number of segments, and the popularity distribution function should model the segment popularity instead of entire content item popularity.

3.5.4.3 Input parameter estimation

In order to calculate the delivery costs, the presented algorithm expects several statistical input parameters, both related to the content provider and the customers. More specifically, the algorithm requires information about: average content duration, average simultaneously delivered content items and the cumulative popularity distribution. The average content duration can easily be derived from the content catalogue. On the other hand, the other two parameters are dynamic over time. The algorithm needs accurate dynamic predictions for these parameter, in order to be applicable in actual deployments. Note that the first step of the algorithm (cf. Section 3.5.2), which sets up direct core Internet paths between the content provider and access ISP, does not require this information to calculate the cheapest solution. Therefore, the content provider can first set up direct delivery paths. As time passes, it will be able to more accurately estimate values for these parameters, allowing it to iteratively improve the solution using the subsequent steps of the algorithm.

The modelling and prediction of popularity distribution curves of multimedia services has been an active research topic for several years [31, 32]. Existing methods can thus be applied to estimate the cumulative popularity distribution. More recently, some methods have been proposed to predict the popularity of individual content items [33, 34]. This can be applied to predict the number of simultaneous requests originating from an access ISP. Alternatively, information from the recent past can be used as a fast estimation of these parameters in the near future.

3.6 Results and discussion

This section evaluates the heuristic presented in Section 3.5. First, the scalability is evaluated. Second, the merits of our approach are validated under a variety of conditions. More specifically, the usefulness of dynamically deployed intermediary caches and cache sharing is evaluated, as these are the novel aspects of our approach compared to existing end-to-end content delivery mechanisms. Additionally, we determine the impact on caching efficiency of several parameters, such as the location of access ISPs and storage sites and the storage cost. The presented results were obtained from a Java-based implementation of the algorithm. The LP optimization problems were solved using the Java version of CPLEX 12.3[1]. All tests were performed on a computer with one Dual-Core AMD Opteron 2212 processor and 4 GiB RAM memory, running the GNU/Linux Debian 6.0 operating system. All depicted results are averaged over 100 iterations, with the error bars showing the standard error of the mean. The algorithm was run for two rounds. First, the initial delivery paths were calculated. Second, a single iterative improve-

[1] http://www.ibm.com/software/integration/optimization/cplex-optimizer/

ment step was performed. This means that the final end-to-end delivery paths consist of at most one intermediary storage site. The parameter K (cf. Section 3.5.3.3) is set to 100 throughout the evaluation.

3.6.1 Evaluation scenario

The core Internet topology used throughout the evaluations, was generated using the ReaSE topology generator [35]. It consists of 250 ASes, including 45 transit domains and 205 stub domains. The ReaSE parameters P and Δ were left at the default values 0.4 and 0.04. The total diameter of the generated core network is 4 hops. Every generated AS corresponds to a single transit ISP $t \in \mathcal{T}$. As the algorithm calculates delivery paths for every content provider separately, we consider only a single content provider $p \in \mathcal{P}$ without loss of generality. The transit ISP gateway t_p of p is randomly selected from the 205 stub transit ISPs. Subsequently, the set of access ISPs $\mathcal{A}$ is created. The number of access ISPs is set to 10, unless stated otherwise. One of the goals of this evaluation is to assess the effect of the position of access ISPs within the network on the merits of cache sharing. As such, we define an *access ISP vicinity* parameter $av \in [0, 1]$. This parameter defines the probability that access ISPs are positioned near each other in the network. Concretely, the gateway of every access ISP is determined as follows. First, a random gateway t_a is selected for the first access ISP $a \in \mathcal{A}$ from the set of stub transit ISPs. The gateway for all other access ISPs $a_i \in \mathcal{A}$ is selected as follows. With a probability $av_1 = av$, a random stub transit ISP that is exactly one hop away from t_a is selected as the gateway of a_i. The probability that a stub transit ISP n hops away is selected, is calculated as follows:

$$av_n = \left(1 - \sum_{i=1}^{n-1} av_i\right) \times av \tag{3.11}$$

If the maximum hop distance from t_a equals m, then $av_m = (1 - \sum_{i=1}^{m-1} av_i)$ to make sure a gateway is definitely selected. As an example, if $av = 0.7$ and $m = 4$, then the access ISP vicinity probabilities become 0.7, 0.21, 0.063 and 0.027. Throughout the rest of this section $av = 0.7$ unless otherwise stated. Finally, the storage sites $\mathcal{S}$ are added to the network. Unless stated differently, 10 storage sites are used. We now define the parameter $sv \in [0, 1]$ as the *storage site vicinity*. The gateway t_s for every storage site $s \in \mathcal{S}$ is selected as follows. First, an access ISP $a \in \mathcal{A}$ is selected in round robin fashion. With a probability of $sv_0 = sv$ the gateway t_a of a is selected as the gateway t_s of s. The probability that a gateway n hops away from t_a is selected is then calculated as follows:

$$sv_n = \left(1 - \sum_{i=0}^{n-1} sv_i\right) \times sv \tag{3.12}$$

Again, if the maximum hop distance from t_a equals m, then $sv_m = (1-\sum_{i=0}^{m-1} sv_i)$. If the value of sv is chosen close to 1, then most access ISPs will have a storage site nearby in the network. When the value of sv is close to 0, storage sites will be further away. Unless stated differently, $sv = 0.7$ throughout the rest of the evaluation.

The evaluation scenario considers a VoD service, with the content provider's catalogue consisting of 5000 unique movies. The average movie duration is 5400 seconds, or 90 minutes. As the algorithm calculates a separate solution for every bit-rate, we can consider a single bit-rate without loss of generality. The movie bit-rate is set to 5 Mbps. In literature, several models have been presented for representing the cumulative popularity distribution of multimedia services. We use the Zipf-mandelbrot distribution, with $\alpha = 0.8$ and $q = 1$ [36].

Every transit ISP is characterised by a set of QoS classes $\mathcal{C}$. As stated, the presented model supports an arbitrary number of QoS parameters. However, as a TCP-based progressive download scenario is assumed, we do not need to consider packet loss and focus the *delay* and *availability* QoS parameters in this evaluation. Delay is defined as the one-way transmission latency, while availability denotes the probability that the associated path through the network is available for use. The transit ISPs support three delay values: 0.001, 0.005 and 0.01 seconds. Additionally, two availability values are supported: 0.999 and 0.9999. They respectively result in an average yearly downtime of 8.76 hours and 52.56 minutes. The reservation cost of a QoS class $c \in \mathcal{C}$ is calculated as follows:

$$\theta_c = \frac{0.00001}{\text{delay}_c \times (1 - \text{availability}_c)} \tag{3.13}$$

This reservation cost is paid for every MBps (megabyte per second) that is transferred through the associated ISP domain. In total, every transit ISP offers all 6 combinations of these QoS parameter values, as depicted in Table 3.3. The cheapest and most expensive QoS classes thus result in a cost of 0.625 and 62.5 per ISP domain per content stream per second. The storage cost λ_s is the same for all storage sites $s \in \mathcal{S}$. As the effect of this cost is studied in this section, it is a variable parameter sc. Its default value is 0.0001, which gives a storage cost per movie of 0.3375 per second. To be able to more accurately study the synergy between the storage and transmission costs, the processing cost is assumed to be negligible throughout this evaluation (i.e., $\varphi_s = 0$ for all $s \in \mathcal{S}$). Caches can also be deployed within the access ISP domain, without an additional storage cost. Such caches are only considered when specifically stated. Otherwise, it is assumed $\sigma_a = 0$ for all $a \in \mathcal{A}$.

Finally, the scenario contains a set of requests $\mathcal{R}$ for setting up QoS-constrained end-to-end delivery paths. A single request $r \in \mathcal{R}$ is generated for every access ISP $a \in \mathcal{A}$. For every request, the average number of simultaneous content streams

Table 3.3: The QoS classes used in the evaluation scenario

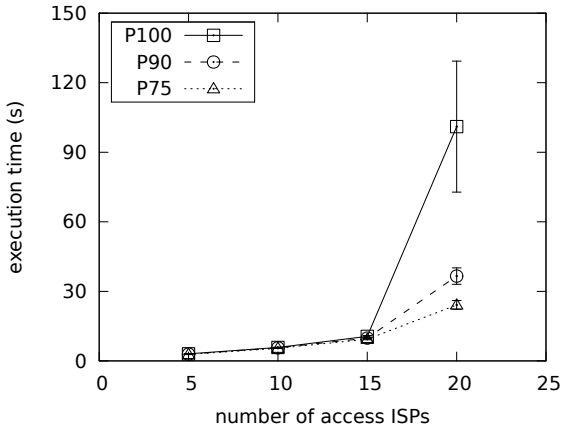
delay	availability	θ
0.01	0.999	1
0.005	0.999	2
0.01	0.9999	10
0.001	0.999	10
0.005	0.9999	20
0.001	0.9999	100

is determined uniformly at random from the range $[50, 250]$. The requested end-to-end delay is chosen uniformly at random from the range $[0.005, 0.05]$ seconds, while the availability is selected from $[0.995, 0.9995]$. The least stringent constrains, with delay 0.05 seconds and availability 0.995, can be satisfied on a path of up to 5 transit ISPs long with the cheapest QoS class. On the other hand, on a path of up to 5 transit ISPs, the most stringent constraints (i.e., delay $= 0.005$ and availability $= 0.9995$) can only be achieved when using the most expensive QoS class in all domains along the path.

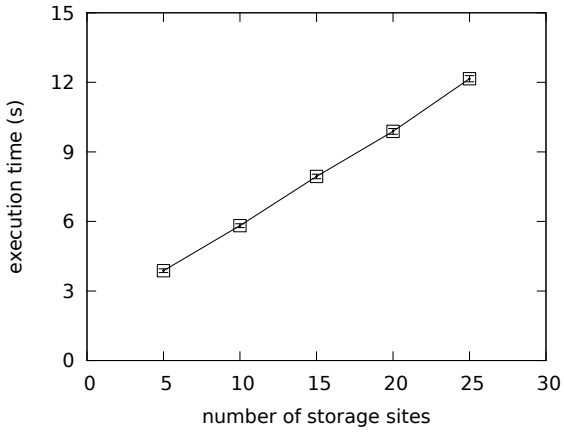
3.6.2 Scalability

In this section, the algorithm's computational performance and scalability are evaluated, in terms of execution time. Its computational complexity is influenced by the number of access ISPs and storage sites within the network. Figure 3.7 depicts the execution time as a function of the number of access ISPs and storage sites. The depicted execution time is for the first iterative improvement round of the algorithm. The execution time of the initial set up process was always less than 1 second and is therefore not depicted. This allows the algorithm to be applied to highly dynamic situations, as the initial solution can be swiftly calculated to set up the initial direct core Internet paths between the content provider and access ISPs. Over time the slower improvement steps of the algorithm can then be employed to iteratively add storage sites.

Increasing the number of access ISPs complicates the end-to-end content delivery problem in several ways. First, the number of decision variables in the LP problem to determine the optimal cache sizes increases linearly. Additionally, the number of such problems that need to be solved increases linearly as well. As pure LP formulations can be solved in polynomial time [37], this results in a (non-linear) polynomial increase in execution time. The number of core Internet paths that need to be calculated also increases linearly. The complexity increase of this step depends on the complexity of the MCOP algorithm. However, as many polynomial time heuristics exist to solve the MCOP problem, this also results in a polynomial increase in execution time. On the other hand, the number of expanded



(a) as a function of access ISP count



(b) as a function of storage site count

Figure 3.7: Execution time of the algorithm

path candidates that exist increases exponentially. Additionally, as the LP problem to determine the set of cost minimizing expanded paths contains integer variables, solving it takes exponential time in the worst case. The exponential increase in the amount of candidate paths can be countered by using the parameter K to limit the subset of considered candidate expanded paths (cf. Section 3.5.3.3). This reduces the increase to a linear one, and in turn limits the complexity of the ILP problem to select the optimal candidates. Nevertheless, the worst case time complexity of the ILP formulation remains exponential. Figure 3.7a depicts the execution time (in seconds) as a function of the number of access ISPs. As expected from the analysis

above, the scaling behaviour is worse than linear. At 20 access ISPs the execution time explodes. However, the standard deviation also grows significantly, suggesting a high variability in the results. Closer inspection shows that this is indeed the case. The execution time of 65 of the 100 runs was below 50 seconds, while only 16 runs showed an execution time above 100 seconds of which 4 were above 300 seconds. The two slowest runs took 1583 and 1845 seconds respectively. To further illustrate this, the graph also shows the $P90$ and $P75$ curves, which are averages over the 90 and 75% best values respectively. These two curves depict a much lower standard error and significantly reduced average execution time for 20 access ISPs (i.e., 36 and 24 seconds respectively). This high variability shows that the ILP formulation can often be solved within a feasible time frame. However, on some rare occasions it takes exponential time to find the optimal solution. This can be prevented by configuring the ILP solver with a maximum calculation time. This will solve the execution time variability, but will in rare cases cause the solver to only find a suboptimal solution.

The number of storage sites also affects the problem complexity. It also causes the number of candidate expanded trees to grow exponentially. However, by selecting the subset of K most promising candidates, the number of candidates considered in the ILP formulation remains constant. Additionally, in contrast to the number of access ISPs, the complexity of the LP formulation is not increased. Instead only the number of times it needs to be solved grows. This is also reflected in the results in Figure 3.7b, which depicts the execution time (in seconds) as a function of the number of storage sites. In line with the above analysis, the algorithm scales linearly with the number of storage sites.

3.6.3 Storage site merits

An important novel aspect of the presented approach is the inclusion of cloud-based storage sites within the end-to-end federations. We intuitively expect this to reduce the transmission costs associated with the delivery of multimedia content. Nevertheless, the conditions under which this is the case are unclear. This section assesses the effect of several input parameters on the merits of intermediary storage sites. More specifically, it is expected both the storage cost sc and storage site vicinity sv will affect the usefulness of storage sites. Figure 3.8 compares the delivery cost of the solution with and without intermediary storage sites as a function of the storage cost sc. Subsequently, Figure 3.9 shows how the vicinity of storage sites to the access ISPs (i.e., sv) affects the total delivery cost and caching efficiency.

Intuitively, it is expected intermediary storage sites are only useful if the storage cost is below a certain threshold relative to the transmission cost. In order to determine this threshold, Figure 3.8 compares the total delivery cost to all access

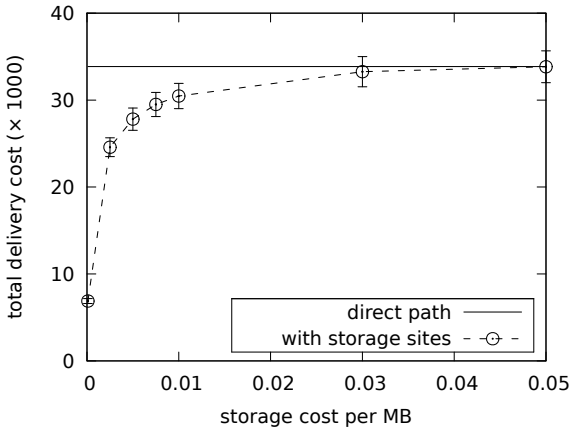


Figure 3.8: Influence of the storage cost sc on the merits of intermediary content caches; comparing end-to-end QoS-aware content delivery with and without intermediary storage sites

ISPs with and without intermediary content caches. As end-to-end QoS negotiation mechanisms traditionally reserve a path through the Internet core directly from the content provider to its customers, the solution without storage sites is comparable to those traditional methods. The delivery cost of the solution without intermediary storage sites is independent of the storage cost. This results in a constant total cost value of 33860 in the evaluated scenario. As our algorithm starts from this solution as well and then iteratively improves it, it can never perform worse. Additionally, the graph shows that it performs significantly better when the storage cost is less than 0.03 per MB (megabyte), or on average 101.25 per movie. Additionally, it can be calculated that the average transmission cost without an intermediary cache is on average about 20 per movie, in the evaluated scenarios. These results therefore show that deploying intermediary content caches can significantly reduce delivery costs, even when the average cost for storing a content items is several times higher than the cost for transmitting one. If the storage cost is 0.01 per MB, the total cost is about 10% better than that of the solution without storage sites. When the storage cost becomes very small (i.e., 0.0001 per stored MB or 0.3375 per movie), the total delivery cost can be reduced down to on average 6883 in the evaluated scenarios. This is a reduction of 80% compared to the traditional QoS negotiation approaches that only employ direct end-to-end paths.

If storage sites are located closer to the access ISPs, content will on average need to traverse a shorter path through the Internet core. This increases the effectiveness of intermediary content caches and is thus expected to reduce transmission costs. Figure 3.9 compares the total delivery cost of the solution with and without

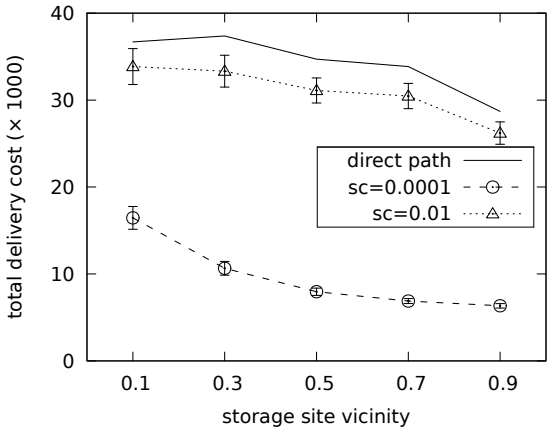


Figure 3.9: Influence of the storage site vicinity sv on the merits of intermediary content caches; comparing end-to-end QoS-aware content delivery with and without intermediary storage sites

including intermediary content caches. As expected, positioning the storage sites closer to the access ISPs (i.e., increasing the value of sv) significantly reduces the total delivery costs. If storage providers are positioned far away from the access ISPs and the storage cost is not very low (i.e., $sc = 0.01$), then using storage sites cannot reduce delivery costs as compared to using the direct path from content provider to access ISP. However, if $sv = 0.5$, which gives an 87.5% chance that a storage site is within three hops of every access ISP, our algorithm significantly outperforms the traditional direct end-to-end path even if the storage cost is not very low. If the storage cost is very low (i.e., $sc = 0.0001$), the traditional approach is outperformed independent of the value of sv.

In summary, it can be concluded that the use of storage sites for deploying dynamic caches inside the Internet core can indeed outperform the traditional QoS-aware end-to-end delivery approach that directly sends content from the provider to its customers. Nevertheless, the significance of the achieved cost reduction depends on several factors. If the cost for storing a content item becomes very high relative to the cost for transmitting it, caching no longer reduces costs. Additionally, unless the storage cost is negligible compared to the transmission cost, storage sites need to be positioned relatively close to the access ISPs. On the other hand, if the storage cost is insignificant relative to the transmission cost, randomly placed storage sites remain useful in reducing the total delivery cost.

3.6.4 Cache sharing merits

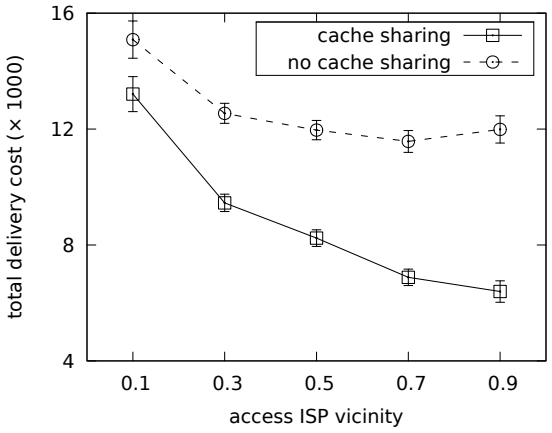
A major advantage of deploying content caches in intermediary domains across the Internet, is the opportunity to share them among several access ISPs. This greatly improves the cache's efficiency, as more content can be served from it without increasing the total storage cost. Nevertheless, the gain that can be achieved through cache sharing is influenced by factors such as the access ISP vicinity av. This section evaluates the effect of av on cache sharing in more detail. As metrics to evaluate cache sharing, the total delivery cost and cache sharing ratio are used. The cache sharing ratio is defined as the average number of access ISPs that share a storage site content cache. It thus takes a value between 1 (i.e., no cache sharing) and $|\mathcal{A}|$ (i.e., the cache is shared by all access ISPs). Figure 3.10 shows the total delivery cost and cache sharing ratio as a function of access ISP vicinity av.

Figure 3.10a compares the total delivery cost, with and without cache sharing, as a function of the access ISP vicinity av. Previously, it was shown that content caches can be more efficiently employed if they are positioned near the access ISPs (cf. Section 3.6.3). As a consequence, we intuitively expect cache sharing to be more efficient if the access ISPs are positioned close together. This is reflected in the results shown in the figure. If the access ISPs are far away from one another (i.e., $av = 0.1$), the solution with cache sharing has only a slightly better delivery cost than the solution without cache sharing (i.e., a 12.5% reduction). However, as the access ISPs are positioned closer together, more cache sharing opportunities become available. If $av = 0.9$, then the total delivery cost of the solution with cache sharing is 46% reduced compared to the solution without cache sharing. This gain is entirely caused by the reduction in storage costs cache sharing introduces. The cache sharing ratio is depicted in Figure 3.10b. This figure confirms our previous findings, showing that more cache sharing occurs as the access ISPs are positioned closer together.

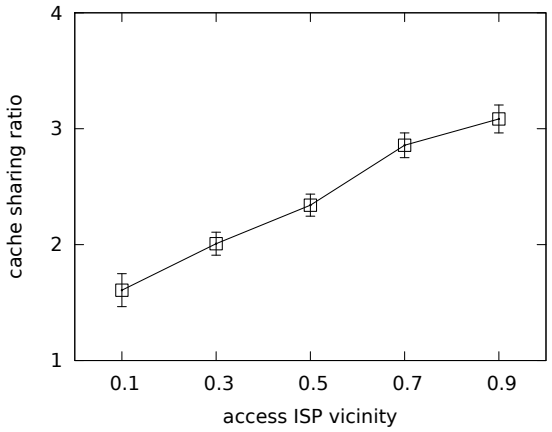
In summary, the presented results prove that deploying content caches in intermediary domains, as opposed to at the client side, does have its advantages. Specifically, it allows those caches to be shared among different access ISPs, reducing the storage costs. Results show that cache sharing is most effective when several access ISPs are positioned close together in the network. For access ISPs further away from each other, cache sharing is not useful, as the shared content cache will be too far away from at least part of the shared access ISPs.

3.6.5 Caching in the access network

In addition to the dynamic content caches deployed across the Internet, the access ISP can also deploy a local cache. This locally cached content does not need to traverse any intermediary transit ISP domains, further reducing the total transmission cost compared to intermediary deployed caches. On the other hand, these locally



(a) effect on total delivery cost

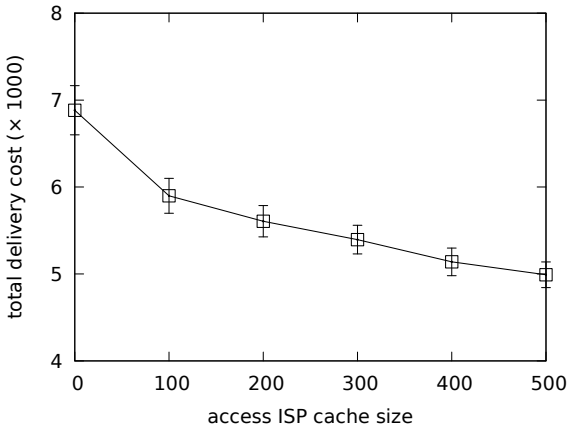


(b) effect on total cache reuse

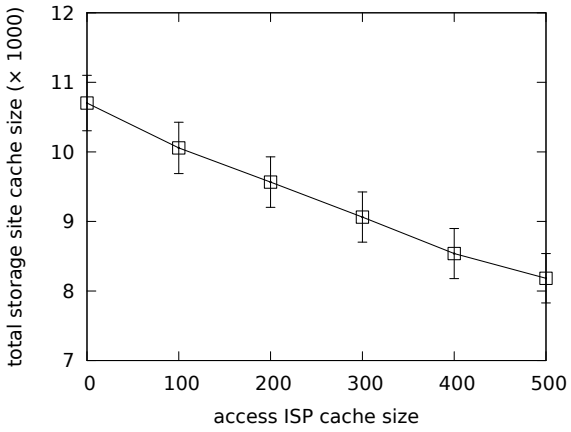
Figure 3.10: Influence of the access ISP vicinity av on the efficiency of cache sharing

deployed caches cannot be shared, potentially reducing their efficiency. Deploying caches within the access domain is known to be expensive [38]. We thus expect the size of such a cache to be relatively small. Figure 3.11b explores the effect of such a small access ISP cache on the total delivery cost and the total size of the storage site caches.

Intuitively, we expect that caching in the access ISP network will reduce the total transmission costs. Additionally, as the intermediary caches will become smaller, the storage cost is also expected to drop. Figure 3.11a depicts the total delivery cost as a function of an increasing access ISP cache size. Note that such a



(a) effect on total delivery cost



(b) effect on storage site cache usage

Figure 3.11: Influence of caching in the access domain on caching efficiency of the storage sites

cache of the same size is deployed in every access domain. As shown in the figure, deploying access caches significantly reduces the total delivery cost. A cache of 500 items in every access domain results in a cost reduction of 27%. However, this is only assuming that no cost is associated with the access caches. As the scenario consists of 10 access ISPs, they have a combined cache size of 5000 items. In this specific scenario, the storage cost is 0.0001. If these access caches had the same cost as the intermediary storage site caches, this would result in a total storage cost of 1687.5. If this is added to the total delivery cost, then the total achieved cost reduction is reduced to a mere 3%, as opposed to 27%. Figure 3.11b

presents the total size of the content caches deployed in the intermediary storage sites. Deploying access caches of 500 items each, reduces the total in-network cache size from 10702 to 8183. The sum of the in-network and access cache sizes is thus higher than the total cache size when only using in-network caches.

In summary, we can conclude that small static caches deployed in the access domain can reduce the total delivery costs. However, this reduction is only significant if there is no cost associated with the caches deployed in the access network.

3.7 Conclusion

This chapter presents a novel framework for setting up end-to-end federations between the stakeholders involved in the delivery of multimedia content. More specifically, it guides the negotiation of SLAs between content providers, ISPs and cloud-based storage sites. This allows them to overcome the disadvantages associated with current delivery approaches, such as OTT content provisioning and content offered directly by access ISPs over a managed IP network. In contrast to existing works, our framework includes storage sites in the end-to-end delivery paths, allowing content caches to be dynamically deployed throughout the network. This introduces an additional complexity to the problem, but allows further optimization of the delivery process. This chapter proposes a detailed mathematical model to optimize the content provider's end-to-end delivery costs. An optimization algorithm is presented for solving the model. It satisfies the customer's requested QoS, while minimizing the total delivery cost. To achieve this, it determines optimal QoS-constrained routes through the Internet core, and identifies well positioned intermediary storage sites for the deployment of content caches. Additionally, the algorithm calculates the amount of resources that need to be reserved along these routes and within the identified storage sites.

The presented framework was thoroughly validated based on evaluation results. The algorithm's scalability was characterized and the merits of our novel approach, that includes intermediary content caches, were quantified. Results show that including intermediary storage sites within the end-to-end delivery paths can significantly reduce delivery costs compared to traditional end-to-end QoS reservation mechanisms that use only direct QoS-constrained paths between the content provider and its customers. The significance of the cost reduction does depend on some external factors, such as the cost for storing an item, as compared to transmitting it and the vicinity in the network of storage sites to the access ISPs. Even if the storage cost per content item is higher than the transmission cost per item, a cost reduction of 10% can be easily achieved. If the storage cost becomes a fraction of the transmission cost, this reduction reaches up to 80% in the evaluated scenario. Additionally, as these content caches are deployed inside the network, they can be shared among different access ISP customers. The results prove that cache shar-

ing significantly decreases the delivery costs for access ISPs that are located near each other. In the evaluated scenario, cache sharing resulted in an additional cost reduction of up to 46%.

Acknowledgment

Jeroen Famaey was partially funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT). Steven Latré and Tim Wauters are funded by the Fund for Scientific Research Flanders (FWO). The research leading to these results was partially performed within the context of the FP7 OCEAN project and received funding from the European Union's Seventh Framework Programme ([FP7/2007-2013]) under grant agreement number 248775.

References

[1] I. Amigo, P. Belzarena, F. Larroca, and S. Vaton. *Network bandwidth allocation with end-to-end QoS constraints and revenue sharing in multi-domain federations*. In Proceedings of the 7th International Conference on Internet Charging and QoS Technologies, pages 50–62, 2011. doi:10.1007/978-3-642-24547-3_6.

[2] M. Serrano, S. van der Meer, V. Holum, J. Murphy, and J. Strassner. *Federation, a matter of autonomic management in the Future Internet*. In Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 845–849, 2010. doi:10.1109/NOMS.2010.5488357.

[3] N. Kumar and G. Saraph. *End-to-end QoS in interdomain routing*. In Proceedings of the 2nd International Conference on Networking and Services, 2006. doi:10.1109/ICNS.2006.45.

[4] H. Xiangjiang, Z. Peidong, C. Kaiyu, and G. Zhenghu. *AS alliance in interdomain routing*. In Proceedings of the 22nd International Conference on Information Networking and Applications – Workshops (AINAW), pages 151–156, 2008. doi:10.1109/WAINA.2008.209.

[5] L. Roberts. *A radical new router*. IEEE Spectrum, 46(7):34–39, 2009. doi:10.1109/MSPEC.2009.5109450.

[6] H. Pouyllau and G. Carofiglio. *Inter-carrier SLA negotiation using Q-learning*. Telecommunication Systems, 2011. doi:10.1007/s11235-011-9505-5.

[7] J. Famaey, S. Latré, T. Wauters, and F. De Turck. *FedRR – a federated resource reservation algorithm for multimedia services*. In Proceedings of the 13th IEEE/IFIP Network Operations and Management Symposium (NOMS), 2012.

[8] J. Famaey, S. Latré, T. Wauters, and F. De Turck. *An SLA-driven framework for dynamic multimedia content delivery federations*. In Proceedings of the Fifth International Workshop on Distributed Autonomous Network Management Systems (DANMS), 2012.

[9] H. Pouyllau and R. Douville. *End-to-end QoS negotiation in network federations*. In Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 173–176, 2010. doi:10.1109/NOMSW.2010.5486578.

[10] T. Korkmaz and M. Krunz. *Multi-constrained optimal path selection*. In Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 834–843, 2001. doi:10.1109/INFCOM.2001.916274.

[11] S. Chen and K. Nahrstedt. *On finding multi-constrained paths*. In Proceedings of the IEEE International Conference on Communications, pages 874–879, 1998. doi:10.1109/ICC.1998.685137.

[12] H. De Neve and P. Van Mieghem. *A multiple quality of service routing algorithm for PNNI*. In Proceedings of the IEEE ATM Workshop, pages 324–328, 1998.

[13] G. Liu and K. Ramakrishnan. *A*Prune: an algorithm for finding K shortest paths subject to multiple constraints*. In Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 743–749, 2001. doi:10.1109/INFCOM.2001.916263.

[14] X. Yuan and X. Liu. *Heuristic algorithms for multi-constrained quality of service routing*. In Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pages 844–853, 2001. doi:10.1109/INFCOM.2001.916275.

[15] J. Xiao and R. Boutaba. *QoS-aware service composition and adaptation in autonomic communication*. IEEE Journal on Selected Areas in Communications, 23(12):2344–2360, 2005. doi:10.1109/JSAC.2005.857212.

[16] J. Yan, R. Kowalczyk, J. Lin, M. B. Chhetri, S. K. Goh, and J. Zhang. *Autonomous service level agreement negotiation for service composition provision*. Future Generation Computer Systems, 23:748–759, 2007. doi:10.1016/j.future.2007.02.004.

[17] S. Balasubramaniam, D. Botvich, R. Carroll, J. Mineraud, T. Nakano, T. Suda, and W. Donnelly. *Biologically inspired future service environment*. Computer Networks, 55(15):3423–3440, 2011. doi:10.1016/j.comnet.2011.07.004.

[18] C. Yuanming, W. Wendong, G. Xiangyang, and Q. Xirong. *Initiator-domain-based SLA negotiation for inter-domain QoS-service provisioning*. In Proceedings of the 4th International Conference on Networking and Services, 2008. doi:10.1109/ICNS.2008.43.

[19] P. Rubach and M. Sobolewski. *Dynamic SLA negotiation in autonomic federated environments*. In Proceedings of On the Move to Meaningful Internet Systems, 2009. doi:10.1007/978-3-642-05290-3_36.

[20] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. *Web services agreement specification (WS-Agreement)*, 2011. http://www.ogf.org/documents/GFD.193.pdf.

[21] D. Battré, F. Brazier, K. Clark, M. Oey, A. Papaspyrou, P. Wieder, and W. Ziegler. *WS-Agreement negotiation version 1.0*, 2011. http://www.ogf.org/documents/GFD.192.pdf.

[22] S. Hudert, H. Ludwig, and G. Wirtz. *Negotiating SLAs – an approach for a generic negotiation framework for WS-Agreement*. Journal of Grid Computing, 7(2):225–246, 2009. doi:10.1007/s10723-009-9118-3.

[23] P. Hasselmeyer, H. Mersch, B. Koller, H.-N. Quyen, L. Schubert, and P. Wieder. *Implementing an SLA negotiation framework*. In Proceedings of Expanding the Knowledge Economy: Issues, Applications, Case Studies (eChallenges), pages 154–161, 2007.

[24] M. Parkin, P. Hasselmeyer, B. Koller, and P. Wieder. *An SLA re-negotiation protocol*. In Proceedings of the 2nd Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop, 2008.

[25] F. Kuipers, P. Van Mieghem, T. Korkmaz, and M. Krunz. *An overview of constraint-based path selection algorithms for QoS routing*. IEEE Communications Magazine, 40(12):50–55, 2002. doi:10.1109/MCOM.2002.1106159.

[26] G. Xue, W. Zhang, J. Tang, and K. Thulasiraman. *Polynomial time approximation algorithms for multi-constrained QoS routing*. IEEE/ACM Transactions on Networking, 16(3):656–669, 2008. doi:10.1109/TNET.2007.900712.

[27] M. J. Todd. *The many facets of linear programming*. Mathematical Programming, 91(3):417–436, 2002. doi:10.1007/s101070100261.

[28] H. Schwarz, D. Marpe, and T. Wiegand. *Overview of the scalable video coding extension of the H.264/AVC standard*. IEEE Transactions on Circuits and Systems for Video Technology, 17(9):1103–1120, 2007. doi:10.1109/TCSVT.2007.905532.

[29] J. Yu, T. Chou, Z. Yang, X. Du, and T. Wang. *A dynamic caching algorithm based on internal popularity distribution of streaming media*. Multimedia Systems, 12(2):135–149, 2006. doi:10.1007/s00530-006-0045-x.

[30] S. Chen, B. Shen, S. Wee, and X. Zhang. *Segment-based streaming media proxy: Modeling and optimization*. IEEE Transactions on Multimedia, 8(2):243–256, 2006. doi:10.1109/TMM.2005.864281.

[31] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. *I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system*. In Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, pages 1–14, 2007. doi:10.1145/1298306.1298309.

[32] S. Mitra, M. Agrawal, A. Yadav, N. Carlsson, D. Eager, and A. Mahanti. *Characterizing web-based video sharing workloads*. ACM Transactions on the Web, 5(2):1–27, 2011. doi:10.1145/1961659.1961662.

[33] Z. Avramova, S. Wittevrongel, H. Bruneel, and D. De Vleeschauwer. *Analysis and modeling of video popularity evolution in various online video content systems: Power-law versus exponential decay*. In Proceedings of the First International Conference on Evolving Internet (INTERNET), pages 95–100, 23-29 2009.

[34] T. Wu, M. Timmers, D. De Vleeschauwer, and W. Van Leekwijck. *On the use of reservoir computing in popularity prediction*. In Proceedings of the Second International Conference on Evolving Internet (INTERNET), pages 19–24, 2010. doi:10.1109/INTERNET.2010.13.

[35] T. Gamer and M. Scharf. *Realistic simulation environments for IP-based networks*. In Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (Simutools), 2008.

[36] W. Tang, Y. Fu, L. Cherkasova, and A. Vahdat. *Modeling and generating realistic streaming media server workloads*. Computer Networks, 51(1):336–356, 2007. doi:10.1016/j.comnet.2006.05.003.

[37] N. Karmarkar. *A new polynomial time algorithm for linear programming*. Combinatorica, 4(4):373–395, 1984.

[38] T. Monath, M. Kind, T. Heger, M. Schlesinger, and J. Aznar. *Economical analysis of experience-optimized service delivery*. In Proceedings of the 9th Conference on Telecommunications, Internet and Media Techno Economics (CTTE), 2010. doi:10.1109/CTTE.2010.5557711.

4

Towards a predictive cache replacement strategy for multimedia services

J. Famaey, T. Wauters, and F. De Turck

Submitted to Journal of Network and Computer Applications

The federated service delivery framework presented in Chapter 3 employs dynamically deployed caches to reduce the considerable bandwidth requirements of modern multimedia services. A cache transparently stores a subset of the content closer to the end-users, allowing it to be served without having to traverse the entire network and thus reducing bandwidth requirements and transmission costs. As caches can only store a subset of the available content, a cache replacement strategy is needed to determine what content to cache. Traditional strategies directly use the historical popularity as an indicator of future popularity. In contrast, this chapter proposes a novel strategy that first predicts the future popularity of content, taking into account popularity dynamics. Additionally, an algorithm is presented to perform the actual popularity predictions. It fits a set of popularity models to the historical request trace. These fits are subsequently used to predict future requests and popularity. The goal is to determine the theoretical maximal caching performance gain that can be achieved using predictive caching. Moreover, the effect of prediction errors on caching performance is studied in detail.

4.1 Introduction

The proliferation of interactive, personalized and on-demand television services is causing an increasing need for bandwidth in telecom operator networks. Obviously, broadcasting or multicasting cannot sufficiently reduce bandwidth consumption of on-demand multimedia services. Proxy caching, which had already been widely employed in the delivery of web content, has been proposed as a way of offloading bottleneck links [1] in on-demand scenarios. Caches are strategically placed throughout the network and store a subset of the available content. However, the size of such caches is usually limited, so they are only capable of storing a fraction of available content. Therefore, it is very important to accurately predict the future popularity of content, so that the most popular items, or item segments, can be offered closer to the end-users.

Over the years, many caching strategies have been proposed. Traditional strategies, such as Least Recently Used (LRU) and Least Frequently Used (LFU), assume that what was most popular in the past, will also be most popular in the future. However, the popularity of multimedia content is known to be highly dynamic [2]. Consequently, caching efficiency can be further increased by taking these dynamics into account and actually try to predict future popularity instead of directly applying historical information.

Predicting the future popularity of individual multimedia content items can be reduced to a time series prediction problem [3]. Several efforts have been made to apply this theory to the prediction of multimedia content popularity [4, 5]. However, to our knowledge, these predictions have never been integrated into an actual cache replacement strategy. Additionally, the effect of important parameters, such as the prediction window size, has not yet been thoroughly evaluated.

This chapter presents a generic popularity prediction algorithm. In contrast to existing algorithms, it is not tailored to a specific service. The prediction algorithm fits a set of functions to the cumulative request pattern of content items. These fitted curves are then used as an approximated model of the request pattern. Through extrapolation the future of the request pattern can then be estimated. Subsequently, we present a novel prediction-based cache replacement strategy. It uses the predicted request patterns to determine the subset of all available content to store in the cache. Additionally, to assess the theoretical maximal gain in caching efficiency that can be achieved using predictions, a theoretical variant is also presented. It assumes the future can be perfectly predicted.

The proposed cache replacement strategies are thoroughly evaluated and compared to traditional strategies that directly employ historical information. The goal of this evaluation is to determine both the theoretical and practical gain in caching efficiency that can be achieved using popularity prediction. Moreover, the effect of the prediction window parameter is assessed. This parameter is defined as the

future time-frame that is predicted (i.e., the counterpart of the history window parameter of LFU). The effect of this parameter is influenced by the cache size. Therefore, the synergy between these parameters is thoroughly evaluated. In order to increase the applicability and validity of the presented results, all evaluations are performed using a trace of an actual deployed Video on Demand (VoD) service of a leading European telecom operator. This gives our evaluations more leverage and credibility than those performed on synthetically generated datasets. The ultimate goal of this study is to show that popularity prediction indeed improves caching efficiency and to determine under what circumstances it achieves the most optimal result.

The remainder of this chapter is structured as follows. Section 4.2 gives a more in depth description of existing work on popularity prediction of multimedia content. Section 4.3 presents our generic popularity prediction algorithm. The cache replacement strategy that uses it, is discussed in Section 4.4. Subsequently, Section 4.5 evaluates the proposed cache replacement strategy using simulation results. Finally, the chapter is concluded in Section 4.6.

4.2 Related work

The large size and stringent sequential delivery demands of multimedia content have caused a push towards novel caching strategies. Traditional caching strategies have been adapted to operate on individual content segments instead of entire items [6, 7]. This allows the caches to better utilize the sequential nature of multimedia content demand patterns. Additionally, such techniques better map to the skewed internal popularity of multimedia content. Yu et al. argue that selecting a suitable segment size is a complex problem and therefore propose an alternative solution that models the internal popularity of multimedia streams independent of segment size [8]. Certain IP-TV services have specific properties that can be exploited by caching strategies. For example, the use of sliding-window caches has been proposed in the context of time-shifted TV services [9]. In line with our work, these techniques aim to improve caching efficiency. Nevertheless, they focus on a different aspect, which falls outside the scope of this chapter.

In the field of time series prediction, a wide range of techniques have been developed for forecasting all sorts of time series. Recently, machine learning techniques, such as support vector machines and artificial neural networks have been applied to this problem [10, 11]. Recently, wyffels et al. have used reservoir computing, a form of recurrent neural networks, for time series prediction [12]. Additionally, time series often exhibit repeating trends and periodical effects. For example, multimedia content request patterns often show repeating effects on a daily and weekly basis. The use of wavelet decomposition has been proposed to decompose time series into signals with dynamics in different scales. This has

been shown to simplify prediction with neural network based techniques [13]. This approach was also successfully combined with reservoir computing [14].

Recently, several studies have been conducted on modelling the popularity of multimedia content. These studies can be split into two types. A first type focuses on characterizing the popularity distribution among different multimedia objects. Concretely, such a distribution models the static popularity relationship between the content items offered by a multimedia service. It can be used to derive the probability that the content item with a specific popularity index (e.g., the X^{th} most popular item) will be requested. Many models have been proposed for modelling the popularity distribution of a multimedia service, with Zipf-like distributions (e.g., Zipf-Mandelbrot) the most popular [15]. The second type focusses on modelling the popularity evolution of individual multimedia files. It thus allows the request evolution of content to be estimated, based on historical request information. This latter type of research is also the focus of our work. Most work on this topic was performed in the context of video-sharing services such as YouTube [4, 5, 16]. Cha *et al.* found that there is a strong correlation between the popularity of a video after two days and after ninety days [16]. These observations were supported by a study performed by Szabo *et al.* [5]. An alternative approach was proposed by Avramova *et al.* [4]. They found that YouTube video popularity traces follow several different distributions, such as power-law or exponential. An analytical model is devised that predicts the distribution associated with specific popularity traces. In the context of VoD services, De Vleeschauwer & Laevens propose a prediction method based on a generic user-demand model derived from traces of VoD and catch-up TV services [2]. Wu *et al.* adapted the previously mentioned reservoir computing approach to the popularity prediction of multimedia content [17]. In summary, some work has been done on the modelling and prediction of content popularity evolution. However, these previous studies have not applied this work to content caching. This application is the focus of our work.

This chapter extends previous work by ourselves [18], where the theoretical variant of the predictive cache replacement strategy was presented. Here, a practical prediction algorithm is proposed and combined with the previously introduced strategy. This additionally allows us to characterise the influence on performance of prediction errors.

4.3 Predicting content popularity

This section presents a generic algorithm to predict the content popularity evolution of multimedia content. Concretely, the prediction algorithm estimates the evolution of the content's cumulative request pattern, based on historical information. It uses non-linear optimization techniques to fit a given set of models to the historical input data. These fitted models can subsequently be used to extrapolate

the request pattern's future evolution. The remainder of this section provides a more in depth and formal overview of the algorithm.

Given is a cumulative request pattern $R_c(t)$, which is a function of time representing the total number of perceived requests of content object c up to time t. The goal of the popularity prediction algorithm is to estimate the value of $R_c(t_1)$ at some future point in time $t_1 > t$, given the value of $R_c(t_2)$ for (a representative sample of) all moments in time $t_2 \leq t$. The algorithm approximates the request history up to time t_2 with a set of popularity distribution models $\mathcal{D}$. A popularity distribution $D(t) \in \mathcal{D}$ is a function of time that mathematically models request patterns. It is characterized by a set of parameters P_D. The presented algorithm supports an arbitrary set of popularity distributions. However, we have identified four that cover a wide range of request patterns:

- **Constant**: This distribution is capable of modelling unpopular content that receives no or very few requests over long periods of time. Additionally, it supports the modelling of a constant request rate. Its parameter set P_D consists of two parameters a and b, which respectively represent the slope and intercept. The associated cumulative distribution represents the linear function and is expressed as follows:

$$D(t, a, b) = a \times t + b \tag{4.1}$$

- **Power-law**: Allows the modelling of steep changes in popularity. It has been previously proposed in literature as a model for cumulative request patterns of multimedia content [4]. It is characterized by two parameters, C and α, which respectively represent the normalization constant and scaling factor. The cumulative distribution function is defined as follows:

$$D(t, C, \alpha) = C \times t^{\alpha} \tag{4.2}$$

- **Exponential**: In contrast to power-law, this distribution is used to model more rounded and slow changes in popularity. Its parameter set consists of the rate parameters α and λ. The cumulative distribution function is expressed as follows:

$$D(t, \alpha, \lambda) = \alpha\left(1 - e^{-\lambda t}\right) \tag{4.3}$$

- **Gaussian**: This distribution represents an S-shaped pattern: a steep increase in popularity pre- and succeeded by a constant request rate. It is characterized by the mean μ and standard deviation σ. The cumulative distribution function is expressed as follows:

$$D(t, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{t} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx \tag{4.4}$$

Subsequently, every distribution $D(t)$ is fitted to $R_c(t)$ using an unconstrained non-linear optimization algorithm [19–21], which is capable of finding the parameter values that minimize the error between $D(t)$ and $R_c(t)$. Throughout the rest of this chapter the Levenberg-Marquardt algorithm [19] is used, as it has been shown to be faster and more robust than other existing approaches [22]. The optimization algorithm is used to find the optimal values $P_{D,c}^{\text{opt}}$ for the parameters P_D that allow $D(t)$ to best approximate $R_c(t)$ according to some metric. We call this metric the *fitting metric*. In this chapter, we use the mean squared error (MSE) as a fitting metric. It measures the average of the squares of the errors. Using this metric, the fitting algorithm minimizes the following function when finding the optimal parameter values $P_{D,c}^{\text{opt}}$:

$$f(P_D) = \frac{\sum_{t_2=0}^{t} \left(R_c(t_2) - D(t_2, P_D)\right)^2}{t} \tag{4.5}$$

Based on the optimal parameter values $P_{D,c}^{\text{opt}}$ of every distribution $D(t) \in \mathcal{D}$, a distribution $D_c^{\text{opt}}(t)$ needs to be selected for predicting the future request evolution of object c. This selection is made based on the quality of the fit to the historical trace. The metric used to assess this quality is called the *selection metric*. The MSE metric could for example be used. However, as our goal is to determine the theoretical performance gain that can be achieved, we define the optimal (OPT) selection metric. It selects the distribution that results in the best absolute prediction within the prediction interval. This is obviously a theoretical metric as it uses information only available in the future. As such, it provides the theoretical upper performance limit of the presented prediction algorithm (i.e. it assumes the best candidate distribution is always chosen). It is calculated using the following formula:

$$\left| R_c(t_1) - R_c(t) - \left(D\left(t_1, P_{D,c}^{\text{opt}}\right) - D\left(t, P_{D,c}^{\text{opt}}\right)\right)\right| \tag{4.6}$$

Finally, the selected distribution $D_c^{\text{opt}}(t)$ and its optimal parameters $P_{D,c}^{\text{opt}}$ are used to predict the future. More specifically, the estimated value of $R_c(t_1)$ is defined as $D_c^{\text{opt}}\left(t_1, P_{D,c}^{\text{opt}}\right)$.

Note that time is a continuous variable. To reduce the total number of data points in the historical request pattern, we introduce the concept *time granularity*. The time granularity θ defines the interval of the sampled data points in the request pattern. Concretely, the historical request pattern $R_c(t)$ contains a sampled value for time instants $\{0, \theta, 2\theta, ..., t\}$. On one hand, reducing the granularity will allow the algorithm to make more fine-grained predictions. On the other hand, this will also increase its execution time. Throughout the rest of this chapter, a value of 1 hour is used for θ.

The algorithm is graphically illustrated by way of an example in Figure 4.1. It shows the request trace of an actual video in a deployed VoD system over the

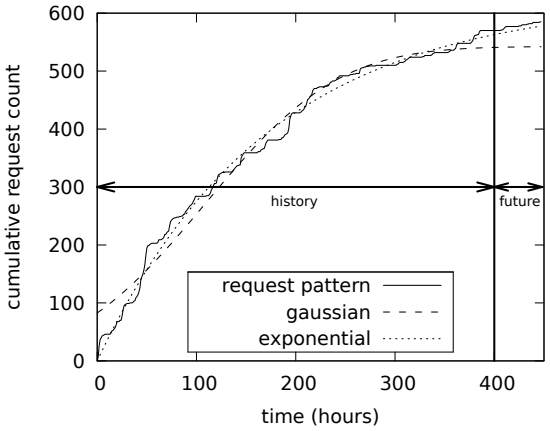


Figure 4.1: Optimal fit of the exponential and gaussian distributions to an example cumulative request pattern; the vertical line represents the current point in time t, on its left is the 400-hour known history, on its right the 48-hour predicted future

course of 448 hours. The prediction algorithm was applied to the first 400 hours of the trace (the known history) using the exponential and gaussian distributions. The parts of the curves before the vertical line represent the fits to known history, while the parts after the line represent the predictions. The actual number of requests that occur in the interval $[400, 448[$ is 15. The exponential distribution predicts 14.55 requests within that interval, which is very close to the real value. On the other hand, the gaussian distribution predicts only 1.25. This is also reflected in the figure, which show that the gaussian distribution poorly approximates the start and end of the request pattern. Note that when using the prediction algorithm in combination with a cache replacement strategy (cf. Section 4.4), the absolute prediction errors are not always a good indicator for the caching performance. Instead, the relative ordering of content items implied by the prediction algorithm's output will determine the eventual caching efficiency.

4.4 Predictive cache replacement strategy

The predicted future popularity of content can be used as input for a predictive cache replacement strategy. This section describes a cache replacement strategy that uses predicted popularity to make more efficient replacement decisions. In contrast to traditional strategies, it uses the expected future popularity of content as a measure instead of the known historical popularity.

The proposed strategy is called *Predictive Least Frequently Used* (P-LFU). It is a predictive version of the LFU caching strategy. LFU keeps track of the num-

ber of times that every object is requested. The objects that receive the highest number of requests within a specified time frame (i.e., the history window) are kept in cache [23]. In contrast, the P-LFU strategy uses the predicted number of requests for each object, instead of the known number of requests in the past. The objects with the most predicted number of requests in the prediction window W, are kept in cache. The prediction window is a configurable parameter. A larger value will allow the algorithm to take into account longer term popularity variations, but is also more prone to prediction errors. We introduce two variants of the strategy. The first assumes the future can be perfectly predicted. The other uses the prediction algorithm presented in Section 4.3. Throughout the rest of this chapter they are referred to as *Perfect Predictive Least Frequently Used* (PP-LFU) and *Optimal-Selection Predictive Least Frequently Used* (OP-LFU) respectively. Using the notations introduced in Section 4.3, we can define the number of estimated requests for object c at time t up to time $t + W$ as follows for PP-LFU:

$$r\left(t, W, c\right) = R_c\left(t + W\right) - R_c\left(t\right) \tag{4.7}$$

The PP-LFU strategy thus uses the actual request pattern R_c to achieve a perfect estimation. On the other hand, OP-LFU uses the optimal estimator D_c^{opt}:

$$r\left(t, W, c\right) = D_c^{\text{opt}}\left(t + W, P_{D,c}^{\text{opt}}\right) - D_c^{\text{opt}}\left(t, P_{D,c}^{\text{opt}}\right) \tag{4.8}$$

Whenever a request arrives for object c_i that is not currently cached, it replaces the cached object c_j if and only if $r(t, W, c_j) < r(t, W, c_i)$ and $\forall c \in \mathcal{C}$: $r(t, W, c_j) \leq r(t, W, c)$, with $\mathcal{C}$ the set of all cached objects. Or in other words, at every time t, the cache contains the subset of objects with the highest estimated request count within the interval $[t, t + W]$.

4.5 Results & discussion

This section evaluates the presented popularity prediction algorithm (cf. Section 4.3) and predictive caching strategies (cf. Section 4.4). More specifically, the prediction algorithm is evaluated in terms of accuracy and execution time. Additionally, the effect of the prediction window parameter W on caching efficiency is studied in more detail. Finally, the predictive cache replacement strategies are compared to the traditional LFU caching strategy [23], as well as the theoretical optimal caching strategy MIN [24]. The results are not compared to the widely used LRU strategy, because our previous results showed that LFU significantly outperforms LRU for the used dataset [18]. The MIN strategy replaces the object in the cache whose next request occurs furthest in the future. It has been proven to be optimal in terms of cache hit rate [25]. It thus gives a theoretical upper bound on performance. However, it has no practical use, as the time of the next request

cannot be known in advance. Throughout this evaluation a history window of 12 hours is used for LFU. We have previously shown this to be a near optimal value for small cache sizes in combination with the dataset used in this chapter [18]. The *cache hit rate* is used as an evaluation metric. It is defined as the percentage of requests that can be served from a cache, as opposed to from the origin content server.

4.5.1 Evaluation scenario

The dataset employed in the evaluation consists of a request trace of the VoD service of a leading European telecom operator, measured over a period of 32 days between Friday February 5 2010 and Monday March 8 2010. Within this period, a total of 75013 requests were sent by 8392 unique users for 4971 different movies. Figure 4.2 graphically depicts the properties of the dataset. The popularity distribution over the movies is shown in Figure 4.2a. The popularity distribution is highly skewed. A total of 691 requests were measured for the most popular movie, while 10 or less requests were received for over 72% of all movies. Figure 4.2b depicts the request count per day. The figure clearly shows the weekly trend in the dataset. The five peaks represent the five weekends part of the trace, with increased activity on Friday, Saturday and Sunday. In addition to the weekly pattern, there is a daily pattern (not depicted in the figure). On weekdays, two peaks are observed. A first, smaller, peak starts as early as 1 pm and lasts until about 5 pm. The second peak occurs during the evening from approximately 8 pm until midnight. On Saturday and Sunday, high request rates persist from 9 am until midnight.

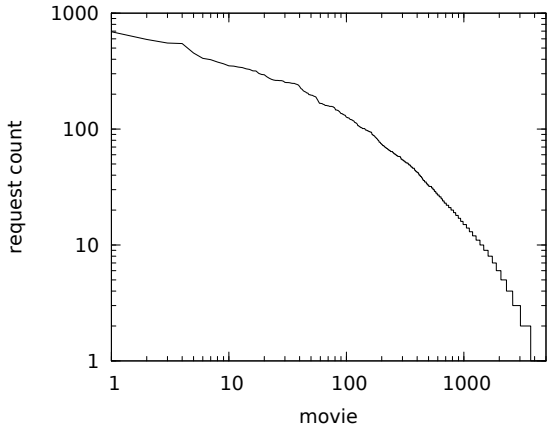
The evaluation scenario considers a single content server and proxy cache. The topology thus consists of a content server directly connected to the proxy cache. This cache is then directly connected to all end-users. The depicted cache hit rate results were measured in the intermediary proxy cache.

4.5.2 Prediction algorithm evaluation

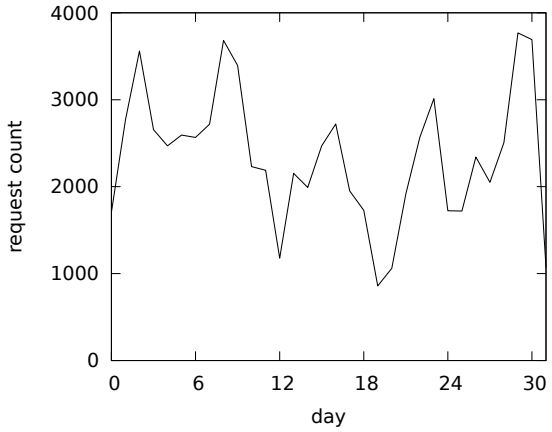
This section evaluates the accuracy and performance of the prediction algorithm presented in Section 4.3. All results are depicted as a function of the history length. This is the length (i.e., number of data points) of the historical request trace used for the curve fitting step of the algorithm.

4.5.2.1 Prediction accuracy

The prediction accuracy represents the error of the predicted future request frequency compared to the actual future request frequency. As a metric for accuracy, the *absolute prediction error* is used. For a popularity distribution $D \in \mathcal{D}$, predic-



(a) popularity distribution



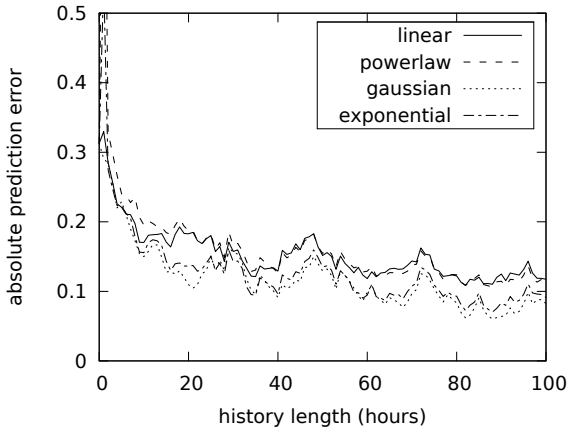
(b) requests per day

Figure 4.2: A graphical representation of the Video on Demand dataset

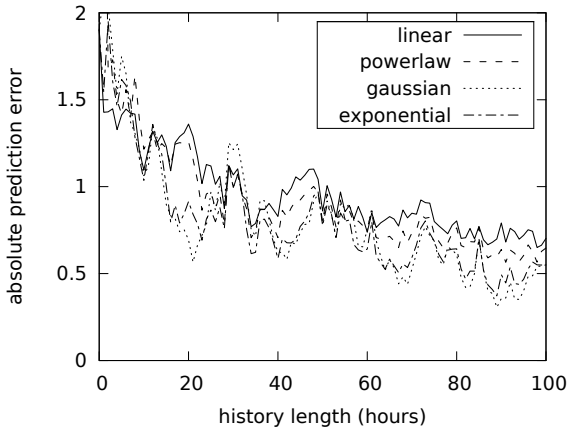
tion window W and distribution parameters P, it is calculated as follows:

$$|R_c(t+W) - R_c(t) - (D(t+W,P) - D(t,P))| \tag{4.9}$$

In other words, the absolute prediction error is defined as the difference between the actual number of requests and the predicted number of requests in the interval $[t, t+W]$. The goal of this section is to assess the effect of the history length and prediction window W parameters on the prediction error. Figure 4.3 depicts the absolute prediction error as a function of the history length for the four different popularity distributions. Figure 4.4 depicts the same for the exponential distribution only, but for multiple values of the prediction window W.



(a) Averaged over all objects



(b) Averaged over the 250 most popular objects

Figure 4.3: The absolute prediction error averaged over all objects as a function of the history request pattern length for $W = 1$ hour

Figure 4.3a plots the prediction error as a function of the number of historical datapoints used in the curve fitting step of the prediction algorithm, averaged over all movies in the trace. On the other hand, Figure 4.3b depicts the prediction error, averaged over the 250 most popular movies only. As expected, the prediction error decreases significantly as the number of historical datapoints grows. For a small history size (i.e., a few hours), the prediction error is very large. However, it quickly converges to the optimum. More specifically, there is no significant difference between the prediction after 20 hours and after 100 hours. This is obviously expected to influence performance of a predictive cache replacement strategy, as

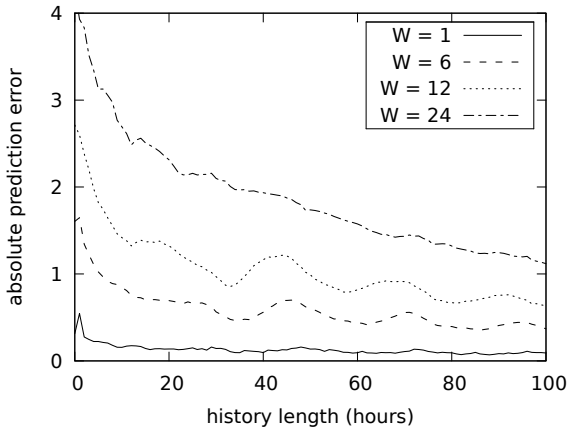
its predictions will be less accurate for newly introduced content. Additionally, comparing the two figures shows a significant different in prediction error averaged over all movies as compared to averaged over the most popular movies. This proves that it is more difficult to predict the future of popular content than that of unpopular content. Finally, the linear and power law distributions result in the worst overall predictions. Although the difference between the four distributions is insignificant when averaged over all movies, it is much clearer when looking at the 250 most popular movies only. In the latter case, the exponential and gaussian distributions clearly outperform the other two, resulting in much more accurate predictions (for a large enough history).

In addition to the history length, the prediction window is also expected to influence prediction accuracy. More specifically, a bigger prediction window is assumed to lead to larger errors, as it is easier to predict the nearby future. Figure 4.4a depicts the prediction error of the exponential fit as a function of the history length for different prediction window sizes W, averaged over all movies. Figure 4.4b depicts the same, but averaged over the 250 most popular movies only. The figures confirm our assumptions and clearly shows the direct linear connection between the prediction error and window W. However, for a request history of more than 100 datapoints, the prediction error averaged over all movies is less than 1 request even for a prediction window of 24 hours. For popular content the error increases and a prediction window of 24 hours results in an average error of less than 6 requests for a history window of 100 datapoints or more.

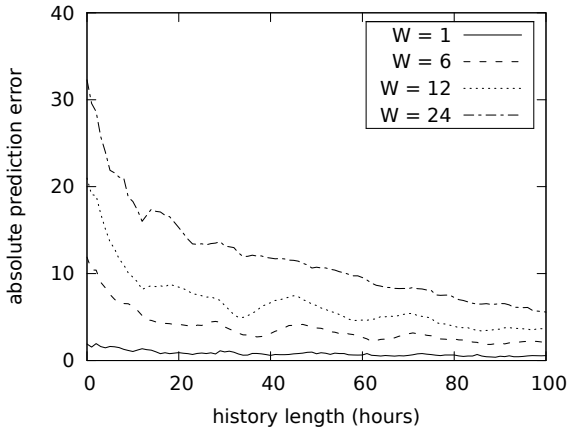
In summary, the above results lead to several pertinent conclusions. First, predicting the future is very difficult if the size of the known history is small. This makes popularity prediction of newly introduced content highly error prone. However, results showed that a short history trace (e.g., 20 hours) already reduces the error to a near optimal value. Additionally, there is a direct linear relationship between the prediction error and the prediction window W. Nevertheless, even for a prediction window as large as 24 hours, the error can be reduced to less than 1 request on average as long as the historical trace is large enough.

4.5.2.2 Execution time

A deployed proxy cache usually operates in an online fashion. It decides whether to cache an object or not on-the-fly as requests arrive. The proposed predictive caching strategies need to execute the prediction algorithm once per time granularity interval θ for every content item that was requested during the interval. As such, it is important that the algorithm executes in a feasible time in order to support online popularity prediction. Figure 4.5 depicts the execution time of the curve fitting step of the prediction algorithm for the four employed distributions. As the curve fitting step of the algorithm is by far the most computationally intensive, its execution time is representative for the entire algorithm. The presented



(a) Averaged over all objects



(b) Averaged over the 250 most popular objects

Figure 4.4: The absolute prediction error of the exponential fit for different prediction windows (in hours) as a function of the history request pattern length

results were obtained using a test machine with a dual-core AMD Opteron™ 2212 processor and 4 GiB of memory.

The figure plots execution time as a function of the historical trace length. As expected, there is a linear correlation between execution time and history length. Additionally, the curve fitting algorithm's execution time is significantly different depending on the popularity distribution. The linear and power law distributions have a very low fitting time compared to the exponential and gaussian distributions. Concretely, for a trace of 400 datapoints, the fitting takes 16, 26, 188 and 1007 milliseconds for the linear, power law, exponential and gaussian distribution

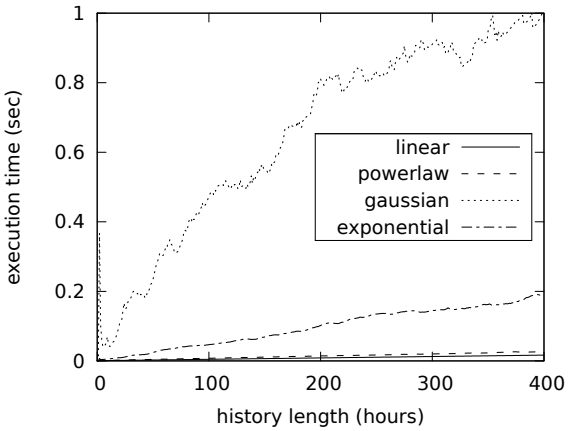


Figure 4.5: The processing time required to fit a single distribution to a request pattern as a function of the history request pattern length

respectively. This is in line with the complexity of each distribution. The linear and power law distributions contain only two variables. The exponential and gaussian distributions have three variables each. Additionally, the gaussian distribution is more difficult to calculate as it contains an integral. Nevertheless, the combined execution time of all distributions for a long trace of up to 400 hours long, is only about 1.2 seconds. This allows up to 3000 popularity distribution updates in a one hour interval. Under more stringent time constraints, this time can be further reduced by either limiting the historical data that is taken into account, or limiting the maximum amount of iterations the fitting algorithm may perform. Additionally, the algorithm's execution time is significantly impacted by the number of variables in the popularity distributions. Limiting the amount of variables in the employed distributions would therefore greatly increase efficiency.

4.5.3 Predictive cache replacement strategy evaluation

This section evaluates the caching performance of the proposed predictive cache replacement strategies. The MIN, PP-LFU and OP-LFU cache replacement strategies all give an upper bound on performance. The MIN strategy achieves the optimal cache hit rate and represents the absolute upper bound on caching efficiency. The PP-LFU strategy provides the upper bound for frequency-based prediction strategies, as it assumes the future is perfectly predicted. Finally, OP-LFU uses a real prediction algorithm (and thus introduces prediction errors), but assumes the best popularity distribution is always chosen. It thus represents the upper bound on performance that can be achieved using the prediction algorithm presented in Section 4.3. This section consists of two parts. First, the optimal value of the pre-

diction window parameter W is determined. Second, PP-LFU and OP-LFU are compared to to the traditional LFU and optimal MIN strategies.

4.5.3.1 Prediction window parameter

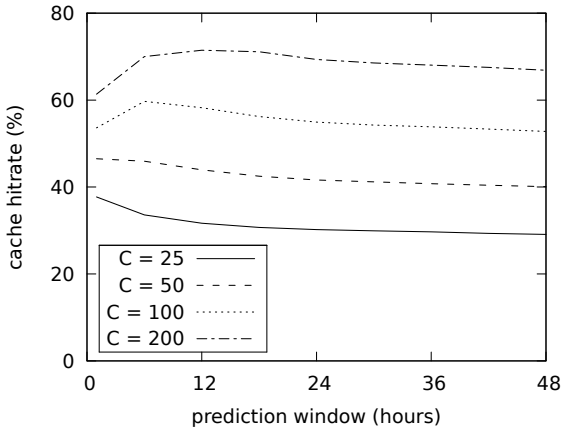
The prediction window parameter allows the predictive cache replacement strategy to adapt the future time frame that is taken into account. If this time frame is too short the strategy runs the risk of ignoring important future popularity fluctuations. However, if it becomes too long the prediction accuracy significantly decreases and the strategy might take into account expected popularity changes that are not yet relevant. The results in Figure 4.6 depict the cache hit rate as a function of the prediction window W for different cache sizes C.

The figure shows that there is indeed a peak in cache hitrate. Additionally, the location of this peak is directly proportional to the cache size. Comparing Figures 4.6a and 4.6b also proves that this is the case for both PP-LFU and OP-LFU. More specifically, for a very small cache of 25 objects, the optimal prediction window is 1 hour. As the cache size increases to 100 objects, the optimum increases to 6 hours. Finally, for a larger cache of 200 objects, the optimal prediction window is 12 hours. The results additionally show that estimating the prediction window parameter value both too large or too small results in decreased performance. In the depicted results, the optimal prediction window value performs up to 33% better than the worst. As such, a practical implementation should intelligently adapt its prediction window parameter to the size of the cache, in order to prevent significant drops in performance. Throughout the remainder of this section, a prediction window of 12 hours is chosen, as it achieves good performance for caches of 50 or more objects.

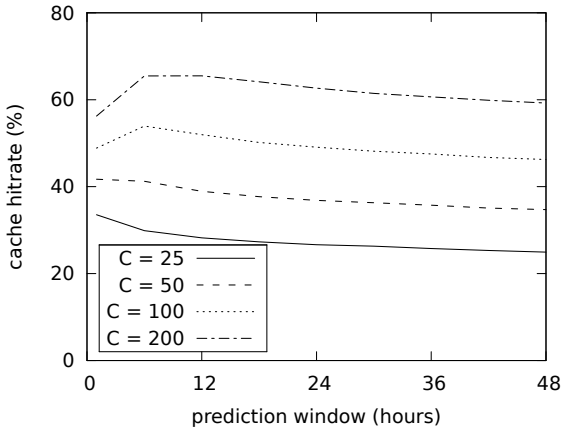
4.5.3.2 Comparison with traditional strategies

This section compares the cache hit rate of the PP-LFU and OP-LFU with that of LFU and MIN. This provides us with insights of how well predictive caching can perform compared to a good traditional strategy (i.e., LFU) and the theoretical optimum (i.e., MIN). Figure 4.7 depicts these results. As expected, MIN performs best, closely followed by PP-LFU, OP-LFU and finally LFU.

The results for PP-LFU represent the theoretical upper bound that can be achieved using a frequency-based predictive cache replacement strategy. As the cache size increases, its results approach the optimum more closely. For a cache size of 50 objects, PP-LFU performs 17% worse (i.e., a cache hit rate difference of 8.5%) than MIN, while for a cache size of 200 objects it only performs 3% worse (i.e., a cache hit rate difference of 2.5%). Additionally, PP-LFU's caching efficiency is considerably better than that of the traditional LFU strategy, performing around 20% better for all depicted cache sizes.



(a) PP-LFU



(b) OP-LFU

Figure 4.6: The cache hit rate as a function of the prediction window W for different cache sizes C

The OP-LFU strategy gives a theoretical upper bound on cache efficiency when using the prediction algorithm presented in Section 4.3. In contrast to PP-LFU, it is thus subject to prediction errors, which explain the reduced performance of OP-LFU. For all depicted cache sizes, its caching efficiency is about 10% worse than that of PP-LFU. Compared to MIN, its efficiency increases as the cache size grows. For a small cache of 50 objects, it is up to 25% worse, while for a larger cache of 200 objects it is only 11% worse. Additionally, OP-LFU performs considerably better than LFU. Its gain in efficiency even increases as the cache size grows, with a 5% improvement for small caches up to 50 objects and over 10% for a larger

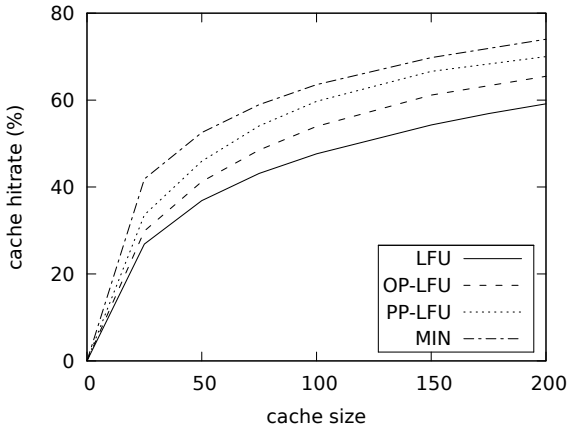


Figure 4.7: Comparison of the different cache replacement strategies in terms of cache hit rate, as a function of cache size for $W = 12$ hours

cache of 200 objects.

In summary, the presented results show that applying popularity prediction to cache replacement has the potential of significantly improving the cache hit ratio compared to traditional strategies such as LFU. Theoretically, employing the perfect prediction allowed the algorithm to improve up to 20% compared to LFU. The prediction errors introduced by an actual prediction algorithm based on curve fitting reduced the maximum performance gain to 10%.

4.6 Conclusion

The goal of this chapter was to investigate the merits of using popularity prediction techniques in cache replacement strategies. To this end, we proposed a novel generic popularity prediction algorithm. It estimates the future popularity of multimedia content by fitting a set of popularity distributions to the known request history. The set of popularity distributions can be adapted to fit the characteristics of the multimedia service to which it is applied. Additionally, a predictive variant of the Least Frequently Used cache replacement strategy, called P-LFU, is proposed. It uses the predicted future request frequency to determine what content to cache. Two theoretical versions of P-LFU are considered. PP-LFU assumes the future can be perfectly predicted. It thus gives an upper bound on performance that can be achieved using P-LFU. OP-LFU instead employs the presented popularity prediction algorithm, but assumes the popularity distribution that best predicts the future is selected. It thus gives an upper bound on performance that can be achieved using P-LFU, when used in combination with the presented prediction

algorithm.

A detailed simulation study, using an actual Video on Demand trace file, was performed in order to evaluate the merits of prediction-based cache replacement. The evaluation consists of two main components. First, the popularity prediction algorithm was validated. Second, PP-LFU and OP-LFU were compared, in terms to cache hit rate, to LFU and the theoretical optimum. This evaluation lead to several pertinent conclusions. First, the prediction accuracy is severely impacted by the number of available historical datapoints. This is especially true for very short history lengths of less than 10 datapoints. This makes predicting the popularity of newly introduced content highly error prone. Additionally, there is a direct linear relationship between the prediction window parameter and the prediction error. Nevertheless, even for a prediction window as large as 1 day, the error can be reduced to on average less than 1 request as long as the known history is large enough. For the predictive cache replacement strategies it was shown that the optimal value of the prediction window parameter is directly proportional to the cache size. Choosing a suboptimal prediction window was shown to lead to performance drops of up to 33%. A practical predictive cache replacement strategy could thus automatically adapt its prediction window according to changes in cache size in order to further optimize performance. Moreover, applying popularity prediction to cache replacement has the potential of significantly improving the cache hit rate compared to traditional strategies, such as LFU. Under the assumption that the future can be perfectly predicted (i.e., PP-LFU) an improvement of up to 20% can be achieved. However, when instead using the actual prediction algorithm, this improvement is reduced to at most 10%.

The presented OP-LFU strategy uses information about the future popularity of content to select the best popularity distribution for prediction. As this approach is infeasible in a practical deployment, our future work consists of finding a good estimator for the selection of the optimal popularity distribution based on historical information.

References

[1] M. Allen, B. Zhao, and R. Wolski. *Deploying video-on-demand services on cable networks*. In 27th International Conference on Distributed Computing Systems (ICDCS '07), pages 63–63, 2007. doi:10.1109/ICDCS.2007.98.

[2] D. De Vleeschauwer and K. Laevens. *Performance of caching algorithms for IPTV on-demand services*. IEEE Transactions on Broadcasting, 55(2):491–501, 2009. doi:10.1109/TBC.2009.2015983.

[3] S. Makridakis. *Time series prediction: Forecasting the future and understanding the past*. International Journal of Forecasting, 10(3):463–466, 1994.

[4] Z. Avramova, S. Wittevrongel, H. Bruneel, and D. De Vleeschauwer. *Analysis and modeling of video popularity evolution in various online video content systems: Power-law versus exponential decay*. In First International Conference on Evolving Internet (INTERNET), pages 95–100, 2009. doi:10.1109/INTERNET.2009.22.

[5] G. Szabo and B. A. Huberman. *Predicting the popularity of online content*. Communications of the ACM, 53(8):80–88, 2010. doi:10.1145/1787234.1787254.

[6] K.-L. Wu, P. Yu, and J. Wolf. *Segmentation of multimedia streams for proxy caching*. IEEE Transactions on Multimedia, 6(5):770–780, 2004. doi:10.1109/TMM.2004.834870.

[7] S. Chen, H. Wang, X. Zhang, B. Shen, and S. Wee. *Segment-based proxy caching for internet streaming media delivery*. IEEE Multimedia, 12(3):59–67, 2005. doi:10.1109/MMUL.2005.56.

[8] J. Yu, C. Chou, Z. Yang, X. Du, and T. Wang. *A dynamic caching algorithm based on internal popularity distribution of streaming media*. Multimedia Systems, 12(2):135–149, 2006. doi:10.1007/s00530-006-0045-x.

[9] T. Wauters, W. Van de Meerssche, P. Backx, F. De Turck, B. Dhoedt, P. Demeester, T. Van Caenegem, and E. Six. *Proxy caching algorithms and implementation for time-shifted TV services*. European Transactions on Telecommunications, 19(2):111–122, 2008. doi:10.1002/ett.1181.

[10] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. *An experimental unification of reservoir computing methods*. Neural Networks, 20(3):391–403, 2007. doi:10.1016/j.neunet.2007.04.003.

[11] R. Samsundin, A. Shabri, and P. Saad. *A comparison of time series forecasting using support vector machine and artificial neural network model*. Journal of Applied Sciences, 10(11):950–958, 2010. doi:10.3923/jas.2010.950.958.

[12] F. wyffels and B. Schrauwen. *A comparative study of reservoir computing strategies for monthly time series prediction*. Neurocomputing, 73:1958–1964, 2010. doi:10.1016/j.neucom.2010.01.016.

[13] S. Soltani. *On the use of the wavelet decomposition for time series prediction*. Neurocomputing, 48:267–277, 2002. doi:10.1016/S0925-2312(01)00648-8.

[14] F. wyffels, B. Schrauwen, and D. Stroobandt. *Using reservoir computing in a decomposition approach for time series prediction*. In European Symposium on Time Series Prediction, pages 149–158, 2007.

[15] W. Tang, Y. Fu, L. Cherkasova, and A. Vahdat. *Modeling and generating realistic streaming media server workloads*. Computer Networks, 51(1):336–356, 2007. doi:10.1016/j.comnet.2006.05.003.

[16] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. *I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system*. In 7th ACM SIGCOMM conference on Internet measurement (IMC). ACM, 2007. doi:10.1145/1298306.1298309.

[17] T. Wu, M. Timmers, D. De Vleeschauwer, and W. Van Leekwijck. *On the use of reservoir computing in popularity prediction*. In The Second International Conference on Evolving Internet (INTERNET), pages 19–24, 2010. doi:10.1109/INTERNET.2010.13.

[18] J. Famaey, T. Wauters, and F. De Turck. *On the merits of popularity prediction in multimedia content caching*. In IFIP/IEEE International Symposium on Integrated Network Management (IM), pages 17–24, may 2011. doi:10.1109/INM.2011.5990669.

[19] K. Levenberg. *A method for the solution of certain non-linear problems in least squares*. Quarterly Journal of Applied Mathmatics, 2(2):164–168, 1944.

[20] J. A. Nelder and R. Mead. *A simplex method for function minimization*. The Computer Journal, 7(4):308–313, 1965. doi:10.1093/comjnl/7.4.308.

[21] M. J. D. Powell. *An efficient method for finding the minimum of a function of several variables without calculating derivatives*. The Computer Journal, 7(2):155–162, 1964. doi:10.1093/comjnl/7.2.155.

[22] S. Katare, A. Bhan, J. M. Caruthers, W. N. Delgass, and V. Venkatasubramanian. *A hybrid genetic algorithm for efficient parameter estimation of large kinetic models*. Computers & Chemical Engineering, 28(12):2569–2581, 2004. doi:10.1016/j.compchemeng.2004.07.002.

[23] N. Megiddo and D. S. Modha. *Outperforming LRU with an adaptive replacement cache algorithm*. Computer, 37(4):58–65, 2004. doi:10.1109/MC.2004.1297303.

[24] J. Gecsei, D. R. Slutz, and I. L. Traiger. *Evaluation techniques for storage hierarchies*. IBM Systems Journal, 9(2):78–117, 1970. doi:10.1147/sj.92.0078.

[25] B. Van Roy. *A short proof of optimality for the MIN cache replacement algorithm*. Information Processing Letters, 102(2-3):72–73, 2007. doi:10.1016/j.ipl.2006.11.009.

5

A hierarchical approach to autonomic network management

J. Famaey, S. Latré, J. Strassner, and F. De Turck

Published in International Workshop on Management of the Future Internet

The continuing growth of the Internet and the proliferation of services with stringent QoS requirements have significantly contributed to the increasing complexity and cost to manage the Internet and its services. Autonomic network management aims to solve this problem, by giving the network the capability to govern itself. To effectively manage dynamic federations of large networks, the self-governing capabilities should be distributed among many autonomic management components, or AEs. These AEs should be able to interact in a scalable and efficient manner. The remainder of this dissertation focusses on these AE interactions. Specifically, this chapter presents an AE collaboration architecture that structures AEs within a domain in a hierarchy. This allows them to communicate and collaborate in a scalable manner. Additionally, the interactions across network domains, which are necessary when setting up federations, are discussed. The architecture's merits are evaluated using an analytical model, which shows its capability to retain scalability under an increasing number of AEs. Chapter 6 and Appendix A further extend the AE collaboration architecture with automated semantic information dissemination, filtering and aggregation. Chapter 7 applies all introduced concepts to a scalable management framework for federated clouds.

5.1 Introduction

In recent years, communication networks have greatly increased in size, complexity, and heterogeneity. Additionally, the end-user and service requirements have become drastically more diverse and stringent. Hence, managing these complex and large-scale systems is proving increasingly difficult and this complexity is likely to increase in the Future Internet. To alleviate the problems associated with managing current and future communication networks, the autonomic communication networks paradigm has been introduced [1, 2].

The ultimate goal of autonomic network management systems is to automatically adapt the network's services and resources in accordance with changing environmental conditions and user needs [3]. Policy-Based Network Management [4–6] gives these systems the ability to automatically perform low-level configurations in compliance with high-level business goals. This will allow human administrators to focus on high-level tasks. Consequently, the increasing management complexity will be handled by the system itself.

It has been generally agreed upon that autonomic architectures for managing current and future networks and services should be distributed for scalability reasons [7–9]. Distribution of autonomic components provides a means to keep up with the exploding growth of the number of network devices, services, and end-users. However, little research has been performed on how exactly these distributed autonomic components should collaborate and communicate. As a first step, a solution has been proposed in the form of combining autonomic components in a hierarchical structure [8, 10]. In this chapter we build upon these first ideas, and give a detailed description of the interactions between autonomic components in a hierarchical autonomic management architecture. We argue that by grouping autonomic components into a hierarchy, the network overhead associated with managing network devices and other resources can be greatly reduced. Additionally, dissemination of context, propagation of policies, and collaboration between autonomic components can be more efficiently orchestrated, resulting in a more scalable architecture. Finally, the hierarchical structure can be logically mapped to the structure of the organization and their infrastructure, simplifying configuration and management at all layers of the organization.

The contributions of our work are threefold. First, we propose a novel hierarchical approach to structuring autonomic components. Second, the interactions in this hierarchically structured autonomic network are identified and discussed in detail. This includes the propagation of context and policies, and governance of child autonomic components. Third, an analytical study that evaluates the scalability of this new approach has been performed. The results are discussed in the second part of this chapter.

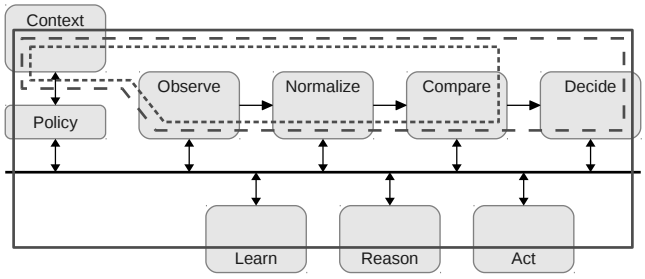
This chapter is organized as follows. A brief overview of existing autonomic

management architectures, and more specifically FOCALE, is given in Section 5.2. Subsequently, Section 5.3 further explores the proposed hierarchical architecture. The introduced concepts are evaluated in Section 5.4. Finally, Section 5.5 concludes this chapter.

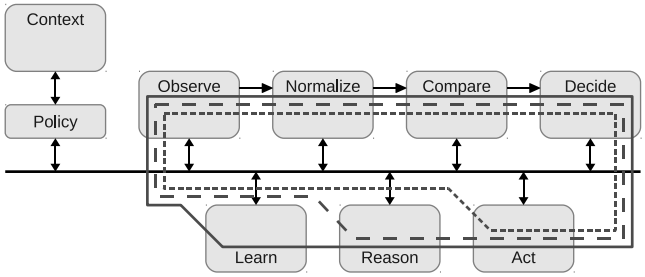
5.2 FOCALE autonomic management architecture

Since the conception of autonomic computing and communications, many autonomic control loops and architectures have been proposed [8, 11]. They share the common goal of autonomically adapting the behavior of managed resources if their state differs from the desired state. However, the term "autonomic" is often interpreted in different ways, which is reflected in the various approaches used to implement autonomic control loops. The hierarchical autonomic architecture proposed in this chapter is based on the FOCALE architecture and control loops [8, 10]. However, the ideas presented in this chapter can conceptually be applied to other autonomic architectures as well, as these architectures face the same challenges regarding distribution. Furthermore, in describing the hierarchical autonomic architecture, we do not introduce specific FOCALE components, but merely use FOCALE as an example throughout the chapter. We have chosen FOCALE because it aims to free network administrators from performing low-level configuration tasks, allowing them to focus on high-level network planning and optimization. These low-level tasks are performed by the network itself, which uses reasoning and learning components to adapt its behavior to context changes. This adaptive behavior is governed by policies, representing the high-level business goals. Additionally, the FOCALE Cognitive Model [8] supports collaboration between Autonomic Elements (AE) by grouping them into communities, and providing hooks that support centralized and decentralized governance. Cooperation between AEs is fundamental towards achieving the hierarchical autonomic architecture proposed in this chapter.

The FOCALE architecture provides a set of outer and inner control loops, as shown in Figure 5.1. The outer control loops perform large-scale adjustments by reacting to context changes. On the other hand, the inner control loops make more detailed adjustments of functionality within a specific context. Both outer and inner loops come in three types: reactive, deliberative, and reflective. The reactive path is taken when adapting to a previously analyzed context change. In such a case, a previously inferred behavior change can be performed, without the need for complex reasoning. The deliberative control loops are used when context changes that are not sufficiently well understood take place. Finally, the reflective loops provide a means to better understand how context changes affect the goals of AEs. Note that the three types of outer and inner loops of FOCALE were inspired by congnitive psychology, and correspond to modeling how a human makes decisions



(a) outer loop view



(b) inner loop view

Figure 5.1: The FOCALE control loops [8]

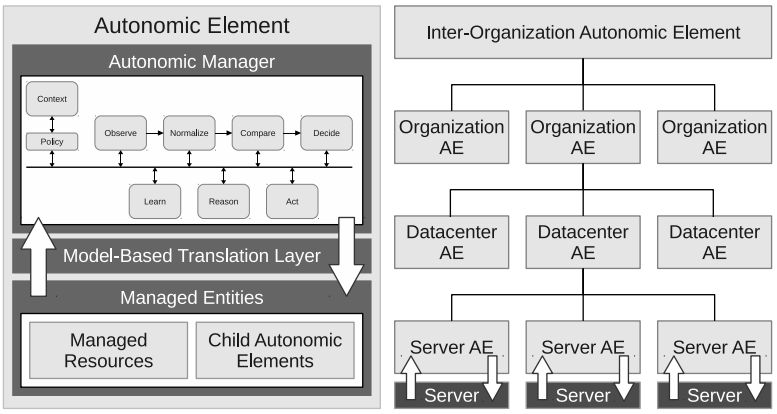
using short- and long-term memory.

In addition to its advanced control loops, FOCALE introduces the notion of the enhanced Autonomic Element. It is an abstraction that allows FOCALE to provide distributed functions such as communication, learning, reasoning, and management. Each AE provides a set of services to perform knowledge management, composition, business-enabling, and orchestration. Additionally, AEs can cooperate and collaborate in communities, by sharing functionality and information.

The next section gives a detailed overview of the concepts we devised to augment existing autonomic management architectures in order to support hierarchical collaboration between autonomic components.

5.3 Hierarchical management architecture

In our proposed hierarchical autonomic network management architecture, AEs are grouped together in cooperating communities, or *clusters*. Each AE is composed of a set of *managed entities*, which are either *managed resources* or AEs themselves. Managed resources are oblivious to the autonomic management capabilities of the network, and fully depend on the parent AE to govern their man-



(a) hierarchical autonomic element (b) example hierarchy tree

Figure 5.2: A hierarchical autonomic element manages a set of managed resources and child autonomic elements, together forming the set of managed entities*; The autonomic elements are structured in a logical tree topology*

agement decisions. Child AEs are guided by their parent, but also have autonomic decision-making capabilities of their own. Figure 5.2 shows the structure of a hierarchical AE and gives an example of how the hierarchy can be mapped to the physical infrastructure. Figure 5.2a shows a simplified view of the FOCALE AE. The heart of our enhancement lies in the managed entities container, which replaces the FOCALE managed resource. The managed entities container consists of managed resources and/or child AEs. In the example shown in Figure 5.2b each server is managed by its own AE. Additionally, all servers within a datacenter are grouped together in a cluster. At the top layer, several organizations cooperate via an inter-organization AE. Note that combining different organizations into a single AE introduces additional difficulties. This, and an alternative method for collaboration between organizations is further discussed in Section 5.3.4.

By introducing parent-child relationships, the AEs in the network will form a logical tree. At the bottom layer AEs only manage a set of managed resources, while the root of the tree effectively governs the entire network. This approach introduces a hybrid management scheme, where AEs within a cluster are managed in a centralized way by the parent AE, while management across the tree is distributed. Note that although logically the parent AE is a single entity, it may be physically distributed across multiple devices to improve scalability and robustness.

The hierarchical autonomic approach has several advantages over a flat autonomic architecture. First, scalability is improved, as context no longer needs to

be exchanged between every pair of cooperating AEs, but only needs to be sent to the parent AE. This greatly reduces management overhead. Additionally, by aggregating and filtering the exchanged context, overhead can be even further reduced. AEs at higher levels in the hierarchy thus have a broader, but less detailed, view on the managed resources. This allows them to perform large-scale reasoning and decision-making in a scalable manner. On the other hand, AEs at the bottom of the hierarchy can use more detailed information to react faster and more precise, but on a smaller scale. This approach also mirrors the design of FOCALE's outer and inner control loops, with the outer loops defining the coarse context for governance, and the inner loops defining the finer-grained management within that context. Second, AEs that need to cooperate or share common goals can be grouped together in a cluster. The hierarchical structure greatly simplifies governing the interactions between them, and aligning their behavior. Additionally, this can be exploited to facilitate the business interactions between organizations. Finally, the layers of the tree can be more easily mapped to the hierarchical structure of organizations and infrastructure. This facilitates the translation and mapping of business goals and policies to the actual network configurations.

The rest of this section elaborates upon the different types of interaction between AEs in the hierarchy tree.

5.3.1 Cluster management

A cluster is defined as the group of AEs that share the same parent AE. It is necessary to determine which AE in the cluster will act as the parent. In a stable network, where devices stay online for prolonged periods, the parent AE can be statically determined. In the example shown in Figure 5.2b, a datacenter AE can be chosen in advance, as datacenters are mostly static environments. However, in a more dynamic network, devices and thus AEs might randomly go offline and online. In such a case, a robust leader election protocol can be used to dynamically determine cluster parents [12, 13].

In addition to parent selection, an AE must be assigned to a specific cluster. As the general structure of a network is static, even over longer periods of time, we believe an AE would not often change its position in the hierarchy. The cluster of an AE can thus be statically specified using policies. However, for scalability reasons, clusters becoming too large could be split into sub-clusters, by introducing an additional layer in the hierarchy. Figure 5.3 shows this process by way of an example.

Policies can be used to define a maximum threshold for overhead generated by intra- and inter-cluster communication. If this threshold is exceeded, the autonomic manager detects the invalid state and executes the cluster splitting algorithm. Analogous to splitting, peer clusters with small populations can be recom-

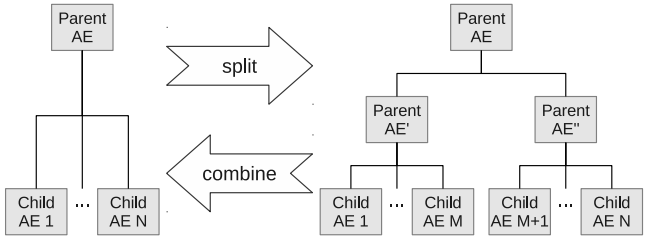


Figure 5.3: A cluster can be dynamically split into several sub-clusters to improve scalability

bined. An example policy for splitting clusters is shown below (using the Ponder policy specification language [14]):

```
inst oblig splitCluster {
    on       overhead(cluster) > X %
    subject  p = parent(cluster)
    do       p.splitChildCluster()
}
```

In the example, the parent AE of a cluster is asked to split its children into multiple sub-clusters if the overhead generated by the cluster for management communications is greater than X% of the consumed bandwidth. More details on determining the splitting threshold are given in Section 5.4.2.

5.3.2 Context dissemination

Context is a vital part of any autonomic system. It is used to model the current state of the managed entities, which in turn allows the system to adapt to changes when necessary. Context of managed resources can be obtained by way of active or passive monitoring using standard protocols or techniques, such as SNMP [15]. Child AEs, on the other hand, cannot be directly monitored. They make parts of their own context available to the parent AE, for example through a publish-subscribe mechanism. Using policies, context can be flagged private or public. Public context is made available to the parent, while private context is not disseminated. This allows for enforcing privacy and ensures that contextual data can be exchanged between organizations. For example, if an inter-organization AE groups together several cooperating organizations, they do not want all information about their infrastructure and services to be made public, but only specific types of information that are needed to support the collaboration. Figure 5.4 shows the process of context dissemination throughout the hierarchy in more detail.

Although all public context is available to the parent AE, not all data is unconditionally sent to the parent. In a typical publish-subscribe mechanism, which

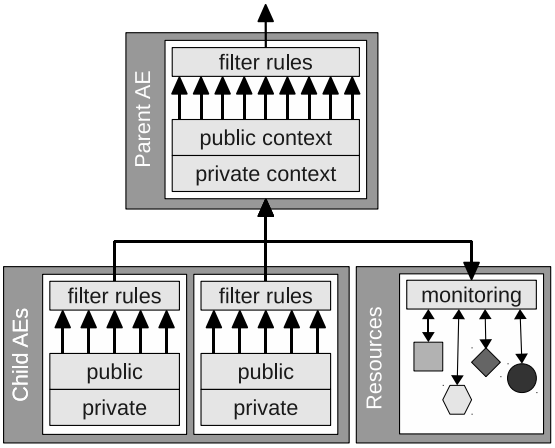


Figure 5.4: Public context of an AE is aggregated and filtered before it is disseminated to the parent AE; Private context is not made available to the parent

can form the basis for the context dissemination process, filter rules are used that allow parent AEs to inform their children about the context in which they are interested [16]. Additionally, filter rules can be used to define the aggregation of detailed information before it is disseminated to the parent. Filtering and aggregation allow the overhead, in terms of bandwidth consumption and reasoning time, to be greatly reduced. Aggregation gives the parent AE a broader, but less detailed, view on the managed entities, allowing it to reason and take decisions on a larger scale, without greatly increasing execution time of reasoning algorithms.

Policies also play a vital role in combination with filter rules: they are used in two distinct situations. First, similar to flagging context public or private, policies are also used as a means to limit the visibility of contextual data to parent AEs. For example, the AE managing a datacenter might have detailed information on the resource consumption of all its servers. However, an AE at a higher level might only be allowed to view more general statistics showing average or maximum consumption over all servers in the datacenter. Policies allow tuning the amount of aggregation that is needed between different AE levels, and thus tune the view a parent AE gets on its children. These policies can be specified both by human operators as well as AEs; in the first case an operator will restrict the view because of trust issues between parent and child AE (e.g. they belong to different organizations), in the latter case policies can be automatically specified when the overhead exceeds a threshold. Second, policies can be used to enable the dynamic composition of filter rules. For example, a parent AE managing a video delivery service might only request a general Quality of Experience score from its children during normal operation. However, when this score becomes too low,

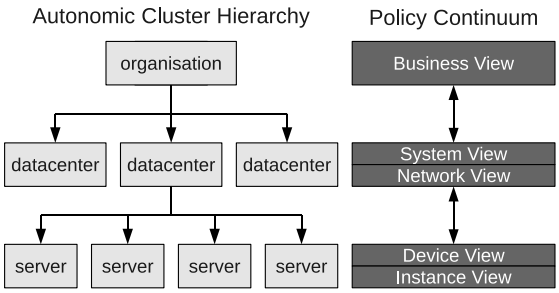


Figure 5.5: High-level policies are propagated down the cluster hierarchy; The AEs can be mapped to the views of the Policy Continuum

more detailed information could be requested, in order to diagnose and resolve the problems. In Latré et al. [16], an ontology based approach is suggested that uses semantic reasoning to dynamically change the set of filter rules. Here, policies can be specified by the network operator in the form of ontology rules.

5.3.3 Policy interaction

Although administrators of an autonomic system are not directly involved in configuring management algorithms and managed entities, they do control the entire process by adding policies to the policy repository. At the higher levels of the hierarchy, these policies correspond to the business goals of the organization. At lower levels they become more specific, defining the desired state of AEs in more technical ways. The approach of translating policies from general business goals to more implementation-specific technical rules has already been proposed as the Policy Continuum [5]. Figure 5.5 shows an example of how the views of the Policy Continuum could be mapped to the cluster hierarchy. However, this should be viewed as just an example. In reality, many possible ways of mapping the continuum views to the hierarchy exist. As shown in the figure, a one-to-one mapping between the Policy Continuum and the cluster hierarchy is not always possible. However, multiple continuum views can be combined within a single AE layer, or views can stretch over multiple AEs.

The Policy Continuum paradigm provides a clear abstraction of the complexity present in the different levels. By linking policies at different levels, management problems can be split more easily into smaller, and hopefully easier to tackle, problems. If the process of linking policies can be automated, the policy translation process can be automated as well. In such an approach, changing a policy at the higher level can immediately trigger the change of one or more policies at the lower levels. If policies can be changed in multiple ways, strategies can aid in determining which path to take. The automation of policy translation requires

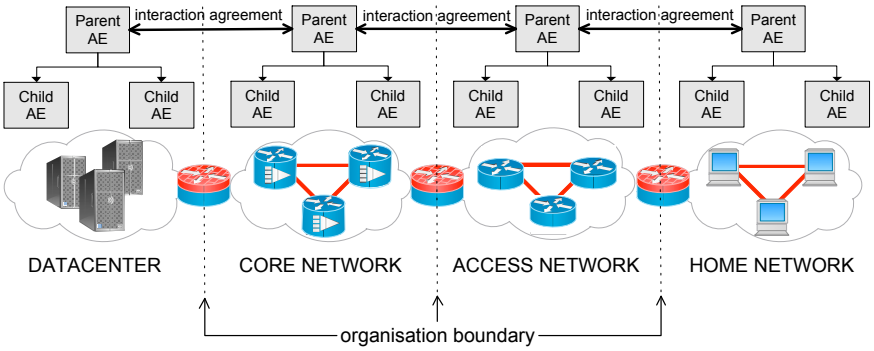


Figure 5.6: Mapping of different hierarchical structures onto an exemplary network topology, spanning multiple domains.

at least a policy authoring infrastructure that is able to detect and tackle policy conflicts, as argued in Davy et al. [17]. In this approach, a policy conflict analysis algorithm is used, which can form the basis for policy translation.

5.3.4 Autonomic element collaboration

In the previous sections we discussed how the exchange of policies and context on one hand and the organization of autonomic elements themselves on the other hand can be handled. Another important communication aspect in a distributed management environment is the behavioral orchestration between AEs. Such orchestration is both needed in an inter- and intra-domain management scenario.

An inter-domain management scenario occurs when two or more organizations collaborate to offer composite services. For example, a cloud computing provider, wanting to offer an end-to-end Quality of Service guarantee for a service, needs to collaborate with the network providers that manage the networks where his data transits to ensure that the management actions he/she undertakes are supported or, even better, re-enforced by those network providers. In the strict hierarchical approach, the solution to this problem would be to instantiate a new inter-organization AE that orchestrates the QoS guarantees. However, in an inter-organizational scenario, such an approach is often not feasible due to privacy issues or a lack of a shared infrastructure. We argue that, between organizations, interactions between parent AEs belonging to different organizations should be maintained instead of enforcing the strict parent child interaction. This is illustrated in Figure 5.6, which depicts the mapping of several management hierarchies onto an exemplary network topology.

This parent-to-parent communication can only work if there is an unambiguous interaction agreement on (1) what each party can expect from each other, or in other words, what management functions are made available to each organization

and (2) what the effect is of each management function on the network context. While crucial in an inter-domain scenario, such agreements can also play an important part in regular parent-child AE interactions. For a parent AE, a good management strategy is to assign his child AEs with specific management tasks that have a smaller scope and require a smaller reaction time. The assigned child AEs are then responsible for independently complying with the assigned task. Without an interaction agreement, the parent AE has no formal guarantee that the assigned task will be executed. Through interaction agreements, one party can delegate management authority to another party in both an inter- and intra-domain scenario.

As proposed in van der Meer et al. [18], such an interaction agreement can be established by using the Design by Contract paradigm as originally presented by Bertrand Meyer [19]. A contract enables the formal specification of the functional and non-functional characteristics of a distributed artefact such as a management function. In van der Meer et al. [18] the L-ADS language is presented, which allows contracts to be defined for a distributed management environment.

5.3.5 Management algorithms

The management algorithms are responsible for configuring the managed entities, to make sure their state reflects a desired state of the system. They are guided by the reasoning and learning components, which may change the algorithms' parameters to adjust their behavior. Managed resources are directly configured by the algorithms, but child AEs are not. However, as discussed earlier, the algorithms are capable of influencing child behavior through policies or contracts.

An algorithm often exists at many layers in the hierarchy. However, its behavior will differ based on its location. For example, at an organization level, a resource reservation algorithm can insert policies into specific datacenter AEs specifying the amount of resources to reserve for each service. At the datacenter level this same algorithm would select specific servers on which to execute the services. Finally, at the server level, the service would be executed and a specific amount of resources would be reserved for it.

5.4 Evaluation

The hierarchical approach of structuring AEs in an autonomic communication network has both qualitative and quantitative advantages compared to classical flat architectures. The qualitative advantages have already been clarified in previous sections. Quantitatively, the hierarchical approach is expected to greatly reduce overhead and increase the view of the autonomic managers on the network and its resources. Existing flat architectures often sacrifice accuracy of context information in order to reduce management overhead, and thus increase scalability. This

often leads to suboptimal decision-making. The hierarchical approach alleviates this problem by giving selected autonomic managers a broader view on the network and its resources, and letting them govern components with more narrow but detailed context information.

In this section, the overhead introduced by exchanging context information in both flat and hierarchical autonomic architectures is evaluated, by way of an analytical model. First, a generic model is given. Subsequently, it is applied to a specific scenario in order to obtain more concrete results.

5.4.1 Analytical model

In flat distributed management architectures, management components enter into peer relationships with other components [20]. These relationships are then used to exchange context information. The number of neighbours of a component thus controls the size of its view on the network, but also the generated overhead. Increasing the number of neighbours of a component will thus increase the accuracy of the available context information, but also the generated network overhead.

Before quantifying this overhead and the size of the network view, we introduce some notations. Assume the network consists of R network resources, each governed by an AE. Additionally, each AE has N neighbours, with $N < R$. Let B represent the size in bytes of one context exchange between two AEs. Information is exchanged between neighbour AEs every I seconds.

The generated overhead per second (in bytes) for a flat architecture can then be calculated as follows

$$\mathcal{O}_{flat} = \frac{R \times N \times B}{I} \tag{5.1}$$

In addition, the size of the view of an AE (as a fraction of the full view) is calculated as follows

$$\mathcal{V}_{flat} = \frac{N+1}{R} \tag{5.2}$$

In the hierarchical architecture, every AE (except the root) is governed by its parent. Therefore, it only sends context information to the parent AE. The parent aggregates and filters the context received by its children before propagating it to its own parent. The equations for the hierarchical architecture thus become more complex, and some additional notations are needed. The architecture consists of L layers. The bottom layer, layer 1, consists of R AEs (one for each network resource). At all layers, AEs are grouped in clusters of at most size C, except the top layer, layer L, which consists of a single root AE. At layer l (with $1 \leq l < L$), a context exchange with the parent is of size B_l bytes. The exchanged context may be different in size at every layer, as it is filtered and aggregated before being propagated. Figure 5.7 shows an example hierarchical AE topology, with $R = 9$, $C = 3$, and $L = 3$.

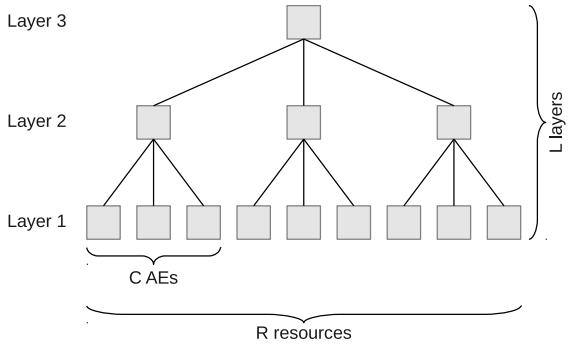


Figure 5.7: An example hierarchical AE topology, with $R = 9$, $C = 3$, and $L = 3$

The generated overhead per second for hierarchical architectures can now be calculated by determining the number of AEs per layer. At layer 1, there is 1 AE per resource, which equals R. At layer 2, there is 1 AE per layer 1 cluster, which equals $\lceil R/C \rceil$. This can be generalized to the number of AEs at layer l (with $1 \leq l < L$) being equal to $\lceil R/C^{l-1} \rceil$. As each AE at layer l sends B_l bytes of context information to its parent every I seconds, the generated overhead (in bytes) can be calculated as follows

$$\mathcal{O}_{hier} = \frac{1}{I} \times \sum_{l=1}^{L-1} \left(B_l \times \left\lceil \frac{R}{C^{l-1}} \right\rceil \right) \tag{5.3}$$

The size of the view is different at every layer. At layer 1, an AE only has context information about its own resource, which results in an $1/R$ fraction of the entire network view. This can be generalized to an arbitrary layer l, resulting in the fraction

$$\mathcal{V}_{hier}(l) = \frac{C^{l-1}}{R} \tag{5.4}$$

Note that if R is not divisible by C, one layer 1 cluster will contain less than C resources, and Equation 5.4 is thus only an upper boundary for that cluster and its parent clusters.

In addition to the total management overhead, the context information that needs to be processed by a single AE also plays an important role in the performance. If the load is not properly balanced, heavily loaded AEs might not be able to process and reason on the large amounts of context they receive. In the flat architecture, under the assumption that all AEs have the same number of neighbours, the load is equally spread across all AEs. The maximum context per second (in bytes) that an AE receives can be calculated as follows

$$\mathcal{R}_{flat} = \frac{N \times B}{I} \tag{5.5}$$

For the hierarchical architecture, the calculation is again less straight forward. At each layer, except the top, an AE has at most C children. However, at the top layer, the root AE has $\lceil R/C^{L-2} \rceil$ children. This results in the following equation for calculating the maximum AE load in bytes per second

$$\mathcal{R}_{hier} = \frac{1}{I} \times \max\left(\max_{1 \leq l \leq L-2} (B_l) \times C, B_{L-1} \times \left\lceil \frac{R}{C^{L-2}} \right\rceil \right) \tag{5.6}$$

In the rest of this section, the analytical model is applied to a specific service delivery middleware scenario, and more concrete results are derived from the equations.

5.4.2 Results

In order to derive more tangible results from the analytical model, we apply it to a *service delivery middleware* scenario [20–22]. A service delivery middleware is a middleware substrate responsible for managing a large set of application services. The tasks performed by the middleware include: allocation of resources to service instances, selection and composition of services, and request admission control. In this section we focus on the resource allocation component. It is responsible for deciding which application services to execute on every server in the datacenter. It is often a distributed component that is executed on every server separately, using context information from other servers. The resource allocation component, as described in [20], uses the following context information:

- The available and used resources per resource type on the server
- The number of requests received and satisfied for every application service
- The amount of required and supplied resources per resource type for every application service

Assume the number of different resource types equals T, and each server executes on average S application services. Additionally, associated with each resource type and application service is a 4 byte identifier. A resource amount is represented using 8 bytes, while request counts are represented using 4 byte integers. The information about a resource type requires $8 + 8 + 4 = 20$ bytes, and about an application service $4 + 4 + 4 + (8 + 8 + 4) \times T = 12 + 20 \times T$ bytes. Therefore, the size in bytes of a context exchange message can be calculated as follows

$$B = 20 \times T + S \times (12 + 20 \times T) \tag{5.7}$$

In the rest of this section we assume: a datacenter with 1000 servers (R), 5 resource types (T), on average 100 application services per server (S), and a context exchange interval of 1 second (I). Using these values, $B = 11300$ bytes

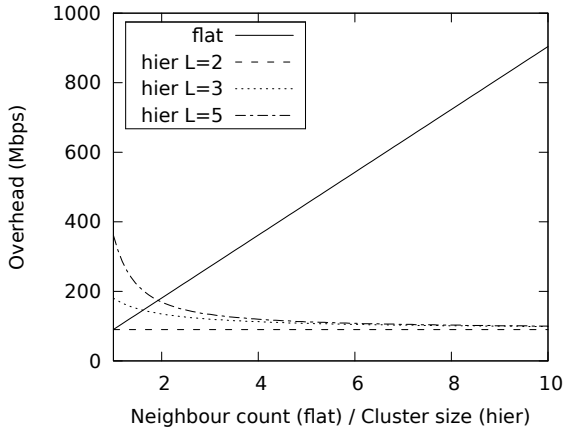


Figure 5.8: The total overhead per AE (in megabits per second) as a function of the neighbour count N (for the flat architecture) and the cluster size S (for the hierarchical architecture)

per second can be derived. Note that the actual values of these parameters do not influence the relative performance of the architectures, but merely the absolute values.

In the hierarchical architecture, context information is averaged over all servers, and averaged per service when it is propagated to the parent AE. This means we can assume the size of a context exchange message is the same at all layers in the hierarchy, or $B = B_l$ for $1 \leq l < L$.

The total overhead generated by context exchange (in megabits per second) is shown in Figures 5.8 and 5.10. The maximum context received by any AE (also in megabits per second) is shown in Figure 5.9. The first two graphs depict the evaluation metrics as a function of the number of neighbours N, for the flat architecture, and cluster size C, for the hierarchical architecture. The third graph is depicted as a function of the network size N.

As shown in Figure 5.8, the hierarchical architecture outperforms the classical flat approach in terms of overhead for almost every combination of N and C. The flat architecture only generates less overhead for a cluster size C of less than 2 AEs per cluster, which is not a suitable cluster size for a network with 1000 resources anyway. Additionally, the performance of the flat approach clearly degenerates quickly as the number of neighbours increases. Even for $N = 10$, which is a view of only about 1% of the entire network, the flat approach generates up to 10 times as much overhead as the hierarchical architecture. More specifically, for 1000 servers, the flat architecture generates 452 and 904 Mbps overhead for 5 and 10 neighbours respectively. On the other hand, the overhead is limited to 130 and 100

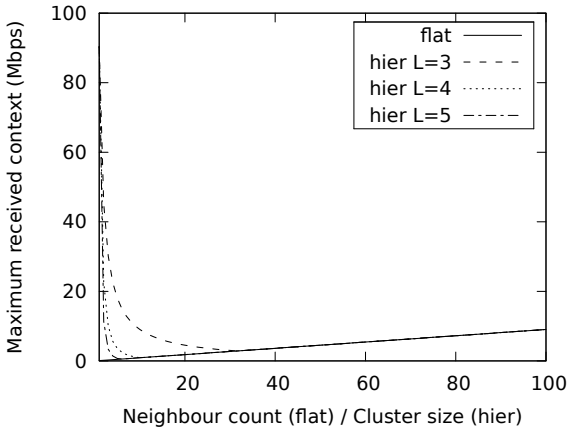


Figure 5.9: The maximum received context per AE (in megabits per second) as a function of the neighbour count N (for the flat architecture) and the cluster size S (for the hierarchical architecture)

Mbps for respectively a cluster size of 3 and 10 in the hierarchical case. Finally, the performance of the hierarchical architecture quickly approaches the optimum, even for a topology with 5 layers and a cluster size of only 10 AEs per cluster.

As previously shown, increasing the cluster size decreases the total generated overhead of the hierarchical architecture. However, as shown in Figure 5.9, it also negatively influences the maximum AE load. Therefore, it is necessary to leverage this trade-off by selecting a cluster size C that gives good results for both metrics. As shown in the figure, the maximum load decreases up to a certain cluster size, after which it grows again. The location of this optimum depends on the number of layers L in the topology, and can be derived from Equation 5.6. From this equation, it can be derived that the optimal maximum load is reached when the following equality holds

$$\max_{1 \leq l \leq L-2} (B_l) \times C = B_{L-1} \times \left\lceil \frac{R}{C^{L-2}} \right\rceil \tag{5.8}$$

As stated earlier, for the considered scenario we assume that $B = B_l$ for $1 \leq l < L$. If furthermore the special case where R is divisible by C is considered, this equation can be rewritten as follows

$$C = \sqrt[L-1]{R} \tag{5.9}$$

Although this equation does not return the exact optimum if R is not divisible by C, it does remain a good approximation. The cluster management algorithm (cf. Section 5.3.1) can exploit Equation 5.9 to help determine the optimal cluster size.

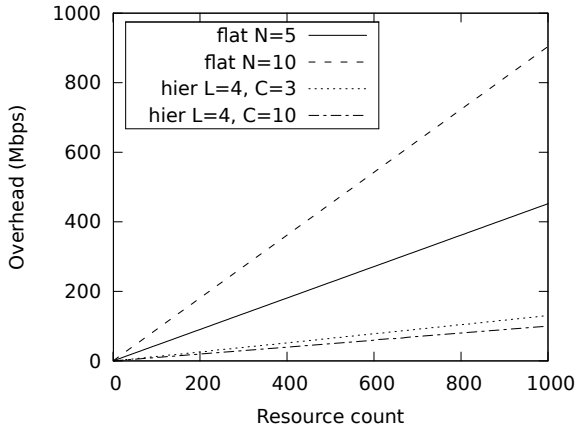


Figure 5.10: The total overhead per AE (in megabits per second) as a function of the size of the network R

The overall scalability of the approaches is compared in Figure 5.10. It depicts the increase in network overhead for context dissemination as a function of increasing number of managed resources, and thus AEs. In order to keep the same relative view size of the network, in the flat architecture, N should grow with the network size. However, as shown in the figure, doubling the size of N also doubles the overhead. This introduces a scalability bottleneck, which can only be circumvented by reducing the relative view size as the network size grows. This in turn will cause the optimality of management algorithms to decrease as the network grows in size. On the other hand, increasing the cluster size in the hierarchical architecture, actually decreases total overhead. This can be exploited to improve scalability. As the network size grows, cluster size can be increased proportionally in order to contain the increase in overhead.

5.5 Conclusion & future work

This work extends existing architectures for autonomic network and service management with several relevant contributions. A novel approach to collaboration and interaction between autonomic elements is introduced. By structuring them in a hierarchical manner, the overhead generated by exchanging context can be greatly reduced. This greatly improves scalability compared to flat decentralized architectures. Additionally, the hierarchical approach provides a more intuitive mapping between the Policy Continuum and the autonomic components. This eases human intervention at all layers of the infrastructure and organization. By way of aggregating and filtering context information, autonomic elements can be

provided with a more fine-grained but narrow, or less detailed but wide view on the network and its resources. This can be exploited by management algorithms to operate at different layers of the hierarchy, taking more general decisions at the top and more specific ones at the bottom layers.

The quantitative advantages, such as a reduction in generated network overhead and improved scalability, were studied in more detail using an analytical model. First, a general model was constructed. Second, this model was applied to a specific scenario, which resulted in several concrete conclusions. The results were used to study the trade-off between scalability in terms of generated network overhead, and the size of the view on the network and its resources.

In future work, we plan to focus on devising service management algorithms that exploit the hierarchical structure of the proposed architecture. Additionally, the hierarchical architecture will be extended to a hybrid federation-based model.

Acknowledgment

Jeroen Famaey is funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT). Steven Latré is funded by the Fund for Scientific Research Flanders (FWO). This work is sponsored in part by the WCU (World Class University) program through the Korea Science and Engineering Foundation funded by the Ministry of Education.

References

[1] S. Dobson, S. Denazis, A. Fernandez, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. *A survey of autonomic communications*. ACM Transactions on Autonomous and Adaptive Systems, 1(2):223–259, 2006. doi:10.1145/1186778.1186782.

[2] B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M. O. Foghlu, W. Donnelly, and J. Strassner. *Towards autonomic management of communications networks*. IEEE Communications Magazine, 45(10):112–121, 2007. doi:10.1109/MCOM.2007.4342833.

[3] N. Agoulmine, S. Balasubramaniam, D. Botvitch, J. Strassner, E. Lehtihet, and W. Donnelly. *Challenges for autonomic network management*. In 1st IEEE International Workshop on Modeling Autonomic Communications Environments (MACE), 2006.

[4] M. Sloman. *Policy driven management for distributed systems*. Journal of Network and Systems Management, 2:333–360, 1994. doi:10.1007/BF02283186.

[5] J. Strassner. *Policy-based network management – solutions for the next generation*. Morgan Kaufman, 2004.

[6] D. Agrawal, K.-W. Lee, and J. Lobo. *Policy-based management of networked computing systems*. IEEE Communications Magazine, 43(10):69–75, 2005. doi:0.1109/MCOM.2005.1522127.

[7] Y. Cheng, R. Farha, M. S. Kim, A. L. Garcia, and J. W. K. Hong. *A generic architecture for autonomic service and network management*. Computer Communications, 29(18):3691–3709, 2006. doi:10.1016/j.comcom.2006.06.017.

[8] J. Strassner, J. Won-Ki Hong, and S. van der Meer. *The design of an autonomic element for managing emerging networks and services*. In IEEE International Conference on Ultra Modern Telecommunications (ICUMT), 2009. doi:10.1109/ICUMT.2009.5345533.

[9] L. Baresi, A. D. Ferdinando, A. Manzalini, and F. Zambonelli. *The CASCADAS framework for autonomic communications*. In Autonomic Communication, pages 147–168, 2009. doi:10.1007/978-0-387-09753-4_6.

[10] J. Strassner, N. Agoulmine, and E. Lehtihet. *FOCALE – a novel autonomic networking architecture*. In Latin American Autonomic Computing Symposium (LAACS), 2006.

[11] J. Kephart and D. Chess. *The vision of autonomic computing*. Computer, 36(1):41–50, 2003. doi:10.1109/MC.2003.1160055.

[12] O. Dagdeviren and K. Erciyes. *A hierarchical leader election protocol for mobile ad hoc networks*. In 8th international conference on Computational Science, pages 509–518, 2008. doi:10.1007/978-3-540-69384-0_56.

[13] N. Schiper and S. Toueg. *A robust and lightweight stable leader election service for dynamic systems*. In IEEE International Conference on Dependable Systems and Networks (DSN), pages 207–216, 2008. doi:10.1109/DSN.2008.4630089.

[14] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. *The Ponder policy specification language*. In International Workshop on Policies for Distributed Systems and Networks, pages 18–38, 2001.

[15] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin. *Simple network management protocol (SNMP)*, 1990. http://www.ietf.org/rfc/rfc1157.txt.

[16] S. Latré, S. van der Meer, F. De Turck, J. Strassner, and J. Won-Ki Hong. *Ontological generation of filter rules for context exchange in autonomic multimedia networks*. In 12th IEEE/IFIP Network Operations and Management Symposium (NOMS), 2010. doi:10.1109/NOMS.2010.5488448.

[17] S. Davy, B. Jennings, and J. Strassner. *The policy continuum-policy authoring and conflict analysis*. Computer Communications, 31(13):2981–2995, 2008. doi:10.1016/j.comcom.2008.04.018.

[18] S. van der Meer. *Architectural artefacts for autonomic distributed systems – contract language*. In Sixth IEEE Conference and Workshops on Engineering of Autonomic and Autonomous Systems (EASe), pages 99–108, 2009. doi:10.1109/EASe.2009.15.

[19] B. Meyer. *Applying "design by contract"*. Computer, 25(10):40–51, 1992. doi:10.1109/2.161279.

[20] C. Adam and R. Stadler. *Service middleware for self-managing large-scale systems*. IEEE Transactions on Network and Service Management, 4(3):50–64, 2008. doi:10.1109/TNSM.2007.021103.

[21] C. Reich, K. Bubendorfer, M. Banholzer, and R. Buyya. *A SLA-oriented management of containers for hosting stateful web services*. In the Third IEEE International Conference on e-Science and Grid Computing (E-SCIENCE), pages 85–92, 2007. doi:10.1109/E-SCIENCE.2007.7.

[22] Y. Cheng, A. Leon-Garcia, and I. Foster. *Toward an autonomic service management framework: A holistic vision of SOA, AON, and autonomic computing*. IEEE Communications Magazine, 46(5):138–146, 2008. doi:10.1109/MCOM.2008.4511662.

6

Semantic context dissemination and service matchmaking in future network management

J. Famaey, S. Latré, J. Strassner, and F. De Turck

Published in International Journal of Network Management

As the size of the managed environment increases, ever-growing amounts of management information are generated and exchanged. The architecture proposed in Chapter 5 copes with this by intelligently aggregating and filtering information. This chapter presents the Semantic Communications Bus (SCB), a substrate that automatically performs such filtering operations. AEs specify their interests in information via semantic filter rules. The SCB employs semantic reasoning to match information with these rules. The use of semantics guarantees the unambiguous interpretation of exchanged information. Moreover, a semantic matchmaking algorithm is proposed and incorporated into the SCB. It facilitates collaboration between AEs, by matching their goals with management functions offered by other AEs. Appendix A extends the SCB with an algorithm that automatically generates and adapts filter rules, based on management goals and the environment. Chapter 7 combines the concepts introduced here, in Chapter 5 and in Appendix A into a management framework for federated clouds. Appendix C incorporates the SCB in an ambient-intelligent framework to optimize continuous care.

6.1 Introduction

The size, complexity and heterogeneity of telecommunications networks has been increasing rapidly in recent years. It is quickly becoming too difficult and costly for human operators to manage such networks manually. As such, a need has arisen for autonomic components that automatically manage and configure the network's resources. These components are guided by the high-level goals set forth by human operators [1]. It is expected that many specialized autonomic elements (AEs) will take part in managing the network to overcome this increased complexity [2]. They all perform specific functions and adhere to a set of different business and/or technical goals, but will need to collaborate and communicate in order to achieve the more complex human-specified management goals. Therefore, it becomes necessary to govern the communication and collaboration between AEs. Recently, the idea of a distributed communications bus, which facilitates the interaction between AEs, has been proposed to solve this problem. For example, the FOCALE architecture [2] includes the enterprise content bus, which is an extension of the enterprise service bus paradigm [3]. This bus should support semantics in order to facilitate a shared understanding of context, goals and actions between AEs.

Additionally, the Future Internet is moving away from silo-based management approaches to inter-domain federated network management [4]. Such federations support the end-to-end management of loosely coupled value networks, spanning several autonomic domains [5]. We argue that in such a case, there is an even greater need for correct interpretation of information in the communication between AEs, as they will need to collaborate across domains. This further proves the necessity of semantically enriched communication.

The communication between AEs is characterized by several types of interaction. In this chapter, we focus on two: dissemination of context and service matchmaking. The management components achieve their goals by monitoring the current state of their managed environment and taking appropriate corrective actions to ensure that the system ends up in the desired overall state. This allows them to optimize performance and detect and rectify problems. In order to maintain an up-to-date view on the state of the managed resources, potentially very large amounts of information need to be disseminated from the network's resources to AEs and between AEs themselves [6]. The context of an entity is defined as the collection of measured and inferred knowledge that describes the state and environment in which that entity exists or has existed [7]. In this chapter, we propose a novel method that allows management components to semantically define the types of context in which they are interested, by way of a set of filter rules. Such a mechanism is necessary in order to reduce the amount of exchanged context to only that which is relevant and thus ensure scalability of the system. Our

proposed method attaches semantics to this context, therefore supporting the definition of filter rules based on the actual meaning of information instead of static predefined keywords and/or string patterns. Additionally, these semantics can be used to automatically generate filter rules by way of semantic reasoning [8].

Although AEs can detect problems in the network's state through the dissemination of context, they may not always be able to solve the detected problems themselves. Therefore, to further support loosely-coupled collaboration between AEs, a matchmaking mechanism for the dynamic discovery of management services is needed. Matchmaking is defined as the process of discovering a set of services or functions that fulfil a given set of requirements. In this chapter, we propose a semantic service matchmaking algorithm that finds AEs offering the requested management functions based on the subsumption relationships of inputs, outputs, preconditions and effects (IOPEs) of the service descriptions. A concept D subsumes a concept B if B is a specialization of D (or alternatively, if D is a generalization of B). By including preconditions and effects in the matchmaking process, management components can better estimate the effects of specific functions on the environmental state. Additionally, as the IOPEs are semantically enhanced, a reasoner can be used to infer semantic relatedness of the requested and offered service definitions.

In order to facilitate these interactions, we propose the Semantic Communications Bus (SCB), which allows AEs to define filter rules that specify the types of context in which they are interested. Additionally, it offers a service matchmaking component, which supports matchmaking of management services with specific inputs, outputs, preconditions and effects. An example scenario, illustrating the SCB's role in the interactions between AEs, is shown in Figure 6.1. It depicts a federated cloud computing scenario, with two datacenters connected through the Internet. The SCB is depicted as a logical substrate spanning the different management domains. Although we make no assumptions concerning the SCB's deployment, in an operational environment its functionality will most likely be distributed among the AEs for scalability reasons. In the presented example scenario, AE 1 and AE 2 monitor the state of the first datacenter's resources. They extract relevant context and publish it onto the bus. AE 3, which previously subscribed to specific types of context information, receives a subset of this published context via the bus. AE 3 uses the gathered context to detect problems that occur within the datacenter (e.g., resource starvation of a specific hosted service). When a problem is detected, AE 3 determines a plan of action to rectify the situation. Such a plan contains a set of resource reconfigurations that need to be performed inside, and possibly outside, the management domain. In order to successfully execute these reconfigurations, the AE needs to discover and initiate functionality offered by other AEs. This is where the SCB's service matchmaking algorithms come into play. In the example scenario, the reconfigurations need to be performed in another

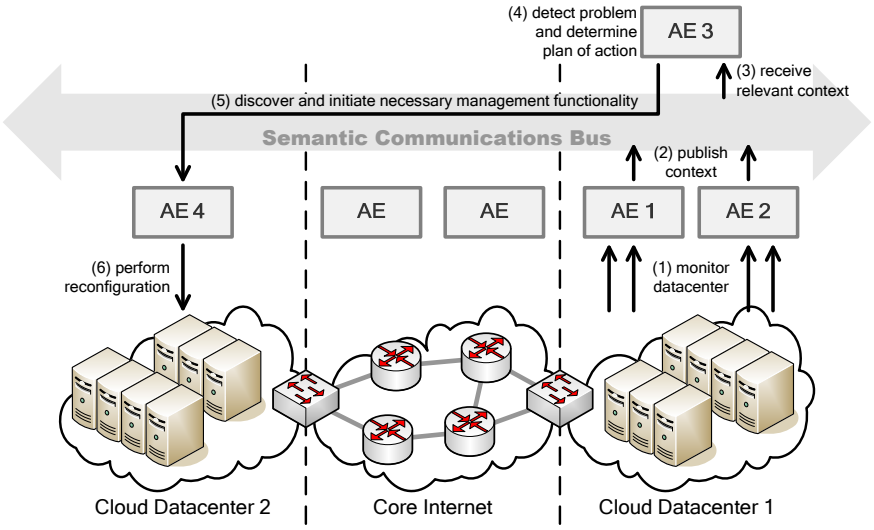


Figure 6.1: An overview of the interactions between the network's resources, management components and human operators

domain, namely datacenter 2. Through the SCB matchmaker, AE 3 discovers that the necessary functionality is offered by AE 4, which could, for example, be the migration of the previously mentioned overloaded hosted service to a server within datacenter 2. The described scenario clearly shows that for AEs to successfully perform their tasks, they need to be able to send and receive relevant context and be able to discover and initiate management services offered by other AEs.

The SCB's semantics are supported through a set of ontologies. These ontologies semantically represent context information, filter rules and service descriptions. Through a combination of semantic models and reasoning, context can be semantically filtered and service descriptions can be mapped to one another. The contributions of this chapter are threefold. First, three approaches to representing filter rules are proposed. The first builds upon our previous work [9] and uses pure OWL 2[1] (Web Ontology Language) constructs. It can be used in conjunction with a pure OWL reasoner. Additionally, two novel approaches using SWRL[2] (Semantic Web Rule Language) and Jena rules [10] as filters are proposed. Although the use of SWRL rules requires a more advanced reasoner, it offers increased expressiveness compared to pure OWL. Jena's expressiveness is comparable to that of SWRL, but does not have the added benefit of OWL inferencing. As such, Jena is expected to execute much faster. Second, we propose a novel matchmaking al-

[1]OWL 2 Web Ontology Language Document Overview: http://www.w3.org/TR/owl2-overview/
[2]A Semantic Web Rule Language Combining OWL and RuleML: http://www.w3.org/Submission/SWRL/

gorithm capable of determining the semantic compatibility between IOPE descriptions constructed of SWRL atoms. Additionally, we formally define these relationships in this chapter. Third, we determine the impact on efficiency of introducing semantics in the communications bus. We thoroughly evaluate performance and scalability, in terms of execution time, of the proposed algorithms and approaches and explore the trade-off between performance and expressive capabilities.

The remainder of this chapter is structured as follows. Section 6.2 discusses existing work on semantic publish/subscribe systems and semantic service matchmaking. Additionally, we thoroughly describe the functional differences with the approaches introduced in this chapter. Subsequently, Section 6.3 presents the SCB. A cloud computing use case is detailed in Section 6.4. The examples and evaluations presented throughout the chapter are based on this use case. Sections 6.5 and 6.6 respectively elaborate upon the internal details of the context dissemination and service matchmaking components. The implementation details and evaluation results of the designed prototype are discussed in Section 6.7. Finally, Section 6.8 concludes the chapter.

6.2 Related work

The context dissemination process pushes messages, containing context information, towards a set of interested subscribers. The solution proposed in this chapter is therefore related to existing work in the field of semantic publish/subscribe systems. Automated service matchmaking and discovery has long been a topic of interest in the context of the semantic web, and more specifically semantic web services. This section discusses the state of the art in these two fields and highlights the novel aspects of our approach.

6.2.1 Semantic publish/subscribe systems

A publish/subscribe system is a messaging paradigm in which a set of publishers loosely communicates with a set of subscribers. Publishers do not send their messages to specific receivers. Instead, subscribers express interest in specific types of information, and published messages that correspond to their interests are delivered to them. In recent years, publish/subscribe systems have evolved from static topic-based to dynamic content-driven systems. In topic-based systems, messages are published to specific topics, usually selected from a static and predefined hierarchy of keywords. In contrast, content-driven publish/subscribe systems allow subscribers to express interest in messages based on their actual content. Additionally, by augmenting the content with semantics, subscriptions can be created that take into account the actual meaning of the content.

Several types of semantic publish/subscribe systems have been proposed in

literature. These works propose systems based on RDF (Resource Description Framework) graph-matching [11–13], ontological inferencing [14, 15] and attribute-value pair matching [16, 17]. RDF graph-matching algorithms represent messages as sets of RDF triples, which can be modelled as directed labelled graphs. Subscriptions take the form of graph patterns, which are matched to the published messages by the graph-matching algorithm. In contrast, we propose an OWL-based approach, which allows new, non-asserted knowledge to be inferred during the message publishing process. The semantic publish/subscribe systems based on OWL inferencing are more closely related to the approach presented in this chapter. Subscriptions are represented as ontological classes, while messages are defined as class instances. An ontological reasoner is used to determine if a message instance satisfies the constraints of a subscription class. Although the ontological inferencing approach is also used by the SCB, it, in contrast to existing work, also supports SWRL and Jena rules as subscription filters. Such semantic rules greatly increase expressiveness through a set of built-ins (e.g., mathematical and comparison operations). Finally, the systems based on attribute-value pairs significantly limit the format that messages and thus information is allowed to take, which is not the case in our approach. The remainder of this section describes the introduced publish/subscribe systems in more detail.

Early work on semantic publish/subscribe systems was performed by Petrovic et al. [11, 18]. They argued that the traditional topic-based systems are incapable of matching semantically related concepts when determining the interest of subscribers in a specific message. To solve these problems they proposed G-ToPSS (Graph-based Toronto Publish/Subscribe System) [11], which uses an RDF graph-matching algorithm for relating subscriptions to messages. Subscriptions are represented as a set of 5-tuples, augmenting the subject and object of the RDF triple with optional constraints. These constraints could be boolean constraints for literal values (e.g., $=$ or $\geq$) and *is–a* constraints for RDF individuals. In our work, we aim to provide more powerful inferencing capabilities by using OWL-based inferencing engines, instead of a custom graph matching algorithm. Additionally, by supporting SWRL and Jena built-ins, we greatly increase expressiveness by introducing a wider array of constraints, such as mathematical operations and date comparison operators. Similar to the approach used by Petrovic et al., Wang et al. proposed a semantic publish/subscribe system, which uses RDF graph-matching [12, 19]. As such, the expressiveness of both approaches is also similar. However, Wang et al. also support regular expressions as constraints, in addition to the boolean operators supported by G-ToPSS. However, it shares G-ToPSS' limitations in expressive and inferencing power. More recently, the authors further applied their approach specifically to the RSS (Really Simple Syndication) document dissemination use-case [13]. In this recent work, OWL is used to represent the concept taxonomy, as opposed to DAML+OIL (DARPA Agent Markup

Language and Ontology Inference Layer) as used in their previous work.

The solution proposed by Li et al. more closely resembles the approach proposed in this chapter [14]. Subscriptions are represented as DAML+OIL classes, while messages are defined as class instances. In contrast, we use the more modern OWL, instead of DAML+OIL, to represent subscriptions. Additionally, we propose the use of SWRL and Jena rules as subscription filters, which greatly increases expressiveness. More recently, Skovronski and Chio adopted a similar method [15]. Messages are also represented by instances in the ontology. However, subscriptions take the form of SPARQL (SPARQL Protocol and RDF Query Language) queries. A SPARQL query consists of a set of RDF triple patterns, which are matched to the RDF triples in the ontology. Consequently, its inferencing and expressive capabilities share the limitations of other existing approaches [11, 12, 14].

Siena is a scalable event notification middleware [16]. Messages are represented as a set of typed attributes, comprised of a name, type and value. The supported types are limited to String, Long, Integer, Double and Boolean. Subscription filters support constraints on the values of attributes, including mathematical and string comparison operators. Keeney et al. proposed two extensions to the Siena system [17]. One extension provides ontological concepts as an additional message attribute type; this enables subsumption and equivalence relationships, along with type queries and arbitrary ontological subscription filters, to be applied. The second extension provides a bag type to be used that allows bag equivalence and filtering. Both of these extensions can be viewed as extending the semantic matching capabilities of Siena. In particular, the first extension looks at the semantics of the data and associated metadata contained in the message in addition to the contents of the message. This approach uses a set of subsumption operators (i.e., more specific (hyponyms), less specific (hypernyms), and equivalent concepts) as well as the ability to match on any ontological property, and then reasons on how subscriptions are related to published data. Our work is different in that we use a richer notion of semantic relatedness, and we are not limited to attribute-value pairs.

In summary, our goal is to improve existing work by greatly increasing inferencing power and subscription expressiveness. This can be achieved by using powerful OWL-based inferencing engines, instead of custom RDF graph-matching algorithms usually employed in existing semantic publish/subscribe systems, since OWL has more powerful features (e.g., a standard vocabulary, the ability to distinguish between classes and individuals enumeration and property restrictions) that enable it to perform more powerful inferencing than RDF. Additionally, we propose the use of SWRL and Jena rules for defining subscriptions, as they support a wide range of built-in operators which greatly increase expressiveness. These improvements should be achieved without sacrificing performance.

6.2.2 Semantic service matchmaking

Service matchmaking has long played an important role in the interaction between web services. Its goal is to find service descriptions that satisfy a set of functional criteria. Traditionally, keyword-based methods, like UDDI, have been proposed. However, they lead to low matching precision due to their lack of semantics [20]. More recently, web service descriptions have been augmented with semantics, which makes them machine-understandable and -readable. Most existing work on service matchmaking algorithms focusses on determining compatibility between inputs and outputs [21]. The matching of preconditions and effects, which is inherently more complex, has received relatively little attention [22].

OWLS-MX is a hybrid semantic web service matchmaker for OWL-S services [23, 24]. OWL-S is an ontological model for semantically describing services. It supports several logical matching relationships, such as exact, plug-in, subsume and subsumed-by. OWLS-MX also supports non-logic-based matching. However, we believe the latter cannot be used in an autonomic network management scenario, as compatibility on a logical level cannot be guaranteed. Additionally, the work focusses only on input and output matching. However, for AEs it is necessary to estimate the influence of services on their managed environment. Therefore, an approach that incorporates precondition and effect matching is required.

Recently, several semantic matchmaking algorithms that do take into account preconditions and effects have been proposed [20, 22, 25]. Shen et al. use description logics to describe service IOPEs [20]. A DL-reasoner can then be used to determine service compatibility. On one hand, using description logics allows them to formally prove the correctness of their algorithm. However, its expressiveness is limited compared to the SWRL-based approach introduced in this chapter. A SWRL-based approach was also proposed by Bener et al. [22]. They assign a score to every match, based on the type of subsumption relationship, semantic distance and synonym equivalence (using WordNet). The final score represents the semantic similarity between the service descriptions. However, a high score does not necessarily mean logical and functional compatibility. In contrast, our proposed algorithm returns the set of known logical and functional compatible service descriptions, which is more suitable for use by autonomic agents. Finally, Sbodio et al. use SPARQL for representing preconditions and effects [25]. However, their work focusses more on determining if the preconditions hold in the current state of the environment and relating the effects with the goals of the agent that is planning to execute the service. As such, this work complements ours.

In summary, most existing semantic service matchmaking algorithms focus on input and output matching only. In recent years, some algorithms have been proposed for the more complex precondition and effect matchmaking problem. However, we have identified several shortcomings in existing approaches (as described

above), such as limited expressive power or the inability to sufficiently capture functional compatibility. Additionally, none of these matchmaking algorithms are capable of relating specific inputs and outputs to specific preconditions and effects. Supporting such relations significantly increases the accuracy of service descriptions and, consequently, of the matchmaking process. The algorithm proposed in this chapter supports such relations by way of SWRL variables.

6.3 Semantic communications bus architecture

Future networks will be managed by a large set of automated management components, which we refer to as autonomic elements (AEs). In order to achieve the network's management goals, they interact in loosely-coupled collaborations with each other, the network's resources and human operators. Although the latter is still responsible for governing the network through the specification of high-level business goals, the lower-level maintenance tasks and configurations are taken over by the AEs. This section describes the semantic communications bus (SCB) that orchestrates the interactions between AEs. The implementation details of the prototype are discussed in Section 6.7.

A detailed overview of the SCB's architecture is shown in Figure 6.2. It plays a central role in the interactions between AEs and network resources. It orchestrates the dissemination of context and the service matchmaking process. Its semantic capabilities stem from the *core ontologies* and *semantic reasoners* embedded within it. The ontologies provide a model for semantically representing the managed environment. There are no requirements about the specific concepts that should be present in these models. However, in order to achieve understanding between communicating entities, they should use (a subset of) the same ontologies. Additionally, the interaction complexity is limited by that of the models. As such, as more complex concepts are added to the ontologies, it becomes possible to model more complex interactions. Throughout the rest of this chapter, DENON-ng is used as a basis for the core ontologies [26]. DENON-ng is an ontological model based on the DEN-ng information model [27]. It can be used to represent the physical and logical state of the network and its resources, as well as the business goals and internal workings of the governing organisations. Section 6.4 describes how these core ontologies can be further extended to a specific problem domain. The semantic reasoners operate on top of the core ontologies. They can be used by the SCB's other components to perform semantic inferencing.

In a federated network management scenario, AEs are expected to communicate across the boundaries of management domains. This introduces additional challenges such as the need for a common communication model for interpreting and understanding exchanged information. In the proposed SCB architecture, this would mean that, the relevant subset of, the core ontologies should be shared

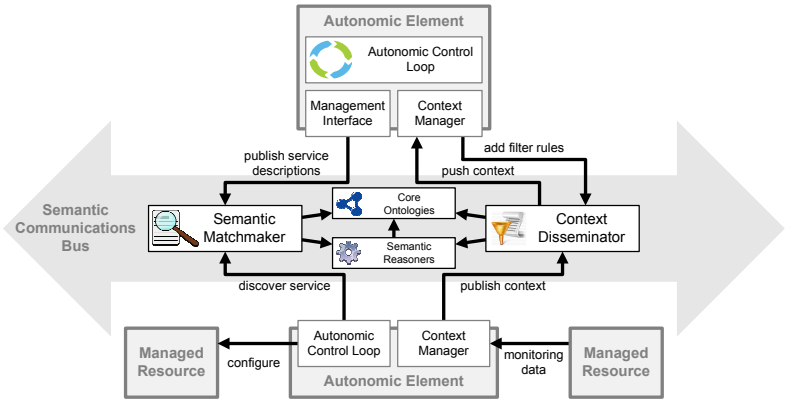


Figure 6.2: The SCB plays a central role in the interactions between AEs and network resources; its core ontologies are used in the matchmaking and context dissemination processes

across the communicating management domains. Such complex interoperability problems are outside the scope of this chapter, but have been thoroughly described in literature [28].

The figure shows the SCB as a single entity. However, in large-scale deployments the number of AEs might potentially become very large. The SCB could then be distributed among those AEs, increasing scalability by letting them all take part in the semantic reasoning processes. For example, structuring AEs hierarchically has been shown to significantly improve scalability and reduce the amount of context that needs to be exchanged [6]. In such a scenario, the bulk of context exchange takes place between parent and child AEs. Every AE thus only needs to register its filter rules with the SCB component of its children, significantly reducing the number of filter rules and amount of context that needs to be processed by every SCB semantic reasoner instance. Nevertheless, the remainder of this chapter focusses on the semantic reasoning algorithms of the SCB and no assumptions are thus made regarding its structure and deployment.

In addition to the core ontologies, two other components play an important role in the interactions between AEs and network resources; the *context disseminator* and the *service matchmaker*. The context disseminator is responsible for routing published context information to the AEs that have expressed interest in it. An AE may express interest in a specific type of context by creating a corresponding filter rule and registering it with the context disseminator. When an AE publishes context information, for example gathered through a monitoring probe on a managed resource, an ontological reasoner is used to match this information to the registered filters. Section 6.5 explains how such filter rules can be expressed and how they are matched with published context. The service matchmaker consists of two

components. The *service model repository* stores service descriptions of management functions offered by AEs or network resources. The *matchmaking algorithm* maps these offered service descriptions to descriptions of requested management services. It is thus responsible for the service matchmaking process. The definition of service descriptions and the internal workings of the matchmaking algorithm are further discussed in Section 6.6.

6.4 Use case: cloud infrastructure management

In this section, a cloud infrastructure management use case is described. The examples given throughout the following sections will be based on this use case. Additionally, the evaluation scenario used in Section 6.7 is based on it. First, the use case is described in more detail. Subsequently, the ontological classes of the core ontologies, relevant to this use case, are presented.

6.4.1 Scenario description

In the considered scenario, a set of AEs are responsible for managing a cloud computing infrastructure. The infrastructure consists of a set of servers, offered to service providers as virtualized resources. As such, every server contains a set of elastic virtual machines (VMs). Through Service Level Agreements (SLAs), service providers indicate the resource requirements of their services. The cloud provider makes sure the necessary amount of resources are provisioned within the bounds of the SLA. Cloud computing enables Service Providers and Enterprises to reduce both capital (CAPEX) and operational expenditure (OPEX) through hard- and software consolidation and automating business processes, respectively. In addition, the dangers of under- and over-provisioning are replaced by on-demand resource allocation that adjusts to varying resource and service consumption patterns.

The AEs are responsible for provisioning resources to the VMs and managing the state of the physical servers themselves. Therefore, they monitor resource availability on the servers, and resource consumption by the services. They make sure that all VMs receive the necessary resources within the bounds of the SLAs. These SLAs are represented as a set of policies.

6.4.2 Ontological concepts

As previously stated, there is no set of rules that determines what information should be present in the core ontologies of the SCB. In this section we describe a minimal set of classes that can be used to semantically describe the context and

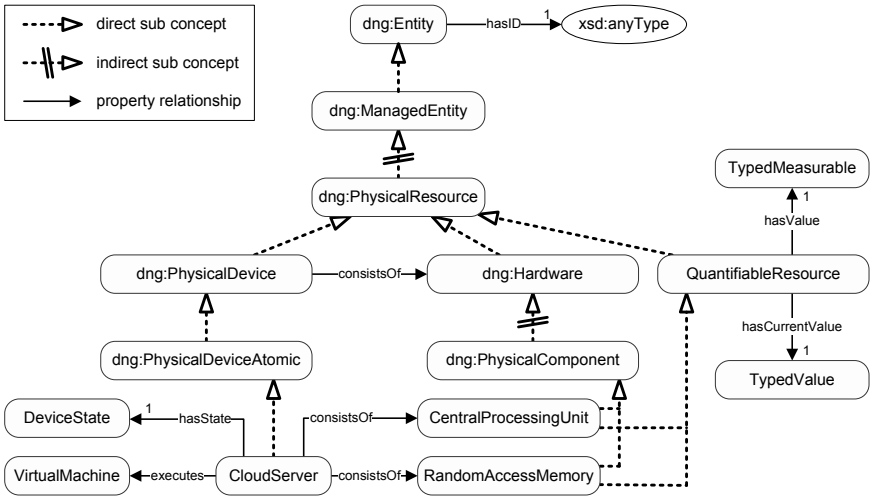


Figure 6.3: An overview of classes representing the physical resources of the cloud management use case; they are connected to the logical resources through the VirtualMachine *class*

management functions of the cloud infrastructure management use case. For clarity, the classes are shown in three different figures. Figure 6.3 presents the physical resources, such as servers and hardware components. The logical resources, including cloud services, requests and virtual machines are depicted in Figure 6.4. Finally, Figure 6.5 shows how measurements, such as CPU values, are represented. The core ontology is based on the DENON-ng ontology. Classes originating from DENON-ng are prefixed with "*dng:*". Additionally, the prefix "*temporal:*" is used for classes from the SWRL temporal ontology [29], which we use for representing time.

The physical resources shown in Figure 6.3 are all subclasses of *dng:Entity*. An entity is any physical or logical object that is important to the managed environment in some way. The *dng:PhysicalResource* class is an indirect subclass of *dng:Entity* and represents a managed piece of hardware. The different types of hardware components are represented by its subclasses. For example, *dng: PhysicalDevice* is the aggregation of the hardware components that make up a device. It enables management of the device as a whole. As such, it has a *consistsOf* relationship with *dng:Hardware* as its range. This last class represents the physical components that make up the device. Its *dng:PhysicalComponent* subclass serves as a super type for hardware that resides inside equipment and cannot be used as a stand-alone object.

In addition to these DENON-ng classes, we have defined several physical classes used to model the Cloud Computing use-case. The *CloudServer* class repre-

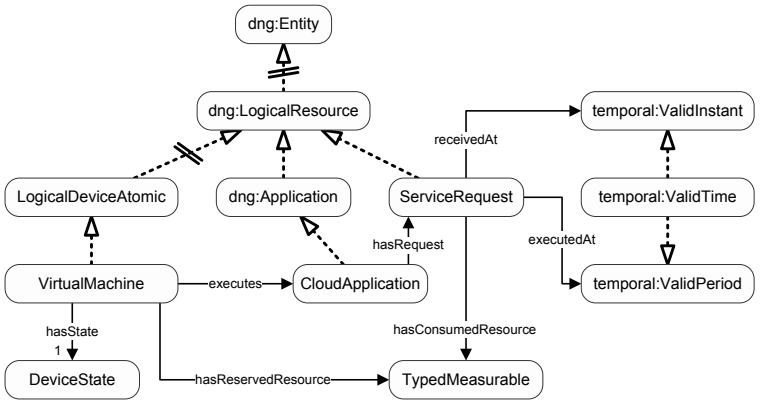


Figure 6.4: An overview of classes representing the logical resources of the cloud management use case

sents a physical server in the cloud datacenter. It has a *DeviceState*, which is one of TurnedOn, TurnedOff, Hibernating, Suspended or Unknown. It can be used to determine if a server is currently ready to receive service requests or not. Additionally, it hosts a set of *VirtualMachine*s. A cloud server consists of many different hardware components. However, our model focusses on two of these that will be used to monitor the resource consumption of the servers. It is easily extensible to include additional entities as necessary. The *CentralProcessingUnit* class represents a CPU core, while *RandomAccessMemory* denotes a memory module. Both these physical components depict physical resources that can be consumed by the software running on the servers. Therefore, they are also subclasses of *QuantifiableResource*. This is a resource that has a set of values associated with it. Such a set of values is represented as a *Measurable*. Additionally, the last measured value is explicitly defined through the *hasCurrentValue* relationship. The *Measurable* and *Value* classes are further explained in Figure 6.5.

The physical and logical resources are linked through the *VirtualMachine* classes. This and other logical resources are depicted in Figure 6.4. Every cloud server is capable of hosting a set of virtual machines. They represent elastic virtual servers that can be reserved and used by the customers of the cloud infrastructure for hosting their services. Just as a physical server, it has a *DeviceState*. Additionally, it has a set of reserved resources associated with it, which are represented by the *TypedMeasurable* class. Finally, services may be executed from the virtual machine, represented by the *CloudApplication* class. The *ServiceRequest* class models the client requests sent to the different cloud applications. Every request has an associated receive time and execution interval. Additionally requests consume resources, which are depicted by the *TypedMeasurable* class.

Management components often make decisions based on the current, past or

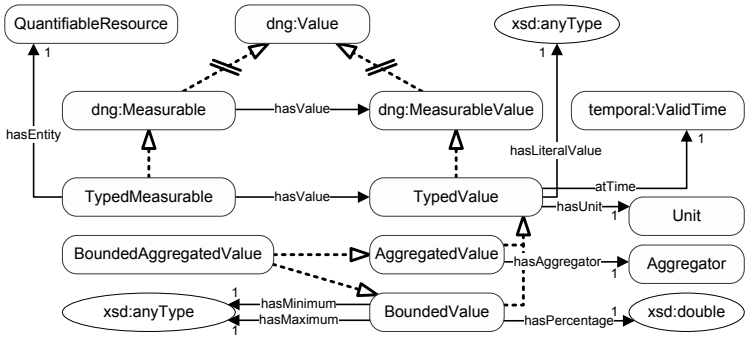


Figure 6.5: An overview of the classes representing measured values

expected future load of specific resources. As such, a mechanism is needed for modelling the load of hardware components, such as CPU and memory modules. Our proposed model, as shown in Figure 6.5, is based on the DENONng *dng:Measurable* and *dng:MeasurableValue* classes. A measurable categorizes a type of value that can be measured. The actual values themselves are represented by the *dng:MeasurableValue* class. A measurable thus depicts the type and groups together several measurable values. In addition to these standard DENONng types, we have defined our own measurable subclasses. The *TypedMeasurable* represents a measurable that is attached to a quantifiable resource. Additionally, it has a unit of measurement (e.g., Hertz, Megabyte), which can be accessed through the *TypedValue* class. The typed value itself also has a literal attached to it, which represents the actual value. Finally, there is also a time instant or period at which the measurement was taken attached to the typed values. The *TypedValue* class has several subclasses further differentiating between different types of measured values. An *AggregatedValue* depicts a measurement that was aggregated over a certain time interval, such as an average or a maximum. A *BoundedValue* has a minimum, maximum and percentage attached to it. Measurements in a network management scenario often are of this type. For example, a CPU load value is constrained by a minimum of 0 and a maximum denoting the CPU's total capacity. Finally, the *BoundedAggregatedValue* class combines the characteristics of *BoundedValue* and *AggregatedValue*. Note that in line with these value types, *BoundedMeasurable*, *AggregatedMeasurable* and *BoundedAggregatedMeasurable* also exist. However, these have been omitted from Figure 6.5 for clarity.

6.5 Semantic context dissemination

An important aspect of the interaction between management components is the exchange and dissemination of context information. The AEs monitor their set of

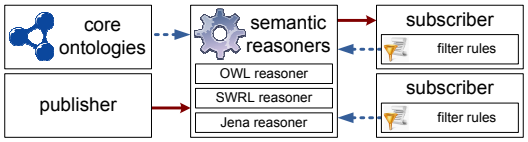


Figure 6.6: A detailed overview of the context dissemination process and the involved actors

managed resources and use the gathered information to create a view on the state of the managed environment. This context information is thus forwarded both from resources to AEs and between AEs themselves. In this section, we present three alternative approaches to modelling filter rules. Additionally, this section describes how these filter rules are matched with published context.

The SCB allows management components to register interest in specific topics, by way of filter rules. The semantic reasoners are an integral part of internal mechanism used by the SCB to match filter rules with the context that is published onto the bus. If a published unit of context matches at least one of the filter rules of an AE, that context is forwarded to the AE. The use of such a publish/subscribe system greatly reduces the amount of context information that is forwarded to AEs, preventing them from becoming swamped with useless information. Additionally, by introducing semantics into the filter rules, management components can indicate interest based on the meaning of data that makes up the context received by an AE, allowing them to reason about what information would be relevant in face of the tasks they perform. Figure 6.6 further clarifies the context dissemination process. Context publishers can freely forward data to the SCB. The SCB uses the semantic reasoners to determine which subscribers are interested in the context contained within the message. The reasoners use the core ontologies to check if the message satisfies at least one filter rule defined by the subscriber. If it does, the message is forwarded.

In order to be able to semantically reason on messages and filter rules, the core ontologies need to be augmented with a *Message* class. As DENON-ng already contains the *dng:Message* class, we use this as a basis for our model. A message represents a *dng:Event* with the addition of a payload. In turn an event is a type of *dng:Entity*. Additionally, a message has exactly one source and zero or more targets, which are both of type *dng:Entity*. When an AE publishes context information, it creates a message, which is an instance of the *dng:Message* class, and adds the context it wishes to publish as the message payload.

The three presented approaches differ in the way filter rules are defined. The first uses pure OWL constructs, while the second and third use SWRL and Jena rules, respectively. The approaches are presented in more detail throughout the rest of this section.

6.5.1 OWL filter rules

The OWL-based algorithm exploits the instance reasoning capabilities of the ontological reasoner. As stated, published context is represented as an instance of the *dng:Message* ontology class. Filter rules, on the other hand, are represented by way of subclasses of the *dng:Message* class. As such, context can be matched to a filter rule by asking the OWL reasoner if the message instance belongs to the *dng:Message* subclass defined by the filter rule.

More specifically, filter rules are defined as an equivalent class of a conjunction of OWL class expressions. A class defined in such a way is called a *defined class*. The conjunction can be seen as a set of *necessary and sufficient* conditions to which instances must adhere in order to belong to the class. This means that the reasoner is capable of inferring that an instance belongs to a specific *defined class*, even if this information was not asserted. This is not possible with a *primitive class*, which is defined using subclass axioms instead of an equivalent class axiom. Additionally, the SCB asserts that the filter rule class is a subclass of *dng:Message*. The reasoner can then be used to check if the filter rule class is satisfiable. If it is unsatisfiable, it means that it is not a valid subclass of *dng:Message* and the filter rule is removed from the ontology. If it is satisfiable, then the subscription operation finishes successfully.

In order to further clarify the use of OWL filter rules, we give some examples based on the use-case . All OWL examples given throughout this section use the OWL Manchester syntax[3]. A simple rule stating that an AE is interested in all messages containing information about servers can be defined as follows:

dng:Message **and** hasPayload **some** CloudServer

As previously stated, the filter rule is a conjunction of class expressions. The first expression is the obligatory subclass assertion, stating that the newly defined filter rule class is a subclass of *dng:Message*. The second expression states that the filter rule should only trigger if the message has at least one payload of type *CloudServer*.

Using OWL 2 specific constructs, such as datatype restrictions, more complex and expressive filter rules can easily be defined. For example, an AE that is responsible for resource allocation of virtual machines could be interested in all types of quantifiable resources with a load higher than 95%, in order to be able to react before those resources become overloaded. This filter rule can be defined as follows:

dng:Message **and** hasPayload **some** (QuantifiableResource
and hasCurrentValue **some** (BoundedValue
and hasPercentage **some** double[$>$ 95.0]))

[3]http://www.w3.org/TR/owl2-manchester-syntax/

Again, the first expression states that filter rule class should be a subclass of *dng:Message*. The second class expression again targets the message payload. However, in contrast to the first example, a more detailed payload description is given. The payload should abide to several restrictions. First, it must be of type *QuantifiableResource*. Second, its current value must be of type *BoundedValue*. Third, the current percentage of this value must be of type double and higher than 95%.

6.5.2 SWRL filter rules

In the previous section, it was shown how filter rules can be specified in the form of OWL class descriptions. The advantage of such an approach is that an OWL-only reasoner can be used to infer if a filter rule matches a message instance. However, in order to further increase expressiveness, this section introduces an alternative approach, using SWRL rules as filters. The advantage of this approach is that it combines OWL inferencing with the increased expressiveness of SWRL. Obviously, in order to apply this approach, a SWRL-capable reasoner is required.

A SWRL rule consists of an antecedent and a consequent, each consisting of zero or more atoms. Multiple atoms are always treated as a conjunction. Semantically, if all the conditions of the antecedent hold, then the conditions specified in the consequent must also hold. Conceptually, our approach works as follows. The filter rule itself makes up the antecedent of the SWRL rule. The atoms in the antecedent thus specify terms that a message must match in order to be forwarded. On the other hand, the consequent is automatically created by the SCB. Every subscriber that registers with the SCB is allotted a unique asserted subclass of *dng:Message*. The consequent adds the message instance to this uniquely defined class. Determining if a message should be forwarded to a specific subscriber thus comes down to triggering the SWRL rules and checking if that message is an individual of the unique class associated with the subscriber. For example, a complete SWRL rule associated with a simple filter rule that accepts all messages that contain information about a server, would look as follows:

$$\text{dng:Message}(?m) \wedge \text{CloudServer}(?s) \wedge \text{hasPayload}(?m, ?s) \Rightarrow \text{SubscriberMessage}(?m)$$

The antecedent consists of three atoms. The first is, in line with the OWL filter rules, obligatory. It identifies the variable that refers to the message instance. The second atom defines a placeholder variable for a *CloudServer*. Finally, the third atom states that the message *?m* should have a payload of type server. The consequent asserts that, if the atoms in the antecedent hold, the message *?m* belongs to the class *SubscriberMessage*. The class specified in the consequent should be the unique class associated with the subscriber that registered the rule. Note that

this consequent is automatically generated by the SCB. The subscriber thus merely needs to define the atoms of the antecedent.

The antecedent atoms of the filter rule that matches all messages that contain information about a quantifiable resource with a load higher than 95% can be defined as follows:

dng:Message(?m) ∧ QuantifiableResource(?r) ∧ BoundedValue(?v)
∧ hasPayload(?m, ?r) ∧ hasCurrentValue(?r, ?v) ∧
hasPercentage(?v, ?p) ∧ swrlb:greaterThan(?p, 95.0)

This more complex example consists of 7 atoms. The first is again the obligatory message class axiom. The second and third atom identify the variables associated with the quantifiable resource and the bounded value associated with this resource. The fourth atom states that the resource *?r* should be a payload of message *?m*. The fifth atom links *?r* to its value *?v*. Finally, the last two atoms identify the percentage associated with the value as the variable *?p* and state that this percentage should be higher than 95%.

The last atom defined in the previous example is called a SWRL built-in atom. The expressive power of SWRL is greatly increased by these built-ins. Built-ins have been defined for comparison (as in the example above), mathematical operations (e.g., summation, division, power), strings (e.g., substring, concatenation) and dates. The operations concerning dates are facilitated through the SWRL temporal ontology [29].

6.5.3 Jena filter rules

The OWL and SWRL-based approaches are capable of inferring complex connections between concepts, using an OWL reasoner. However, this is computationally hard [30], leading to degraded performance. Therefore, we propose a third approach, based on Jena rules. Conceptually, it is similar to the SWRL-based approach. Nevertheless, it does not perform OWL-based inferencing, but applies the rules directly to the underlying RDF graph. This is expected to greatly increase performance, at the cost of decreased semantic inferencing capabilities.

Jena rules are applied as context filters in the same way as SWRL rules. The rule body, which corresponds to the SWRL antecedent, contains the actual filter. The head, which corresponds to the SWRL consequent, consists of a single atom that adds the message to the subscriber's message subclass. The example that admits all messages that contain as a payload an entity of type *CloudServer* is defined using Jena as follows:

(?m rdf:type dng:Message) (?s rdf:type CloudServer)
(?m hasPayload ?s)

The first two atoms state that variables *?m* and *?s* belong to the classes *dng:Message* and *CloudServer*. The *rdf:type* property thus corresponds to the SWRL class atom. The third atom connects the variables and states that the message should have a payload of type *CloudServer*. The more complex example that admits all messages about resources with a current load of 95% or higher, is defined as follows:

(?m rdf:type dng:Message) (?r rdf:type QuantifiableResource)
(?v rdf:type BoundedValue) (?m hasPayload ?r)
(?r hasCurrentValue ?v) (?v hasPercentage ?p) greaterThan(?p, 95)

As shown in the last atom, Jena also supports a set of built-ins, such as comparison and mathematical operators.

6.6 Semantic service matchmaking

AEs and managed resources may offer management services to other AEs across the network. However, a scalable matchmaking mechanism is needed for finding suitable candidate services to complete specific tasks. This section describes the service matchmaking algorithm, introduced in Section 6.3, in further detail. The algorithm is responsible for matching offered and requested service descriptions. Service descriptions consist of a set of inputs, outputs, preconditions and effects (IOPEs). The inputs represent the information that the service requires in order to perform its function. The outputs, on the other hand, represent the information that results from its execution. The preconditions and effects respectively represent the state that the environment should be in before the service execution starts, and the state it will be in after it finishes. The IOPEs are defined using SWRL atoms. SWRL was chosen instead of Jena, as it is compatible with existing service models, such as OWL-S. Inputs and outputs take the form of SWRL class or data range atoms, while preconditions and effects can be any type of SWRL atom. The algorithm uses the subsumption relationship between these SWRL atoms to determine potential matches between requested and offered service descriptions. Concept A is said to subsume concept B if B is a specialization of A, or in other words, if B is a subclass of A. Additionally, SWRL supports the use of variables. This makes it possible to link inputs and outputs to preconditions and effects, which would not be possible when using pure OWL. As the IOPEs are modelled using SWRL atoms, our algorithm is compatible with any semantic service description model that supports SWRL IOPEs, such as OWL-S[4].

As depicted in Figure 6.7, the matchmaking algorithm operates in five steps when checking the compatibility between an offered and requested service description. First, it checks the validity of the subsumption relationship between the

[4]http://www.w3.org/Submission/OWL-S/

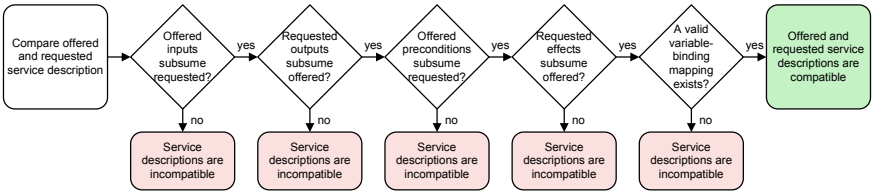


Figure 6.7: A flowchart depicting the five steps of the semantic matchmaking algorithm

offered and requested input atoms. This process is subsequently repeated for the output, precondition and effect atoms. Finally, the algorithm checks if the variable bindings of the offered service correctly map to those of the requested service. If during the execution any of the five steps fail, the algorithm knows that the service descriptions are incompatible, and it can skip to the next offered service description. The remainder of this section describes these five steps in more detail. However, as the first four steps of the algorithm rely on the definition of subsumption between SWRL atoms, we first formally define the subsumption relationship between the different SWRL atoms.

6.6.1 Subsumption relationships

The SWRL recommendation does not formally define the subsumption relationship between SWRL atoms. Therefore, we first present a formal definition of this relationship. SWRL defines seven atom types: class, data range, object property, data property, sameAs, differentFrom and built-ins. Note that an atom can only subsume atoms of the same type.

A class atom $C(x)$ subsumes a class atom $D(y)$ if D is an equivalent class or subclass of C, and if x and y are matching arguments. An argument can be a SWRL variable, OWL individual or OWL data value. Two arguments x and y match if either they are both SWRL variables, they refer to the same OWL individual, or they refer to the same OWL data value. The equivalence and subclass relationships between C and D can be checked through applying an ontological reasoner on the core ontologies.

The subsumption relationship between object and data property atoms is similar to that between class atoms. An object property atom $P(x_1, x_2)$ subsumes another object property atom $Q(y_1, y_2)$, if Q is an equivalent property or subproperty of P and if arguments x_1 and y_1 match and x_2 and y_2 match. Alternatively, if P and Q are symmetric, then x_1 may match y_2 and x_2 may match y_1. The subsumption relationship between data property atoms is defined in the same way, with the minor difference that data properties cannot be symmetric.

The sameAs atoms cannot subsume one another, as it is not possible to define a subconcept relationship between them. However, two sameAs atoms can be

considered equivalent, which means that subsumption holds in both directions. Two sameAs atoms *sameAs(x_1,x_2)* and *sameAs(y_1,y_2)* are considered equivalent if x_1 matches y_1 and x_2 matches y_2 or x_1 matches y_2 and x_2 matches y_1. The equivalence of *differentFrom* atoms is defined in the same way.

The subsumption between data range atoms is somewhat more complex. OWL supports several types of data ranges; datatypes, intersections, unions, complements, data oneOf and datatype restrictions. A data complement restriction $\neg C(x)$ subsumes a data complement restriction $\neg D(y)$ if $D(y)$ subsumes $C(x)$. A datatype atom $C(x)$ subsumes a datatype atom $D(y)$ if D is an equivalent or sub-datatype of C and x and y match. To determine sub-datatypes, we used the XSD datatype hierarchy specified in the W3C recommendation "XML Schema: Datatypes"[5]. A data one of atom $C(x)$ subsumes a data one of atom $D(x)$ if C is a superset of D (i.e., C contains all the values in D) and x and y match. Furthermore, datatype restrictions, intersections and unions are not supported by our algorithm. Note that the semantics of datatype restrictions can be achieved using the built-in atoms for comparison. Therefore, not supporting datatype restrictions does not reduce expressiveness.

Finally, SWRL defines a wide range of built-in atoms. In this chapter, we define subsumption relationships only for the comparison built-ins. Subsumption relationships for the other SWRL built-ins can be defined in a similar fashion. A *swrlb:equal(x_1,x_2)* built-in atom subsumes another *swrlb:equal(y_1,y_2)* built-in atom if x_1 matches y_1 and x_2 matches y_2 or x_1 matches y_2 and x_2 matches y_1. The subsumption relationship of the *swrlb:notEqual* built-in is defined identically. The subsumption of the other comparison built-ins is more complex. More specifically, *swrlb:greaterThan*, *swrlb:greaterThanOrEqual*, *swrlb:lessThan* and *swrlb:lessThanOrEqual* can all subsume each other. As *swrlb:lessThan(x_1,x_2)* and *swrlb:lessThanOrEqual(x_1,x_2)* can be translated into *swrlb:greaterThan(x_2,x_1)* and *swrlb:greaterThanOrEqual(x_2,x_1)*, they can be ignored without loss of expressiveness. Intuitively, one of these atoms subsumes another if the range of possible values specified by the first contains the range specified by the second atom. Formally, an atom *swrlb:greaterThan(x_1,x_2)* subsumes an atom *swrlb:greaterThan(y_1,y_2)* if one of the following conditions holds for the arguments:

- All four arguments are SWRL variables
- x_1 and y_1 are SWRL variables, x_2 and y_2 are OWL data values and $x_2 \leq y_2$
- x_2 and y_2 are SWRL variables, x_1 and y_1 are OWL data values and $x_1 \geq y_1$

In all other cases, the subsumption does not hold. A *swrlb:greaterThanOrEqual* built-in atom subsumes a *swrlb:greaterThan* or *swrlb:greaterThanOrEqual* atom

[5]http://www.w3.org/TR/xmlschema-2/

under the same conditions. Finally, to check if a *swrlb:greaterThan* atom subsumes a *swrlb:greaterThanOrEqual* atom, the conditions are slightly different. More specifically, in the second and third condition, $x_2 \leq y_2$ and $x_1 \geq y_1$ respectively become $x_2 < y_2$ and $x_1 > y_1$.

Let us further clarify by way of an example: *swrlb:greaterThanOrEqual(*x*,*5*)* specifies that $x \in [5, \infty[$, while *swrlb:greaterThan(*y*,*6*)* specifies that $y \in]6, \infty[$. Intuitively it is apparent that the first atom subsumes the second, as the second value range is entirely contained within the first. Additionally, the atoms adhere to the second condition above, which validates the subsumption relationship.

IOPEs usually consists of a set of SWRL atoms. Therefore, this section is concluded with a definition of the subsumption relationship between SWRL atom sets. Additionally, we propose an algorithm to check this relationship. Assume we want to check if a set of SWRL atoms $A = \{a_1, a_2, ..., a_n\}$ subsumes a set of SWRL atoms $B = \{b_1, b_2, ..., b_m\}$. The set A subsumes the set B if a mapping exists between the atoms in A and B, where every atom in A is mapped to exactly one atom in B and every atom in B is linked to at most one atom in A. An atom a_i of A can be mapped to an atom b_j of B if a_i subsumes b_j. The algorithm finds all valid mappings, because in the final step of the service matchmaking process they are needed to check variable binding matches (cfr. Section 6.6.4). The algorithm does this as follows. First, it constructs a bipartite graph, with all atoms of A as left vertices and those of B as right vertices. An edge is added to the graph between every atom a_i of A and b_j of B, if a_i subsumes b_j. If a left vertex has a degree of 0, no valid mappings exist and the algorithm finishes unsuccessfully. Otherwise, the algorithm constructs all valid mappings. In the context of bipartite graphs, finding such valid mappings is called the *maximum bipartite matching problem*. Existing algorithms, such as the one proposed in [31], can be leveraged to solve this problem.

6.6.2 Inputs & outputs

The inputs and outputs defined in a service description represent the arguments and return values of the associated management function, respectively. As such, both the inputs and outputs are defined as SWRL class and data range atoms, which take the form $A(x)$. Here, A is an OWL class or data range expression, and x is a SWRL variable, an OWL individual or an OWL data value. If a SWRL variable is used in an input or output, the same variable can be reused in a precondition or effect in order to semantically link them together. For example, a management function could exist that activates a hibernating server. This function would have a server as input, a precondition stating that a server should be in hibernation and an effect stating that a server should be turned on. By reusing the same variable in the input, precondition and effect, they would be semantically linked and the reasoner

would know all three refer to the same server instance.

The goal of the matchmaking algorithm is to find offered service descriptions of which the inputs and outputs are compatible with those of the requested service description. The inputs and outputs have slightly different semantics attached to them in the requested and offered service descriptions. In a requested service description, the inputs represent the information that the requester is willing to provide, while the outputs represent the minimum set of information that the requester wants to receive. On the other hand, the offered inputs describe the minimum set of information that the executor requires in order to successfully execute the management function, while the offered outputs represent the exact information that will be returned. As such, an offered set of inputs I_o is compatible with a requested set I_r if I_o subsumes I_r. This can be checked using the subsumption definition and algorithm for SWRL atom sets introduced in Section 6.6.1. The reason the offered inputs need to subsume the requested ones is explained as follows. If a service is requested that takes as input a specific concept (e.g., server) and one is offered that takes as input a more broad concept (e.g., device), then the offered service can potentially be used to perform the requested task, as every instance of the more specific concept is also an instance of the broader concept. For example, a broad management function that is capable of turning on any type of device, can be used by an AE that wants to turn on a server. Additionally, this means that the set I_r may contain additional inputs not defined in I_o. However, all inputs in I_o must have a match in I_r.

For outputs, the subsumption relationship is inversed. In other words, an offered set of outputs O_o is compatible with a requested set of outputs O_r if O_r subsumes O_o. Intuitively, this is explained as follows. If an AE requests a function that returns output about a concept, it might also be interested in a function that returns output about one of its subconcepts. Additionally, this means that the offered service may return more outputs than requested.

The matchmaking algorithm compares the compatibility of inputs and outputs between every offered service description and the requested service description. As previously stated, checking the subsumption between offered and requested inputs or outputs results in a set of maximum bipartite matchings. If the returned set of matchings of the inputs or outputs is empty, the offered and requested services are incompatible and the matchmaker no longer needs to check preconditions, effects and variable bindings for this offered service.

6.6.3 Preconditions & effects

The preconditions of a service description describe the conditions that must be valid before the management function can be executed, while the effects represent the conditions that will hold after the execution finishes. Similarly to inputs and

outputs, the matchmaking algorithm checks the compatibility between offered and requested preconditions and effects. Again there is a slight semantic difference between preconditions and effects in the requested and offered service descriptions. A requested set of preconditions actually defines the current state of the environment, or at least the state the environment will be in when the requester is planning to execute the management function. A set of requested effects represent the state the requester expects the environment to be in after the execution finishes. As such the requested effects are obligations the executor must adhere to. On the other hand, the set of offered preconditions define the state of the environment that must be valid before the management function can be executed, while the executor ensures that the set of offered effects will be valid after execution. Much like inputs, preconditions are compatible if the offered preconditions subsume the requested ones, which also means that the requested service description may contain more preconditions than the offered one. In contrast, the offered and requested effects are compatible if the requested subsume the offered effects. Obviously, checking the subsumption relationship for precondition and effect sets is more complex than for inputs and outputs, as they may contain any type of SWRL atom, as opposed to only class and data range atoms.

In line with inputs and outputs, the matchmaking process between preconditions and effects also yields two sets of maximum bipartite matchings. Again, if either set is empty, the offered and requested services are incompatible. Otherwise, the matchmaker checks compatibility between the variable bindings of both service descriptions.

6.6.4 Variable binding matches

If the matchmaking algorithms finds at least one maximum bipartite matching for the inputs, outputs, preconditions and effects between the offered and requested service descriptions, it will check if there exists a 4-tuple of matchings (one of inputs, outputs, preconditions and effects each), that has a compatible variable binding. If this is the case, the offered and requested service descriptions are considered compatible.

As stated, the variable binding matchmaking algorithm takes as input four maximum bipartite matchings; one of inputs, outputs, preconditions and effects. First, the algorithm creates a variable mapping of the offered service, which maps all SWRL variables on the set of SWRL atoms that contain this variable and are present in the IOPEs of the offered service description. For every SWRL variable x of the offered service description, this results in the set $A_x = \{a_1, a_2, ..., a_n\}$. Using the four selected maximum bipartite matchings, the SWRL atom b_i of the requested service that maps to every a_i of A_x can be determined. Note that some a_i from the outputs and effects may not have an associated b_i; in this case, they can

be ignored. Subsequently, the algorithm checks if all atoms b_i contain the same variable y as an argument on the correct position. For most SWRL atom types the correct position is the position of x in the argument list of the corresponding atom a_i. However, some atom types, such as the built-in atom *swrlb:equal*, represent a symmetric relationship. As such, if an atom a_i, and correspondingly b_i, represents a symmetric relationship, the variable y in b_i does not necessarily need to be on the same position as x in a_i. If all b_i corresponding to the atoms in A_x contain the same variable y on the correct position, then the variable bindings between the requested and offered service match for variable x. If this is the case for all variables that occur in the offered service description for the four selected maximum bipartite matchings, then this 4-tuple of matchings is considered valid and the offered and requested service descriptions are fully compatible. Otherwise, the algorithm selects a different combination of matchings and retries. This process is repeated until a valid 4-tuple is found, or all possible combinations have been depleted. If no valid 4-tuple exists, then the descriptions are incompatible.

6.6.5 Illustrative examples

In order to further clarify the described matchmaking process, this section is concluded with some example service descriptions. Additionally, an example is given that clarifies the matchmaking process between offered and requested services.

In a cloud computing environment, the amount of reserved and consumed resources varies greatly over time. In order to reduce energy consumption, idle servers might be put in hibernation mode when the load on the infrastructure is low. As such, an AE might exist that offers management functions to turn on or off devices. Another AE, which executes a management algorithm that decides when to put servers in or out of hibernation, requires this function to perform its tasks. It could, for example, request a management service to activate specific hibernating servers. An example semantic description of the offered function to turn on devices and the requested function to activate servers is shown in Table 6.1.

The service matchmaking algorithm checks if the offered and requested descriptions match as follows. First, the algorithm constructs the bipartite graph mapping the inputs. As both descriptions contain only a single input SWRL class atom, the algorithm only needs to check if *dng:PhysicalDevice(?d)* subsumes *CloudServer(?s)*. The ontological model in Figure 6.3 shows that *CloudServer* is indeed an indirect subclass of *dng:PhysicalDevice*. Additionally, both $?d$ and $?s$ are SWRL variables, so the arguments of both atoms match. The two input atoms will thus be connected in the bipartite graph. For this very simple graph, with two vertices and one edge, the maximum bipartite matching equals the graph itself and it is thus stored by the algorithm. As neither service description contains any outputs, the second step of the algorithm is skipped. In the third step, the precon-

Table 6.1: An example of an offered and requested service description to turn on devices and activate servers respectively

IOPEs	Offered	Requested
inputs	dng:PhysicalDevice(?d)	CloudServer(?s)
outputs		
preconditions		hasState(?s, Hibernating)
effects	hasState(?d, TurnedOn)	hasState(?s, TurnedOn)

ditions are matched. Although the offered service description contains no preconditions and the requested does, both precondition sets are still compatible. More specifically, the set of offered preconditions still subsumes the set of requested preconditions, as the definition we proposed allows atoms in the subsumed set to have no match in the subsuming set. Intuitively, unmatched preconditions in the requested service are not a problem, as even under these conditions the offered service can still be executed. Subsequently, the match between effects is determined. Both service descriptions are compatible if the requested atom *hasState(?s, TurnedOn)* subsumes the offered *hasState(?d, TurnedOn)*. This is trivially clear, as both object property atoms refer to the same object property. Additionally, there is a match between the arguments, as in both cases the first argument is a SWRL variable and the second argument refers to the same *DeviceState* individual *TurnedOn*. The fifth and final step of the algorithm constitutes the matching of variable bindings. As previously stated, a map is created that links all SWRL variables in the offered service to the atoms in which they occur. In this case, there is only variable $?d$, which is linked to both of the atoms occurring in the service description. Subsequently, the algorithm iterates over all 4-tuples of IOPE maximum bipartite matchings. As there are no defined outputs, and the offered service defines no preconditions, the matchings of outputs and preconditions are empty. The other two matchings both contain a single edge, connecting *dng:PhysicalDevice(?d)* to *CloudServer(?s)* and *hasState(?d, TurnedOn)* to *hasState(?s, TurnedOn)*. As such, the algorithm needs to check if the atoms *CloudServer(?s)* and *hasState(?s, TurnedOn)* contain the same SWRL variable argument on the position of $?d$ in the corresponding atoms of the offered service. As they both refer to the variable $?s$ on this position, the variable bindings are compatible. The matchmaking algorithm thus concludes that the offered service description meets the requested requirements.

In conclusion of this section, a more complex service description is given to further illustrate the expressive power of the proposed SWRL-based approach. An important management issue in cloud computing environments is the allocation of physical server resources to virtual machines. As such, we formalize a service description of a management function that allows AEs to allocate resources to virtual machines. The details are shown in Table 6.2. The function takes three inputs,

Table 6.2: A service description of a management function that allows AEs to allocate physical resources to a virtual machine

IOPEs	SWRL atoms
inputs	VirtualMachine(?v), QuantifiableResource(?r), xsd:long(?i)
outputs	
preconditions	CloudServer(?s), executes(?s, ?v), hasState(?s, TurnedOn), consistsOf(?s, ?r), hasCurrentValue(?r, ?c), hasLiteralValue(?c, ?l), hasMaximum(?c, ?m), swrlb:substract(?d, ?m, ?l), swrlb:greaterThanOrEqual(?d, ?i)
effects	hasReservedResource(?v, ?t), hasEntity(?t, ?r), hasValue(?t, ?u), hasLiteralValue(?u, ?i)

the virtual machine for which the resources should be reserved, the physical resource that should be reserved and the exact amount of resources that should be reserved. It has no outputs, but does change the state of the environment and therefore has several preconditions and effects. Informally, the preconditions makes sure that the virtual machine and the reserved resource are actually on the same cloud server (atoms 1 and 2), the server is turned on (atom 3) and the requested amount of resources is actually available (atoms 4-9). The latter is done by using several built-in atoms. The current $?l$ and maximum $?m$ used resources of $?r$ are defined and substracted from one another. This results in the variable $?d$ containing the currently available resources of $?r$, which must be greater than or equal to $?i$. Finally, the effects state that after the execution of the function finishes, the requested amount of resources will be reserved for the specified virtual machine.

6.7 Evaluation & results

Semantic reasoning and ontologies facilitate the understanding and interpretation of context information and management functions by autonomic elements. However, this comes at the cost of performance degradation, as ontological reasoning is widely known to scale poorly in terms of ontology size and expressiveness. Specifically, it has been proven that the more expressive OWL 2 profiles have an NP-complete (or worse) reasoning complexity[6]. In this section, we explore the effects of semantic reasoning on the SCB's performance, as a function of several parameters. As a performance metric, the total *reasoning time* is used, which corresponds to the total execution time of the algorithms and excludes any other delays such as network latency.

The remainder of this section considers performance of three aspects of the

[6]http://www.w3.org/TR/owl2-profiles/#Computational_Properties

SCB in separate subsections. These subsections evaluate performance of creating filter rule subscriptions, context publication and the service matchmaking algorithm, respectively. However, first follows a brief description of the implemented prototype and the evaluation setup.

6.7.1 Implementation & evaluation set-up

A prototype implementation of the context dissemination and service matchmaking components was created specifically for this evaluation. The prototype was built in Java, based on the Pellet OWL 2 reasoner version 2.1.1[7] and OWL-API version 3.0.0[8]. In addition to OWL 2 reasoning, Pellet supports DL-safe SWRL rules [32]. Jena rules are supported through Jena's own built-in rule reasoner, of which version 2.6.2[9] was used. All evaluations were performed using a core ontology containing a subset of DENON-ng, the complete SWRL temporal ontology and our own cloud computing model (as described in Section 6.4). In total, the core ontology consists of 160 classes, 37 object properties, 21 data properties, 19 individuals and a total of 490 asserted axioms. The used DENON-ng subset contains 135 classes in total, including all higher level classes of the dng:Entity subtree and the complete dng:Resource substree. For simplicity, some class relationships, not needed for the evaluated use-case, were omitted, including those of the policy model, identity and behavioural aspects. Finally, the tests were performed on a server with two dual-core AMD Opteron 2 Ghz processors and 4 GB memory, running Debian 5.0 and Linux kernel 2.6.30.

The evaluation setup is based on the use case described in Section 6.4. For the evaluation of the context dissemination component, filter rules are used that admit messages that contain information about *QuantifiableResource*s with a current load higher than a randomly generated percentage. This filter is thus similar to the example used throughout Section 6.5, which admits messages about *QuantifiableResource*s with a current load higher than 95%. The messages used in the evaluation similarly contain context information about one or more *QuantifiableResource*s with a randomly generated current load. The complexity of filter rules and messages is expressed as the number of payloads. A single payload contains information about one *QuantifiableResource*. In the evaluation of the semantic matchmaking algorithm, service descriptions similar to the one depicted in Table 6.2 are used. It describes a management function that allows AEs to reserve resources for a specific *VirtualMachine*.

[7] http://clarkparsia.com/pellet
[8] http://owlapi.sourceforge.net/
[9] http://jena.sourceforge.net/

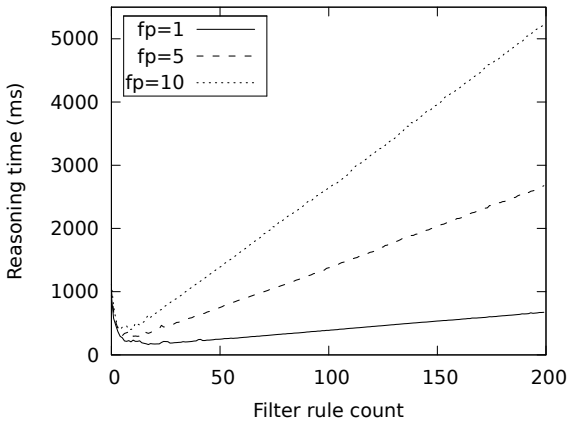


Figure 6.8: The evolution of total reasoning time as more OWL filter rules are added to the SCB

6.7.2 Filter rule subscription

The SCB's context dissemination component operates as a publish/subscribe system. It thus consists of two steps, subscription through filter rules and publication of messages. This section considers scalability in terms of reasoning time of the subscription step. In the experiment, 200 filter rules were added sequentially to the SCB. The entire process was repeated 30 times. The results depicted in Figures 6.8, 6.9 and 6.10 show the average reasoning time for adding each of these 200 filters, averaged over the 30 iterations. More specifically, every value x on the x-axis shows the time (in milliseconds) it took to add a filter rule to the SCB when $x - 1$ filter rules have already been added. The figure depicts reasoning time as a function of the number of filter rules for the three proposed filter rule approaches (i.e., OWL, SWRL and Jena). All graphs depict results for filters with an increasing number of payloads (*fp*). The more payloads a filter rule contains, the more complex it becomes.

As described in Section 6.5.1, the use of OWL filter rules allows their satisfiability and consistency to be checked by the OWL reasoner. On the other hand, this is not possible when using SWRL or Jena filter rules. The depicted results demonstrate that performing such satisfiability and consistency checks severely impacts scalability. Figure 6.8 shows that, for OWL filters, the subscription time increases as the number of filters in the SCB increases. Additionally, the effect becomes worse as their complexity (i.e., *fp*) increases. On the other hand, as depicted in Figures 6.9 and 6.10, SWRL and Jena show the desired scaling behaviour, as their subscription performance does not degenerate as more filter rules are added to the ontology. In terms of filter rule complexity, SWRL shows a slight degra-

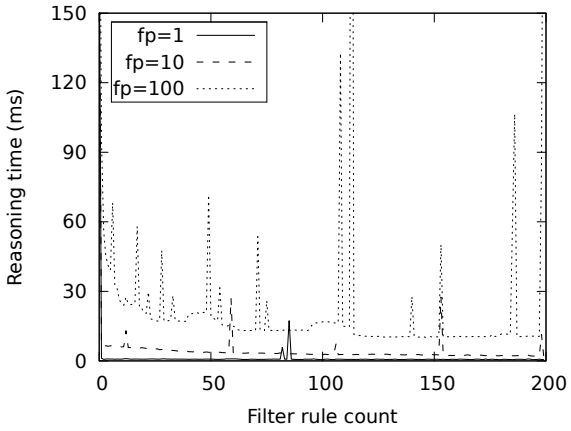


Figure 6.9: The evolution of total reasoning time as more SWRL filter rules are added to the SCB

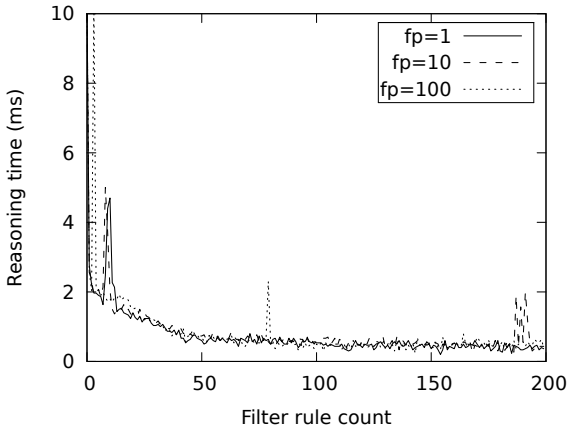


Figure 6.10: The evolution of total reasoning time as more Jena filter rules are added to the SCB

dation in performance, while Jena shows none. However, even for very complex rules ($fp = 100$), adding a subscription when the ontology already contains 200 filter rules takes, on average, only 10 and less than 1 ms for SWRL and Jena respectively. Note that all three approaches show a slight increase in reasoning time when adding the first few filters. This is caused by the dynamic class loading behaviour of Java and can thus be safely ignored. Additionally, the SWRL and Jena curves show random peaks, which are caused by measurement inaccuracies occurring due to measured times being in the range of only a few milliseconds.

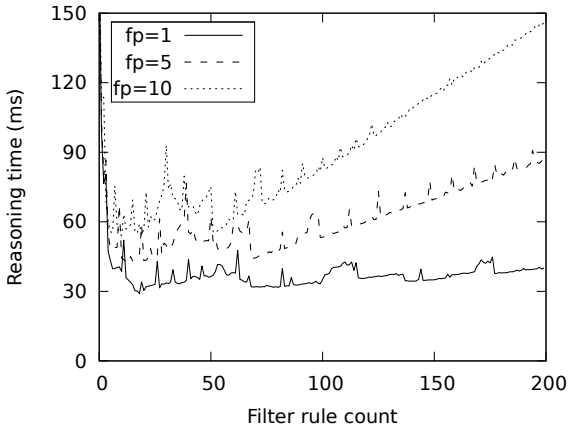


Figure 6.11: The evolution of total reasoning time as more OWL filter rules are added to the SCB with satisfiability and consistency checks turned off

For reference, Figure 6.11 shows the reasoning time for adding OWL filter rule subscriptions with satisfiability and consistency checks turned off. Surprisingly, the reasoning time still increases as more filter rules are added to the SCB. However, the reasoning time is reduced to over one thirtieth as compared to using OWL filter rules with satisfiability checks. Consequently, scalability is greatly increased.

In summary, SWRL and Jena filter rules have been shown to scale well, both in terms of subscription count and rule complexity. From the poor scaling behaviour of the OWL approach it can be concluded that checking the satisfiability and consistency of filter rules severely impacts overall performance, making it unsuitable for large-scale dynamic systems. We have shown that turning off satisfiability and consistency checks greatly increases OWL's scalability. However, even then its scalability remains worse than that of SWRL and Jena.

6.7.3 Context publication

An important aspect of the SCB is the publication of context messages. In contrast to filter rule subscriptions, which only change occasionally, the publication of context information happens frequently in highly dynamic environments, such as the management of future networks. As such, it is important that messages are matched with filter rules swiftly. This section further explores performance of the context publishing component, which performs the actual matching and dissemination of the messages. In the experiment, the effect on performance of three different parameters (i.e., subscription filter count, message complexity and filter rule complexity) was evaluated. For every evaluated combination of the three pa-

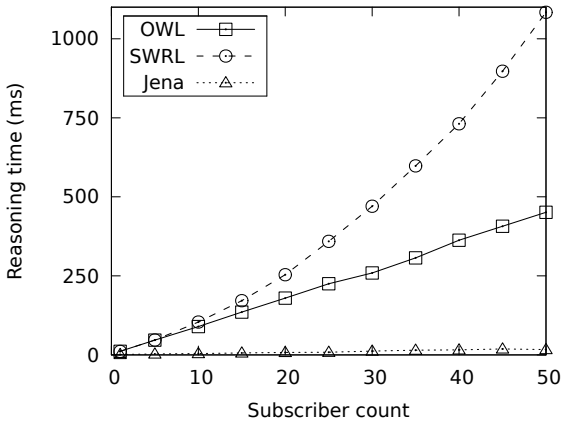


Figure 6.12: Average reasoning time for publishing a single message on the SCB, as a function of number of subscribers.

rameters, 500 messages were sequentially published unto the SCB. Filter rules of the same type as in the first experiment were used, but with a fixed load threshold of 50%. Additionally, every message payload contains information about a single *QuantifiableResource* with a randomly generated current load between 0 and 100%. The depicted reasoning time equals the time it takes to match the message with all subscription filters, averaged over the last 450 published messages. The first 50 messages are ignored, as the reasoning time for sending the first few messages is adversely influenced by Java's dynamic class loading behaviour. All experiments were repeated for OWL, SWRL and Jena filter rules. Figures 6.12, 6.13 and 6.14 depict the results.

The results shown in Figure 6.12 depict the reasoning time as a function of the number of subscription filters, for messages and filters with 1 payload each. The graph shows that SWRL and OWL filter rules scale poorly in terms of the number of subscription filters. In a scenario with 50 subscription filters, it takes over 1 second to publish a single message using SWRL filters and almost 500 milliseconds when using OWL. Such large delays are obviously unacceptable in a large scale dynamic network management scenario where context is constantly being exchanged between components. The fast degradation of these approaches is a consequence of the fact that the reasoner performs OWL inferencing when matching messages to filter rules, which is known to scale poorly. As such, a third approach, based on Jena rules was proposed. In contrast, it does not perform OWL inferencing, but merely applies the rules to the asserted RDF graph. This is clearly reflected in the results, as the Jena approach scales much better. Even for 50 subscription filters, publishing a messages takes only 16 ms on average, which

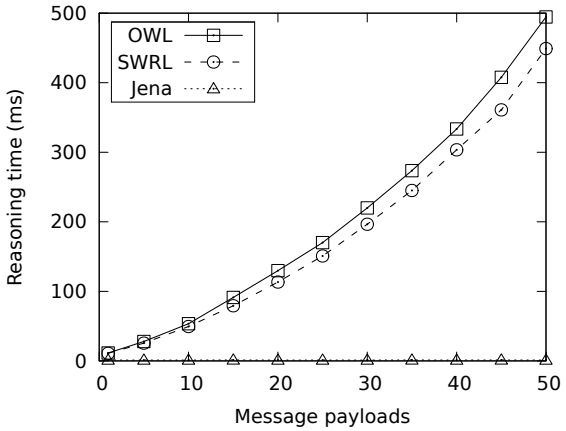


Figure 6.13: Average reasoning time for publishing a single message on the SCB, as a function of number of message complexity.

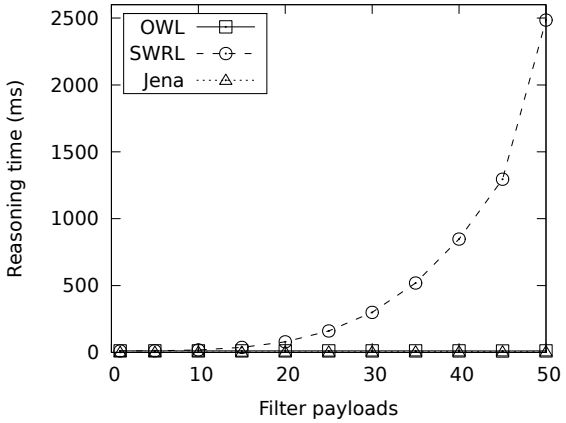


Figure 6.14: Average reasoning time for publishing a single message on the SCB, as a function of number of filter rule complexity.

means that over 60 semantic messages can be published per second.

Figure 6.13 shows the reasoning times as a function of the message complexity, for 1 subscription filter with 1 filter payload. The results are in accordance with previous observations and once again show that the message publishing process scales much better when using Jena rules. There is actually no noticeable degradation as message complexity increases. Specifically, for a message with 1 payload, publishing takes, on average, 1.15 ms, while for a message with 50 it takes only 1.25 ms.

Finally, the graphs in Figure 6.14 show scalability in terms of filter rule complexity. SWRL once again shows poor scalability. However, OWL and Jena show no performance degradation. This shows that determining if a message instance belongs to a certain OWL filter rule is independent of the filter rule's complexity. On the other hand, SWRL rule complexity does greatly influence performance.

Several conclusions follow from these observations. First, SWRL performs worst in terms of subscription filter count and filter complexity, while OWL performs worse in terms of message complexity. Pure OWL reasoning thus scales worse as a function of increasing ABox size (i.e., number of individuals), while SWRL reasoning scales worse in terms of increasing TBox size (i.e., number of classes and SWRL rules). Second, it was shown that only Jena rules are currently suitable for usage in a large-scale scenario containing many publishers and subscribers. It has been shown to scale very well in terms of subscription filter count, message complexity and filter rule complexity.

6.7.4 Service matchmaking

In contrast to the context dissemination component, the service matchmaker has less stringent timing constraints. Although it is still expected to react in a timely fashion, its delay can be in the order of seconds, rather than milliseconds. This section explores the effect of several parameters on the service matchmaker's performance in terms of execution time. The matchmaker's performance is influenced by two parameters, the number of service descriptions it offers and the percentage of these descriptions that actually match (i.e., are compatible with) the requested service. Obviously, the service matchmaker has to iterate over all offered service descriptions in order to find the matches. The number of offered service descriptions is thus expected to be directly proportional to the execution time of the matchmaker. However, as was explained in Section 6.6, the service matchmaker is often capable to detecting incompatibilities between service descriptions early on during the comparison. For example, if the offered service's input parameters do not match the requested ones, the matchmaker no longer needs to check outputs, preconditions and effects. As such, determining matchings takes longer if the services are actually compatible. Consequently, execution time depends on the percentage of offered service descriptions that match the requested service.

During each experiment run, the matching process was repeated 100 times for the same requested service description. The first 50 iterations are ignored, once again to negate the effects of Java's dynamic class loading. The depicted results are averaged over the last 50 iterations. The used service descriptions are based on the complex example given in Table 6.2, which allows server resources to be reserved for virtual machines. Of the offered service descriptions that do not match the requested, 2 out 3 have an incompatibility in the inputs, while the other third

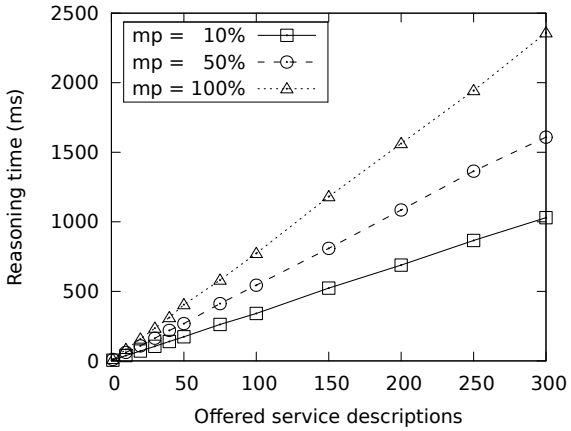


*Figure 6.15: The evolution of total reasoning time as a function of the number of offered service descriptions, for different percentages of matching descriptions (*mp*)*

has an incompatibility in the preconditions.

The experimental results are depicted in Figure 6.15. The graph shows the execution time of the matchmaking algorithm as a function of the total number of offered service descriptions, for different percentages of matching descriptions (*mp*). The graph clearly shows that there is a direct linear relation between reasoning time and both the number of offered service descriptions and the percentage of matches. In reality, the amount of service descriptions that actually match with the requested service is expected to be very low. As a wide range of differing services will be offered and requested. Consequently, the reasoning time will be significantly reduced. Even when 10% of the offered descriptions match with the requested service, the reasoning time is reduced by over half compared to when all service descriptions match. Additionally, these results show that on the test server, the matchmaker is capable of evaluating 300 possible matches, with a match rate of 10%, in under 1 second.

6.8 Conclusion

This chapter presents the Semantic Communications Bus (SCB), which facilitates the communication and interaction between AEs and network resources. It supports the semantic dissemination of context and matchmaking of service descriptions. This chapter presented several novel contributions. First, we proposed three alternative methods for representing filter rules, with differing inferencing capabilities, expressiveness and performance. Additionally, we have shown how existing semantic reasoners can be employed to match these rules with context. Second,

we proposed a method for modelling service inputs, outputs, preconditions and effects (IOPEs) by means of SWRL atoms. In contrast to existing work, our proposed matchmaking algorithm takes into account semantic links between different IOPE atoms.

In a federated network management scenario, context information, representing the state of the network and its resources, needs to be efficiently disseminated between AEs in order to detect and solve problems in a timely fashion. The proposed context dissemination approach uses ontologies and semantic reasoning, which support context filtering based on the actual meaning of information instead of static string patterns or predefined topics. Three different reasoning approaches were introduced, respectively based on OWL, SWRL and Jena. As these approaches have different inferencing capabilities, expressiveness and performance, we believe they all have their merits. The evaluation of our implemented prototype shows that Jena-based filter rules exhibit the best scaling behaviour in terms of number of filters and message complexity. Jena allows context to be matched to filter rules in a matter of milliseconds, while for OWL and SWRL rules it takes several hundreds of milliseconds in larger scenarios. On the other hand, OWL and SWRL filter rules support advanced inferencing, which is not supported when using Jena rules.

When managing large-scale networks, detected problems can often not be solved locally. Consequently, AEs need to cooperate in order to solve the network's management issues. The semantic matchmaking algorithm proposed in this chapter allows AEs to discover the management services, offered by other AEs, they require in order to complete their management tasks. Additionally, by taking into account the preconditions and effects of the management services, AEs can determine their consequences on the state of the managed environment. As the IOPEs of these services are semantically defined using SWRL atoms, the matchmaking algorithm can determine compatibility between offered and requested functionality based on the meaning and inferred semantic relatedness of ontological concepts. This greatly augments its accuracy compared to traditional keyword-based matchmaking approaches. The evaluated prototype implementation shows that the algorithm can determine semantic and functional compatibility between two service descriptions in a few milliseconds.

Acknowledgement

Jeroen Famaey is funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT-Vlaanderen) under grant no. 73185; Steven Latré is funded by the Fund for Scientific Research Flanders (FWO-Vlaanderen).

References

[1] B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M. Ó. Foghlú, W. Donnelly, and J. Strassner. *Towards autonomic management of communications networks*. IEEE Communications Magazine, 45(10):112–121, 2007. doi:10.1109/MCOM.2007.4342833.

[2] J. Strassner, S.-S. Kim, and J. Won-Ki Hong. *The design of an autonomic communication element to manage future internet services*. In 12th Asia-Pacific Network Operations and Management Symposium, pages 122–132, 2009. doi:10.1007/978-3-642-04492-2_13.

[3] B. Christudas. *Service-oriented Java business integration: Enterprise service bus integration solutions for Java developers*. Packt Publishing, 2008.

[4] M. Serrano, S. van der Meer, V. Holum, J. Murphy, and J. Strassner. *Federation, a matter of autonomic management in the Future Internet*. In 12th IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 845–849, 2010. doi:10.1109/NOMS.2010.5488357.

[5] B. Jennings, R. Brennan, W. Donnelly, S. Foley, D. Lewis, D. O'Sullivan, J. Strassner, and S. van der Meer. *Challenges for federated, autonomic network management in the Future Internet*. In 1st IFIP/IEEE International Workshop on Management of the Future Internet (ManFI), pages 87–92, 2009. doi:10.1109/INMW.2009.5195942.

[6] J. Famaey, S. Latré, J. Strassner, and F. De Turck. *A hierarchical approach to autonomic network management*. In 2nd IFIP/IEEE International Workshop on Management of the Future Internet (ManFI), pages 225–232, 2010. doi:10.1109/NOMSW.2010.5486571.

[7] J. Strassner, J. de Souza, D. Raymer, S. Samudrala, S. Davy, and K. Barrett. *The design of a novel context-aware policy model to support machine-based learning and reasoning*. Cluster Computing, 12(1):17–43, 2009. doi:10.1007/s10586-008-0069-4.

[8] S. Latré, S. van der Meer, F. De Turck, J. Strassner, and J. Won-Ki Hong. *Ontological generation of filter rules for context exchange in autonomic multimedia networks*. In 12th IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 575–582, 2010. doi:10.1109/NOMS.2010.5488448.

[9] J. Famaey, S. Latré, J. Strassner, and F. De Turck. *An ontology-driven semantic bus for autonomic communication elements*. In 5th IEEE international conference on Modelling Autonomic Communication Environments (MACE), pages 37–50, 2010. doi:10.1007/978-3-642-16836-9_4.

[10] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. *Jena: Implementing the semantic web recommendations*. In 13th International World Wide Web Conference, 2004. doi:10.1145/1013367.1013381.

[11] M. Petrovic, H. Liu, and H.-A. Jacobsen. *G-ToPSS: Fast filtering of graph-based metadata*. In 14th international conference on World Wide Web (WWW), pages 539–547, 2005. doi:10.1145/1060745.1060824.

[12] J. Wang, B. Jin, and J. Li. *An ontology-based publish/subscribe system*. In 5th ACM/IFIP/USENIX international conference on Middleware (Middleware), pages 232–253, 2004. doi:0.1007/978-3-540-30229-2_13.

[13] J. Ma, G. Xu, J. Wang, and T. Huang. *A semantic publish/subscribe system for selective dissemination of the RSS documents*. In Fifth International Conference Grid and Cooperative Computing (GCC), pages 432–439, 2006. doi:10.1109/GCC.2006.19.

[14] H. Li and G. Jiang. *Semantic message oriented middleware for publish/ subscribe networks*. In Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense III, volume 5403, pages 124–133, 2004. doi:10.1117/12.548172.

[15] J. Skovronski and K. Chiu. *An ontology-based publish-subscribe framework*. In International Conference on Information Integration and Web-based Applications Services (iiWAS), 2006.

[16] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. *Design and evaluation of a wide-area event notification service*. In Foundations of Intrusion Tolerant Systems, pages 283–334, 2003. doi:10.1109/FITS.2003.1264940.

[17] J. Keeney, D. Roblek, D. Jones, D. Lewis, and D. O'Sullivan. *Extending siena to support more expressive and flexible subscriptions*. In Second International Conference on Distributed Event-Based Systems (DEBS), pages 35–46, 2008. doi:10.1145/1385989.1385995.

[18] M. Petrovic, I. Burcea, and H.-A. Jacobsen. *S-ToPSS: Semantic toronto publish/subscribe system*. In 29th International Conference On Very large Data Bases (VLDB), pages 1101–1104, 2003.

[19] J. Wang, B. Jin, J. Li, and D. Shao. *A semantic-aware publish/subscribe system with RDF patterns*. In 28th Annual International Computer Software and Applications Conference (COMPSAC), pages 141–146, 2004. doi:10.1109/CMPSAC.2004.1342818.

[20] G. Shen, Z. Huang, Y. Zhang, X. Zhu, and J. Yang. *A semantic model for matchmaking of web services based on description logics*. Fundamenta Informaticae, 96(1):211–226, 2009. doi:10.3233/FI-2009-175.

[21] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. *Semantic matching of web services capabilities*. In First International Semantic Web Conference (ISWC), pages 333–347, 2002. doi:10.1007/3-540-48005-6_26.

[22] A. B. Bener, V. Ozadali, and E. S. Ilhan. *Semantic matchmaker with precondition and effect matching using SWRL*. Expert Systems and Applications, 36(5):9371–9377, 2009. doi:10.1016/j.eswa.2009.01.010.

[23] M. Klusch, B. Fries, and K. Sycara. *Automated semantic web service discovery with OWLS-MX*. In Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, pages 915–922, 2006. doi:10.1145/1160633.1160796.

[24] M. Klusch, B. Fries, and K. Sycara. *OWLS-MX: A hybrid semantic web service matchmaker*. Web Semantics: Science, Services and Agents on the World Wide Web, 7(2):121–133, 2009. doi:10.1016/j.websem.2008.10.001.

[25] M. L. Sbodio, D. Martin, and C. Moulin. *Discovering semantic web services using SPARQL and intelligent agents*. Web Semantics: Science, Services and Agents on the World Wide Web, 8(4):310–328, 2010. doi:10.1016/j.websem.2010.05.002.

[26] M. J. Serrano, J. Serrat, J. Strassner, and M. Ó Foghlú. *Management and context integration based on ontologies, behind the interoperability in autonomic communications*. In SIWN International Conference on Complex Open Distributed Systems (CODS), 2007.

[27] J. Strassner, J. N. Souza, S. van der Meer, S. Davy, K. Barrett, D. Raymer, and S. Samudrala. *The design of a new policy model to support ontology-driven reasoning for autonomic networking*. Journal of Network and Systems Management, 17(1):5–32, 2009. doi:10.1007/s10922-009-9119-3.

[28] A. Wong, P. Ray, N. Parameswaran, and J. Strassner. *Ontology mapping for the interoperability problem in network management*. IEEE Journal on Selected Areas in Communications, 23(10):2058–2068, 2005. doi:10.1109/JSAC.2005.854130.

[29] M. J. O'Conner and A. K. Das. *A lightweight model for representing and reasoning with temporal information in biomedical ontologies*. In International Conference on Health Informatics (HEALTHINF), 2010.

[30] M. Krötzsch, S. Rudolph, and P. Hitzler. *On the complexity of horn description logics*. In 2nd Workshop on OWL: Experiences and Directions (OWLED), 2006.

[31] H. Alt, N. Blum, K. Mehlhorn, and M. Paul. *Computing a maximum cardinality matching in a bipartite graph in time* $o(n^{1.5}\sqrt{m/\log n})$. Information Processing Letters, 37(4):237–240, 1991. doi:10.1016/0020-0190(91)90195-N.

[32] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. *Pellet: A practical OWL-DL reasoner*. Web Semantics: Science, Services and Agents on the World Wide Web, 5(2):51–53, 2007. doi:DOI: 10.1016/j.websem.2007.03.004.

7

A hierarchical context dissemination framework for managing federated clouds

J. Famaey, S. Latré, J. Strassner, and F. De Turck

Published in Journal of Communications and Networks

The recent rise to fame of the cloud computing paradigm has resulted in a large number of cloud computing data centers to pop up around the Internet. The growing popularity of utility computing has encouraged cloud providers to further expand their data centers. Modern cloud computing data centers thus consist of many servers, virtual machines and services, which generate huge amounts of monitoring information. Additionally, the paradigm is expected to converge towards global federations of cloud data centers. The hierarchical architecture proposed in Chapter 5 was designed to overcome the management issues of large-scale networked systems, such as clouds. This chapter presents an adaptation of the hierarchical management architecture, tailored to the specific needs of managing federated clouds. Additionally, the SCB (cf. Chapter 6) and the filter rule generation algorithm (cf. Appendix A) are incorporated in the framework, by adapting them to operate in a distributed and hierarchical manner. Appendix B proposes a methodology to transform resource reservation algorithms for clouds into hierarchical versions, compatible with the framework presented in this chapter.

7.1 Introduction

The proliferation of value-added services offered across the Internet has resulted in a significant growth in size and complexity of modern computing and communications infrastructures. It has been argued that current, monolithic management systems will not be able to keep up with this ever-increasing complexity [1]. To alleviate these concerns, researchers have proposed novel network management architectures based on the cooperation between many automated management components, often called Autonomic Elements (AEs) [2, 3]. Every AE autonomously manages a small subset of the network's resources, while being guided by high-level, human-specified business goals. Obviously, AEs will need to communicate and collaborate in order to achieve their goals. They need to exchange context, which is used to model the current state of their environment, in order to detect sub-optimal behaviour and faults in the underlying network and computing resources. Additionally, as not all problems can be solved locally, it might be necessary for them to initiate management functionality offered by other AEs or negotiate contracts to set up federations across management domains.

Although the increasing management complexity is an issue in many different computing and communication areas, it is especially prevalent in the management of cloud computing data centers. To keep up with the growing demand for cloud computing resources, cloud providers have greatly increased the capacity of their networks and data centers. Additionally, there has been a shift from monolithic single-domain clouds towards large-scale federated cloud environments [4]. This has led to the hypothesis of an Internet-wide inter-cloud, a global cloud of clouds [5]. It is expected these trends will persist in the future, leading to an ever-increasing size, heterogeneity and complexity of cloud computing infrastructures. Consequently, they have stringent requirements concerning management scalability and efficiency.

This chapter proposes a framework that facilitates the collaboration and communication between AEs in a large-scale distributed management architecture. More specifically, the presented framework is responsible for the dissemination of huge amounts of context. Context is defined as "the collection of measured, inferred and exchanged knowledge that describes the state and environment in which an Entity exists or has existed" [6]. Additionally, it is detailed how this context dissemination framework fits within a typical autonomic management architecture. The interactions with other important components, such as policies, the knowledge base, federation contracts and management services are thoroughly described.

Within a management domain, AEs are structured in a hierarchy, which we have previously shown to scale better than flat distributed or centralized approaches [7]. The location of AEs within the hierarchy reflects their management responsi-

bilities. High-level AEs have a wide, yet aggregated, view on the managed environment, and similarly perform high-level management functions. Low-level configuration tasks are delegated to the underlying AEs, which have a more detailed view on a subset of the environment. By restricting the view and/or granularity of the context of AEs depending on the location within the hierarchy, scalability is ensured throughout. The context requirements of an AE obviously depend on its location within the hierarchy. Additionally, these requirements may change dynamically, as the state of the managed environment changes. Existing management approaches often disseminate all context that might be needed by the management algorithms at some point in time continuously, which obviously leads to huge amounts of overhead and reduced scalability. To solve these issues, we propose a dynamic context dissemination algorithm that delivers the right context, to the right place, at the right time. The algorithm automatically generates filter rules, which describe what context an AE is interested in. Both these filter rules and the actual context are augmented with semantics. This, in turn, allows AEs to define what context they are interested in, based on the underlying meaning of the information rather than static string patterns. Additionally, it supports the interpretation of received context and thus allows the AE to more accurately perform its automated filter rule generation process.

The management of modern cloud computing infrastructures requires huge amounts of context to be disseminated and processed. Therefore, this management area would greatly benefit from scalable and efficient context dissemination and filtering. Additionally, clouds have specific requirements concerning the filtering and aggregation of context throughout the management hierarchy. More specifically, cloud resource allocation algorithms require specific types of context based on the state of the underlying server resources. To encompass these cloud-specific aspects, and validate our framework for a large-scale management scenario, the presented framework is applied to a cloud management scenario. A prototype of the presented framework was evaluated in detail using a context dissemination scenario in a cloud data center of up to $100,000$ servers.

In summary, this chapter offers the following novel contributions, which, to our knowledge, are not present in existing context dissemination frameworks. First, a hierarchical context dissemination framework is presented. Context is aggregated and transformed as it is propagated through the hierarchy, offering different views on the managed resources. Second, it not only filters context semantically, based on meaning, but is also capable of dynamically adapting these filter rules to satisfy changing context requirements. Third, the framework is applied to the management of cloud computing data centers. The specific context requirements are identified and it is shown how the context dissemination framework is able to meet them. Finally, by way of an analytical model and an implemented prototype, we evaluate the scalability and applicability of our framework in a large-scale cloud

management scenario.

The remainder of this chapter is structured as follows. Section 7.2 lists related work on federated management and the use of semantics in network management. The hierarchical context dissemination between management components in clarified in Section 7.3. Subsequently, Section 7.4 gives a detailed overview of the architectural building blocks that make up the AE and explains how context dissemination fits into it. The introduced architectural concepts are applied to management of cloud environments in Section 7.5. Section 7.6 provides a thorough evaluation and validation of the presented framework, based on an analytical formulation and an implemented prototype. Finally, Section 7.7 concludes the chapter.

7.2 Related work

This section elaborates on state of the art research related to the work presented in this chapter. First, recent research on federated network and cloud management is discussed in more detail. Second, the semantic components presented in this chapter are compared to existing semantic reasoning efforts in network management.

7.2.1 Federated network management

In recent years, it has been argued that future networks should support loosely coupled collaborations across management domains, or federations. As such, there has been an increasing interest by the research community in federated network management in general [1, 8–10] and federated cloud computing in particular [4, 5, 11, 12].

Jennings *et al.* [1] identified several important challenges that need to be tackled in order to achieve a federated Future Internet, capable of supporting loosely coupled end-to-end value networks. Their work addresses six challenges in total, across three levels of abstraction: federated service management, service monitoring and configuration and network infrastructure. An important identified requirement is the facilitation of understanding the meaning of measured and inferred data, and supplying the results of these data and their associated conclusions to interested collaborating management entities. Our work builds on that requirement, and provides additional and more concrete semantic reasoning components to accomplish this. Chai *et al.* [8] proposed an orchestration plane that coordinates the interactions within network federations. Their work focusses on a negotiation framework for setting up collaborations between management domains. The proposed work, for now, only supports resource reservations, and is not concerned with the semantics that are needed to achieve understanding between the negotiating parties. Nevertheless, their work can be considered an interesting first step towards a generic contract negotiation framework for setting up dynamic network

federations. Feeney *et al.* [10] further study the sharing of capabilities across network domains. In contrast, they do propose a semantic approach, which allows capabilities to be represented in Resource Description Framework (RDF) syntax. They present the Layered Federation Model (LFM), which, among other things, provides a semantic mapping framework. This framework maps diverging semantic models, defined in different management domains, onto one another, which eases the understanding of capabilities across domains.

Recently, the idea of the intercloud, a global federation of clouds, was launched [5]. Its ultimate goal is to allow scaling of applications across multiple cloud providers. This is necessary to achieve the (seemingly) infinite scaling of resource provisioning, as promised by the cloud computing paradigm [13]. Rochwerger *et al.* [11] proposed a modular and extensible cloud architecture for the federation of clouds, called RESERVOIR. It allows providers of cloud infrastructure to dynamically partner with each other to create a seemingly infinite pool of IT resources, while fully preserving the autonomy of technological and business management decisions. More recently, Buyya *et al.* [12] furthered this research towards a federated cloud computing architecture that tackles several pertinent challenges. It is capable of predicting demands and behaviour of the hosted services, and has an economic-model-driven optimization process. Finally, Celesti *et al.* [4] proposed a three stage model for the evolution of clouds, from the current monolithic cloud providers to horizontal federations, where cloud providers will federate themselves to gain economics of scale and an enlargement of their capabilities. They argue that currently clouds are moving towards the second stage, or the vertical supply chain, where cloud providers leverage cloud services from other providers. We feel that, in order to realize the vision of the intercloud, more semantically rich notions of context, capabilities, constraints and requirements are necessary. Otherwise, providers cannot efficiently work together, because they do not have the ability to effectively and correctly interpret information and collaborate. Hence, this chapter proposes semantic reasoning components that enrich the autonomic control loops that govern the providers participating in the intercloud. This provides the appropriate context to enable each provider to make informed decisions when resource and/or service requests are received.

In this chapter, we present approaches that facilitate both the design and maintenance of a management framework for federated clouds. The design principles used in this framework are based on best practice libraries such as the Information Technology Infrastructure Library[1] (ITIL) and Enhanced Telecom Operations Map[2] (eTOM). These libraries define high-level best practices that describe processes and process workflows. Specifically for this chapter, ITIL aspects such as service operation were taken into account. Similarly, the proposed framework is

[1] http://www.itil-officialsite.com

[2] http://tmforum.org/BusinessProcessFramework/1647/home.html

also based on several eTOM aspects such as Resource Management and Operations and Partner Relationship Management.

7.2.2 Semantics in network management

Semantics have been widely employed to solve several network management issues, including context dissemination [14] and service matchmaking [15, 16]. The context dissemination process pushes messages, containing context data and associated inferences, towards a set of interested subscribers. It can thus be modelled as a publish/subscribe system or enterprise service bus. Several semantic publish/-subscribe mechanisms have been proposed throughout the years. Early work was often based on RDF graph matching [17, 18]. Messages are represented using RDF graphs, while subscriptions take the form of graph patterns. More recently, Skovronski and Chio [19] presented an approach based on SPARQL Protocol and RDF Query Language[3] (SPARQL) queries as subscriptions, which similarly perform matching based on RDF triples. The semantics that can be captured with graph matching algorithms or SPARQL queries are limited to property and hierarchical relationships. Li *et al.* [20] proposed a more expressive approach, using standard semantic reasoners to determine a match between messages and subscriptions. Subscriptions take the form of DARPA Agent Markup Language and Ontology Inference Layer (DAML+OIL)[4] classes, while messages are represented by instances. If the reasoner infers that a message instance belongs to a subscription class, then the message satisfies the subscription. The filtering approach presented in our work is similar to that proposed by Li *et al.* However, we use the more modern OWL, instead of DAML+OIL. Our proposed context disseminator additionally supports subscriptions to be defined in the form of Semantic Web Rule Language (SWRL)[5] and Jena[6] rules, which further increases expressiveness through a wide range of built-ins, including comparison, string and mathematical operators.

In the context of web services, several semantic service matchmaking methods have been presented. OWLS-MX is a hybrid semantic web service matchmaker for OWL-S services [21]. OWL-S[7] is an ontology built on top of OWL for describing semantic web services. However, OWLS-MX only uses information about inputs and outputs, not preconditions and effects. For AEs it is necessary to estimate the influence of services on their managed environment and as such an approach that incorporates preconditions and effects is necessary. More recently, several semantic matchmaking algorithms that do take into account preconditions and effects have been proposed, based on description logics [22], SPARQL [23]

[3] http://www.w3.org/TR/rdf-sparql-query/
[4] http://www.daml.org
[5] http://www.w3.org/Submission/SWRL/
[6] http://jena.sourceforge.net/
[7] http://www.w3.org/Submission/OWL-S/

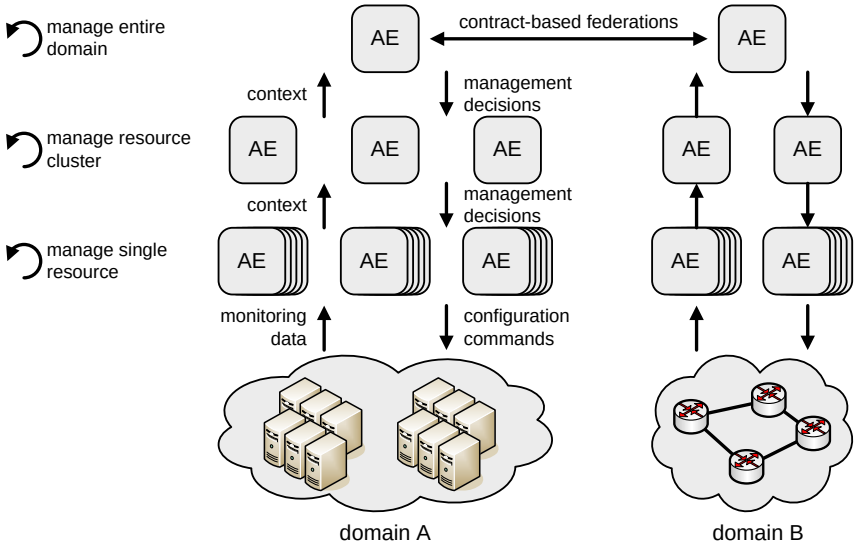


Figure 7.1: An overview of the collaborative, AE-driven, hierarchical management architecture

and SWRL [24]. In line with Bener *et al.* [24], we propose a semantic matchmaking algorithm based on SWRL. However, our algorithm takes into account a more complete definition of SWRL-atom subsumption relationships, thus resulting in more accurate matchings. Additionally, our algorithm, in contrast to the state of the art, supports the use of SWRL variables, which provide a means to semantically link inputs and outputs to preconditions and effects.

7.3 Hierarchical context dissemination

In this section, we identify and describe the interactions that take place between AEs in a highly dynamic, distributed, management environment. Specifically, the section focusses on context dissemination between management components throughout a management hierarchy.

7.3.1 Architectural overview

A modern network domain consists of a large number of physical (e.g. servers, switches, and network links) and logical (e.g. virtual machines, operating systems, software libraries, and services) resources. Figure 7.1 depicts how a set of AEs interacts with such an infrastructure. The AEs within a management domain, are structured in a hierarchical fashion. In previous work, we have high-

lighted the qualitative and quantitative advantages of AE hierarchies, both in a generic network management context [7] and specifically for the management of clouds [25]. Throughout this chapter, a hierarchy with three main levels is used. The bottom level directly manages the physical resources, the middle level a cluster of resources, and the top level the entire domain. No assumptions are made concerning the number of AEs within a level. For example, a cluster of resources could be managed by a single AE, or by a hierarchy of collaborating AEs. The overall structure of the AE hierarchy, and more specifically the total number of sub-levels within each of the three main levels, obviously has its implications on scalability. Section 7.6 studies this in more detail. Additionally, no assumptions are made concerning the physical location of the AEs. They can, for example, be co-located with the managed infrastructure, executing them on the computing resources within the domain. Alternatively, to increase security and availability, part of the physical infrastructure can be reserved specifically for the AEs, effectively decoupling the managed environment from the management framework.

7.3.2 Resource management

At the bottom level, AEs are directly responsible for the management and configuration of a single, or small group of, resources. For example, an AE managing a router could be, among others, responsible for configuring and managing a context-aware admission control function. They perform low-level maintenance tasks, such as monitoring, basic problem detection and configuration. The AEs have three distinct responsibilities. Their main responsibility is to convert the continuous stream of monitoring data into semantically annotated context and forward an aggregated and summarized version to their parent AE. Second, the AE can be fitted with some basic reasoning functionality, allowing it to solve low-level problems, for which no global knowledge is necessary. For example, the AE could be responsible for managing and configuring an admission control component that rejects service requests if a managed server's load becomes too high. The third and final responsibility of the bottom level AEs, is the configuration of the network's resources. They offer a set of management functions for configuring the underlying resources. The implementation of these functions is vendor specific. The AEs should thus be aware of the type of resources they manage and should be able to translate generic commands into vendor specific ones. AEs at higher levels in the hierarchy can influence the managed environment by dynamically discovering and initiating the management functions of bottom level AEs.

7.3.3 Cluster management

As we move up through the hierarchy, AEs take on more responsibilities and become capable of performing higher level and more widespread management tasks.

For example, an AE managing a group of interconnected switches and routers is capable of changing domain-wide routing policies, such as configuring consistent QoS parameters across all devices. AEs at the second level are responsible for managing a cluster of resources. They model the state of their environment using context gathered through their child AEs. Depending on the current state of the environment, the AE can adapt the granularity and dissemination interval of the context it requests from its children. For example, when the underlying environment is in a stable state, only aggregated statistics are necessary in order to detect possible future problems. However, once a potential problem has been detected, more detailed and frequent context is needed in order to react in a timely fashion. This dynamic context dissemination process is further described in Section 7.4. At the middle level, AEs have a view over potentially many resources. Therefore, they are capable of detecting more intricate problems and deploying more widespread solutions.

7.3.4 Domain management

At the top of the AE hierarchy is one, or a set of, root AEs. They have a high-level, highly aggregated view of the network. Their responsibilities include high-level management tasks that impact the entire domain. For example, in a cloud scenario this entails resource provisioning and service migrations across data centers. Therefore, they are capable of taking into account network related parameters and metrics in their decision making process. For example, services could be migrated closer to the customers that consume them, in order to optimize network-related parameters.

7.3.5 Federated management

In addition to managing the high-level aspects of the domain, root level AEs are responsible for setting up loosely-coupled federations. They communicate with root AEs of other domains in order to negotiate contracts and set up federations. Such federations support novel types of collaboration, for example allowing resource and revenue sharing. For example, in a cloud scenario, AEs could decide to migrate services from one cloud provider to another. This allows customer demands to remain satisfied even when an entire cloud provider's infrastructure becomes overloaded. To achieve this level of collaboration, providers need to exchange context information about availability, infrastructure and resources. However, making detailed information available may not be possible due to scalability reasons, or wanted due to business reasons. The context filtering and aggregation techniques applied within a management domain, can therefore also be used in the context dissemination process between domains.

7.3.6 Context continuum

The management responsibilities change throughout the AE hierarchy. This is reflected in differing capabilities and context requirements. Additionally, these responsibilities map to the Policy Continuum [26]. At the top, policies represent high-level business objectives and context is highly aggregated and summarized. As we move down the hierarchy, policies are gradually translated into low-level device configurations. A similar translation can be seen on the context level: context is propagated upwards and is dynamically transformed, aggregated and summarized. This allows us to maintain scalability throughout the hierarchy. At the bottom, AEs have a very detailed, yet narrow view of the domain, while at the top the view is domain-wide, but much less detailed. A Context Continuum can thus be defined in line with the Policy Continuum. Furthermore, policies at the highest level of the continuum will typically relate to context at the highest level and vice versa.

7.4 Autonomic element architecture

The previous section focussed on the global system architecture and high-level interactions between AEs. In contrast, this section zooms in on the internal workings of the AE architecture and addresses the functional details of AE interactions. Section 7.5 further applies the algorithms and concepts introduced in this and the previous section to a practical resource allocation use-case in a federated cloud computing environment.

An overview of the internal AE architecture is depicted in Figure 7.2. As illustrated, an AE consists of loosely coupled components that asynchronously interact with each other. The AE is made up of six main components: knowledge base, context disseminator, autonomic manager, service repository, contract repository and policy framework. Together, they form dynamic control loops, which adapt their behaviour based on the AEs location within the hierarchy and the current state of the environment. The remainder of this section discusses these components in more detail.

7.4.1 Knowledge base

To perform their management tasks, AEs require information about their managed environment to reason upon and deduce new configurations. This information takes the form of monitoring data (received by the managed resources), remote context (received by the neighbouring AEs) and knowledge (locally deduced by the AE itself). New knowledge is typically deduced by the Autonomic Manager, through ontological reasoning or machine learning algorithms. For example, based on measured monitoring data, the AE could infer that a specific server hardware

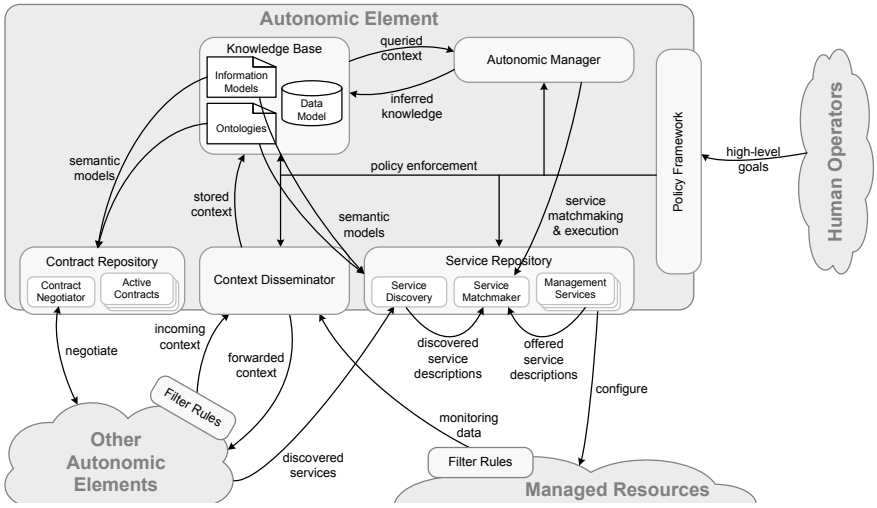


Figure 7.2: Overview of the internal AE architecture and its interactions with other AEs and managed resources

component has failed. The proposed AE architecture takes a dual approach in representing this knowledge. The bulk of the knowledge is stored in a data model. The AE is capable of interpreting, and reasoning upon, this stored knowledge using a set of embedded information models and ontologies.

7.4.1.1 Information models & ontologies

An information model represents the concepts, relationships, constraints, rules and operations that define a specific domain. For example, in a cloud environment, an information model typically defines concepts that are of interest to the overall managed environment. Such concepts range from abstract and high-level (e.g., a server) to low-level (e.g., the load of a CPU core on a server). This enables different entities to be related to each other. The semantic capabilities of AEs stem from the information models and ontologies embedded within. They are used by other components inside the AE for reasoning and deducing new knowledge, which can in turn be added to the ontologies or stored in the data model. Although the information models and ontologies provide a consistent view of the management domain, their size can quickly expand up to hundreds of concepts and thousands of relationships. The performance of ontological reasoning is known to scale exponentially [27]. As such, every component only uses the subset of the complete information models and ontologies that are relevant to its own context and environment, which significantly reduces overhead. The components must define the parts they are interested in.

7.4.1.2 Data model

All data relevant to the AE is stored in the data model. The data model is an instantiation of the information models. It allows storing values and instances for the defined concepts, and thus acts as a database that characterizes the state of the managed resources as well as other business oriented aspects, such as high-level policies.

7.4.2 Context disseminator

The data being stored into the data model can originate locally (e.g., because new knowledge has been deduced) or remotely (i.e., from other AEs or managed resources). When context originates remotely, a decision needs to be made on which context needs to be transferred from other AEs and managed resources to the local AE. As discussed in [7], forwarding all context to all other AEs and managed resources is simply not possible for scalability reasons: the filtering of context can reduce the contextual overhead considerably. In the proposed architecture, this is the responsibility of the context dissemination component. For this, the context disseminator uses the publish/subscribe paradigm. In this paradigm, context consumers express their interest in data and/or information that is meaningful to them. In our proposed architecture, AEs specify interest in information by way of filter rules, which identify the type of information or context in which they are interested. Depending on the type of interaction, several types of filter rules are supported. For example, in a typical network management scenario, Simple Network Management Protocol (SNMP) traps [28] can be used to request monitoring information from a router. Additionally, semantic filter rules are supported for the exchange of context between AEs. This allows them to request context based on its actual meaning. In this approach, context messages take the form of ontological instances, while filter rules can be specified as ontological concepts or semantic rules (e.g., SWRL or Jena). Ontological reasoners are then used to match this semantically defined context with semantic filter rules. As this chapter focusses on the architectural components and the interactions between them, the algorithmic details of the filter rule matching process are omitted. The reader is referred to our previous work for more details [29]. Nevertheless, some examples, specific for the management of a large-scale cloud computing data center, are given in Section 7.5.

The contextual requirements of an AE change with the state of its managed environment. Existing approaches often disseminate all context that might be needed at some point at all times. In contrast, we propose a dynamic context dissemination process, where filter rules are automatically generated, based on the current state of the managed environment. For example, a reasoning component that migrates applications when a server becomes overloaded can choose to only monitor the average load of the server as a whole. When this average load indicates that the server

is overloaded, the reasoning component then requests additional information, such as the footprint of every application individually. The context dissemination component supports the context-sensitive generation of filter rules through ontological reasoning. The conditional contextual requirements of each component of an AE are defined by the context dissemination component. Among others, dependencies between contextual requirements can be introduced (e.g., stating that certain context may only be requested if other context has a particular value). For example, detailed statistics about the processes running on a server might not interest the AE managing this server, unless its aggregated resource load becomes too high. For more details on this generation algorithm, the reader is referred to [30].

7.4.3 Autonomic manager

The autonomic manager is responsible for detecting and solving problems and suboptimal performance. It performs this complex task through the use of specialized semantic reasoners, learning algorithms and management algorithms. Depending on the goals of the AE, different algorithms and reasoners, of differing complexity, may be active within the same autonomic manager. The autonomic manager is the premier consumer and producer of contextual data: it continuously extracts context from the knowledge base and stores newly inferred knowledge into it. To make effective decisions, it also requires contextual data that is only available remotely. As such, it depends heavily on the context dissemination process. This context is used to assess the current state of the environment and plan corrective actions. As such, this component forms the heart of the control loops of the AE. Depending on the scope and complexity of the management problem, as well as the location of the AE in the hierarchy, different management algorithms may need to be used within the control loops. This results in a wide variety of dynamic control loops, with varying responsibilities.

7.4.4 Service repository

When the state of the managed environment no longer matches the desired state, the AE's autonomic manager constructs a plan containing corrective actions that need to be performed. However, these actions need to be mapped to the available management services. Additionally, AEs offer specialized functionality based on their location within the hierarchy and environment. Therefore, they might not be able to perform all planned actions themselves. The service discovery component is responsible for both mapping actions to services (i.e., matchmaking), as well as discovering functionality offered by other AEs.

The management services come in two forms. The first type is capable of performing well-defined and straightforward tasks that alter or configure the managed resources. In a cloud environment, typical examples of such management services

are the migration of virtual machines, the allocation of resources and the hibernation of cloud servers. The second type of management service can be used to steer the autonomic manager. For example, in a cloud computing scenario, the top level AE might decide it is necessary to migrate a hosted cloud service from one data center to another. However, as it does not have a detailed view of the target data center, it could instruct the AE responsible for this data center to find a suitable server (or set of servers) for hosting the service. The AE responsible for this data center should then have a management service that instructs its autonomic manager to allocate resources for a specific hosted service.

Services are described through a set of inputs, outputs, preconditions and effects (IOPEs). The inputs represent the context that is required by the management service in order to perform its task. On the other hand, the outputs depict context that will be generated by the management service. The preconditions define the environmental conditions that must hold for the service to be usable, while the effects describe the conditions that will hold after the service's execution. Together, the preconditions and effects thus describe how the managed environment will change when the management service is executed. This allows AEs to better match these services with their planned actions. Additionally, variables may be attached to specific IOPEs, which allows preconditions and effects to be semantically linked to each other and inputs or outputs. The proposed matchmaking algorithm uses the subsumption relationships of IOPEs to determine compatibility between service descriptions and goals.

7.4.5 Contract repository

In a federated cloud computing scenario, the interactions between AEs within different management domains need to be governed by a contract negotiation process, as new issues, such as trust and conflicting management policies, arise that are not present within a single management domain. As previously stated, these contracts need to be augmented with semantics, based upon a shared information model, in order to support understanding and correct interpretation between the participants of the federation. Contracts are similar to service descriptions, as IOPE definitions can be used to semantically describe costs and benefits of the involved parties. While service descriptions only describe functional properties, contracts may also include business-related aspects of the interactions.

7.4.6 Policy framework

Autonomic environments are typically governed by policies. These policies represent the high and low-level goals human operators can introduce into the AE, used to steer the management process. The policy framework is the entity that collects these policies for further forwarding to the specific components inside the

AE. In the policy framework, policy related tasks such as the detection of conflicts between policies are maintained. Several policy conflict detection algorithms have been proposed in previous work [31, 32]. As policies form an inherent part of the inner workings of an AE, the policy framework interacts with all components inside the AE. For example, in the context dissemination, the filter rule generation process in the context dissemination can be governed by a policy that limits the amount of context that can be requested from a particular AE. The effect of such a policy on the filter rule generation process is that the generated filter rules will either request context at a lower frequency or increase the level of performed aggregation of lower level context.

As previously stated, policies take different forms throughout the AE hierarchy. This corresponding policy hierarchy is also referred to as the Policy Continuum. In order to achieve true autonomic behaviour, and thus further reduce management complexity for human operators, algorithms are needed that automatically perform the translation of policies throughout the hierarchy.

7.5 Managing the cloud

This section applies the generic concepts introduced in Sections 7.3 and 7.4 to the specific area of cloud computing. As previously stated, the size of cloud computing data centers is growing steadily. In turn, this is causing a significant and ever-increasing growth in context information that needs to be processed by management components. Consequently, our presented framework greatly increases efficiency and scalability of cloud management architectures by intelligently disseminating and dynamically filtering context based on the current state of the underlying resources. The contributions of this section are threefold. First, it is shown how the AE management hierarchy is mapped to cloud data centers. Second, the context requirements at all levels of the hierarchy are identified. Third, it is described how the presented framework is capable of meeting these dynamic context requirements in a scalable manner.

An important aspect of managing a cloud computing infrastructure is the allocation of physical server and network resources to virtual machines and hosted services. In previous work, we showed how existing resource allocation algorithms can be adapted for use in an AE hierarchy [25]. In contrast, this chapter focusses more on the interactions between AEs in order to facilitate such hierarchical collaborations. As such, specific algorithmic details are omitted.

When performing resource allocation in a cloud computing environment, several different types of context are needed at different levels of the AE hierarchy depending on the current environment. AEs dynamically adapt their filter rules in order to receive the correct context based on the current situation. Additionally, context is aggregated in order to increase scalability. Figure 7.3 shows the

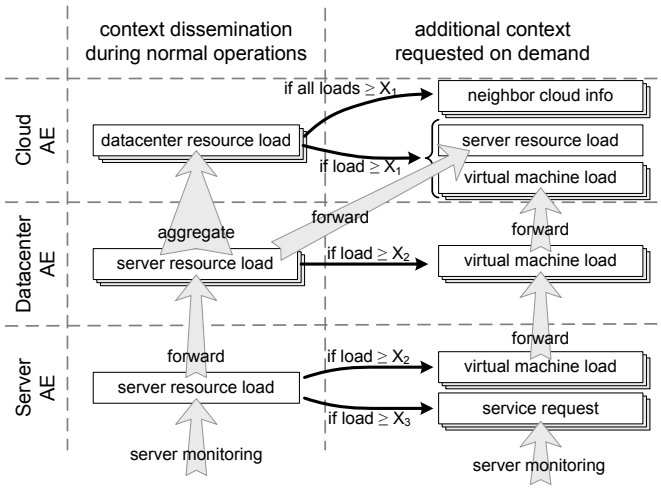


Figure 7.3: The context continuum for resource allocation in cloud computing; context is forwarded and aggregated through the AE hierarchy using dynamic context-aware filter rules

context requirements for the resource allocation case described in this section. As the performed management tasks significantly differ at the different levels in the hierarchy, this section is split into three subsections, which present AE operations at different levels in the hierarchy. For each hierarchical level we discuss the details of the context dissemination, management services and autonomic manager components.

Throughout this section several examples of filter rules and context dependencies are given. The examples use the OWL Manchester syntax[8] and SWRL rules. The concepts used within these rules are derived from the ontologies within the knowledge base. This allows the language in which these rules are defined to be dynamically expanded, by extending the ontologies. The ontologies used for the presented examples are described in our previous work [16].

7.5.1 Cloud management

The top level AE has several important responsibilities. First, it orchestrates the interactions across data centers. Second, it negotiates, configures, maintains and monitors federations with other cloud providers.

7.5.1.1 Context dissemination

The root AE is concerned with resource allocation on a data center level. To determine the need for virtual machine migrations across data centers, it continuously

requests statistics about average, maximum and minimum CPU and memory consumption from its child AEs, which operate on the data center level. Obviously, to maintain scalability, the statistics are aggregated over all servers within a data center. This means that under normal conditions, the root AE only maintains a few context values per data center. One of the filter rules associated with this context requirement is defined in the OWL Manchester syntax as follows:

Message **and** hasPayload **some**
(AggregatedMeasurable
and hasEntity **some** Datacenter
and hasType **value** CentralProcessingUnit)

The rule states that the AE is interested in messages that contain context about aggregated measurable values related to data centers. Additionally, the type should be a CPU. A similar filter rule can be defined for the type *MemoryModule*, which would pick up memory-related context.

If one of the average resource values of a data center goes over a pre-specified threshold X_1, a new set of filter rules is dynamically generated based on the change in context, at which the root AE starts requesting more detailed information. More specifically, it instantiates filter rules that request detailed statistics about servers with a load higher than X_2. These statistics include aggregated resource loads of the server itself, but also information of its hosted virtual machines. The filter rule that admits measurements of overloaded resources is the following:

Message **and** hasPayload **some**
(AggregatedMeasurable
and hasEntity **some** CloudServer
and hasValue **some** double[$> X_2$])

The automatic context sensitive generation of filter rules is a task of the context dissemination component. For this, dependencies are introduced between the different context types at different layers in the context continuum. A context dependency from context type A to context type B denotes that A can only be requested if B has a particular value, defined in the dependency. For example, to allow the generation of the above example, the following context dependency is used:

$$\begin{aligned} &\text{HighDatacenterLoad(?load)} \\ \wedge&\text{hasContextDependencyList(?load,?list)} \\ \wedge&\text{hasContextDependency(?list,?dep)} \\ \wedge&\text{hasContextType(?dep,?max)} \\ \wedge&\text{TooHighMaxServerLoad(?max)} \\ \Rightarrow\ &\text{ContextToQuery(?load)} \end{aligned}$$

The concepts HighDatacenterLoad and HighServerLoad are also part of the context dependency and are defined as follows:

HighDataCenterLoad $\equiv$
AggregatedMeasurable
and hasEntity **some** Datacenter
and hasType **value** CentralProcessingUnit
and hasValue **some** double[$> X_1$]

and

HighServerLoad $\equiv$
AggregatedMeasurable
and hasEntity **some** CloudServer
and hasType **value** CentralProcessingUnit
and hasValue **some** double[$> X_2$]

As can be seen, it is in these definitions that the two thresholds, X_1 and X_2, are defined. In this case, the generation of filter rules works as follows. At the data center's AE, each time the aggregated data center load is updated, the context dependency is checked, which may lead to the context type HighServerLoad being classified as ContextToQuery. This leads to the generation of a filter rule requesting HighServerLoad instances. These HighServerLoad instances are then classified at the server's AE, according to the above definition.

The second task of the root AE is to configure and manage federations with other cloud providers. If the average load of all data centers within the cloud becomes too high, the root AE can no longer migrate virtual machines between data centers. The problem can then only be solved by leasing infrastructure from another cloud provider and setting up a cross-domain collaboration. This will trigger several filter rules for gathering the context needed in order to perform the resource re-allocation and negotiate a contract. This includes context about the virtual machines that can be outsourced and billing and availability information of other cloud provider networks.

7.5.1.2 Autonomic manager

As shown, the context dissemination process dynamically adapts itself to changes in the environment. When the autonomic manager needs to perform resource migrations across data centers or cloud infrastructures, the filter rule generation algorithm already made sure that context necessary for executing the resource allocation algorithms is present in the AE's data model. The autonomic manager uses the available context to determine which virtual machine(s) to migrate and select a suitable destination. However, the root AE does not have a detailed view on the destination data center or cloud infrastructure. Therefore, it contacts the AE

responsible for the destination through a suitable management service. This management service triggers the destination AE's autonomic manager, which contains a suitable algorithm, and the required context, to determine to which exact server to migrate the virtual machine(s). The root AE thus governs the migration process, but delegates the actual low-level configuration tasks to its child AEs.

7.5.1.3 Management services

The root AE offers a few management services, which can be used by its own autonomic manager or in the contract negotation process when interacting with other root AEs. First, it offers a composite management service for migrating virtual machines from one data center to another. The service is actually composed of several lower-level management functions. It stops the virtual machine on its source server, releases the reserved resources and contacts the destination data center AE to find a suitable destination server. Second, it contains some management services for setting up federations. For example, it could offer a service that allows other cloud providers to lease part of their infrastructure. These services are used as a basis for the contract negotiation process.

7.5.2 Data center management

At the data center level, AEs are responsible for managing a single data center. However, as the size of a data center grows, scalability can be maintained by allowing several AEs, grouped in a hierarchy, to each manage part of the data center.

7.5.2.1 Context dissemination

Under normal conditions, the data center AE periodically queries its children, for context concerning the aggregated load of their managed server(s). It thus maintains a few context values per server or sub-cluster. This context is needed for several reasons. First, it needs this information to satisfy the context requirements of the root AE, which needs aggregated statistics about the data center resource load. The data center AEs thus use the gathered server- and cluster-granularity statistics to infer data center-granularity context. Second, if the load of one of its servers surpasses the threshold X_2, it needs to request more detailed context of that server in order to prepare for a virtual machine migration. Once the threshold is reached for a server, the data center AE adapts the filter rules of the corresponding server AE in order to request context about the virtual machines of the server.

As stated, when context is passed between levels in the management hierarchy, it is dynamically aggregated and transformed. For basic types of aggregation (i.e., averaging or maximizing values such as the aggregated statistics of the load) the filter rules themselves can be used to perform such transformations. As discussed

in [30], the context model defines operators that allow aggregating the data. The type of aggregation operators supported correspond with those that are possible in a typical database query language such as SQL. For more advanced aggregation (e.g., taking 2nd-order statistics of data), the context should be aggregated by the autonomic manager itself. In this case, the aggregated context is regarded as a new piece of inferred knowledge. Note that the aggregation obviously has an impact on the precision of context at the root of the tree: the more information is aggregated, the less detailed the view a parent AE has on the status of the underlying network. Introducing an additional layer into the hierarchy can therefore also result in a loss of precision if a high amount of aggregation is performed. However, in this case, the context dissemination framework allows requesting more detailed context based on the received aggregated value. As such, the loss of precision can be controlled by carefully specifying contextual requirements.

7.5.2.2 Autonomic manager

It features a reactive management process that corrects the overload of cloud servers by migrating virtual machines from one cloud server to another. Existing resource allocation algorithms can be employed to decide how the migration should be performed [33]. Similar to the root level, the adaptive filter rule generation algorithm makes sure that the necessary context is available when the resource allocation algorithm is triggered.

7.5.2.3 Management services

As stated, one of the responsibilities at the data center level is virtual machine migration. As such, a management function is present to perform such migrations. This function is actually a composition of several low level management functions. More specifically, it consists of the following steps. First, the virtual machine needs to be stopped on the original cloud server and its resource reservations need to be cancelled. Second, the virtual machine needs to be started on the newly selected cloud server and the necessary resource reservations need to be made.

Additionally, the data center AE offers a management service that interfaces with its autonomic manager. More specifically, the root AE needs to be able to trigger the resource allocation process when it migrates a virtual machine from one data center to another. The root AE only determines to which data center a virtual machine should be moved. The data center AE can itself decide which cloud server or servers to use. This management service takes as input a virtual machine and as effect places that virtual machine on one of its servers. Internally, it triggers the data center AE's autonomic manager which uses a resource allocation algorithm to find a suitable server for the virtual machine.

Table 7.1: A service description of a management function that allows AEs to allocate physical resources to a virtual machine

IOPEs	SWRL atoms
inputs	VirtualMachine(?v), QuantifiableResource(?r), xsd:long(?i)
outputs	
preconditions	CloudServer(?s), executes(?s, ?v), hasState(?s, TurnedOn), consistsOf(?s, ?r), hasCurrentValue(?r, ?l), hasMaximum(?c, ?m), swrlb:substract(?a, ?m, ?l), swrlb:greaterThanOrEqual(?a, ?i)
effects	hasReservedResource(?v, ?t), hasEntity(?t, ?r), hasValue(?t, ?i)

7.5.3 Server management

The bottom level AEs are each responsible for a single cloud server. Their main goal is to closely monitor this server's state and configure it through a set of management services.

7.5.3.1 Context dissemination

The server AE directly interfaces with the underlying managed resources. As such, it must be able to communicate with them in a device specific manner. Here we assume that the cloud servers can be queried via the SNMP protocol. In contrast to higher levels, filter rules take the form of SNMP traps instead of semantic definitions. Normally, the server AE periodically queries the CPU and memory load of its underlying server using SNMP GET, which is a pull-based mechanism. Additionally, it sets an SNMP trap that triggers when one of the server's resources reaches the threshold X_2 or X_3 (with $X_2 \leq X_3$). When the threshold X_2 is reached, the server AE starts requesting additional information about the individual virtual machine resource requirements. Note that this change in filter rules is not directly initiated by the server AE, but rather a consequence of a request for more detailed virtual machine context by the data center AE. Finally, if one of the server's resources reaches X_3, a problem has occurred that cannot be solved swiftly enough at the higher levels. This initiates an admission control mechanism. Additional filter rules, that intercept detailed context about individual service requests, which is needed by the admission control component, are instantiated.

7.5.3.2 Autonomic manager

The server AE has a very narrow view on the managed environment. Therefore, it can only offer basic solutions to occurring problems. Its main responsibility is initiating the admission control algorithms when the resource load of the server

passes the threshold X_1. When this happens, all service requests arriving on the server are placed in a queue. Through suitable filter rules, the server AE is notified whenever a new request arrives in the queue. The admission control algorithms decide whether or not to let the request through and notify the server via a suitable management service. Note that admission control is only a last resort. Whenever possible, an overloaded server problem will be efficiently solved at a higher level and admission control will not be necessary.

7.5.3.3 Management services

The server AE forms the gateway to the underlying managed resources. Therefore, it offers a plethora of management functionality that can be used by its own autonomic manager and its parent AE to influence the state of the environment. The offered management services include: changing a server's state, change the resource allocation of a virtual machine, start or stop virtual machines and turn on or off the admission control system. Table 7.1 depicts an example service description modelled using SWRL atoms. It represents a management function that allocates a specified amount of resources, of a specific hardware component to a specific virtual machine. The service takes three inputs: a reference to a virtual machine $?v$, the specific resource $?r$ that should be reserved (e.g., a CPU core or memory module) and the amount $?i$ of the resource that should be reserved. The preconditions are somewhat more complex. The first three state that the server $?s$ that hosts the virtual machine $?v$ should be turned on. The remaining atoms calculate the currently available amount of resources of $?r$ and store it in variable $?a$, Finally, the last precondition states that there should be at least $?i$ resources available on $?r$. The effects state that after successful execution of the service, the requested amount of resources will be reserved for $?v$ on $?r$. Note that by way of SWRL variables, inputs and outputs can be linked to preconditions and effects. For example, the input variable $?v$ is reused in the preconditions and effects. Consequently, the semantic matchmaking algorithms and reasoners know these conditions refer to the same virtual machine as specified in the inputs.

7.6 Results & discussion

The context dissemination process plays a central role in the AE's autonomic control loops. Therefore, its performance and scalability severely impact the management system in general. This section presents an evaluation of the proposed context dissemination framework. First, an overview of our prototype implementation is given. Second, the scaling behaviour of the presented hierarchical approach is studied using analytical formulations. Third, the impact on performance of introducing semantics into the context dissemination process is evaluated using the

presented prototype. The evaluation uses the cloud management scenario detailed in Section 7.5. The contextual requirements, and context dissemination process detailed there are used throughout this section.

7.6.1 Prototype implementation

A prototype AE implementation, based on the presented concepts, was created. The prototype was built in Java, based on the Pellet[9] OWL 2 reasoner version 2.1.1 and OWL-API[10] version 3.0.0. In addition to OWL 2 reasoning, Pellet supports DL-safe SWRL rules [34]. Jena rules are supported through Jena's own built-in rule reasoner, of which version 2.6.2 was used. The embedded information models take the form of OWL ontologies, using the DENON-ng ontology as a basis [35]. DENON-ng is an ontology derived from the DEN- ng information model [36]. Additionally, DENON-ng was extended with a model for representing Cloud Computing specific concepts. A detailed description of these ontologies can be found in our previous work [16]. The Jena TBD high-performance triple store is used as a data model. The context disseminator implementation supports several types of filter rules. They can take the form of OWL 2 class definitions, SWRL rules, Jena rules or SNMP traps. The actual context messages are defined as OWL instances, and are matched with the semantic filter rules using the embedded Pellet and Jena reasoners. The filter rule generation process uses context dependencies defined in SWRL to generate filter rules of different types, using the embedded Pellet reasoner. Service descriptions are also defined using SWRL atoms. A custom service matchmaking algorithm, which uses the Pellet reasoner to determine subsumption relationships between these atoms, was devised and implemented. The autonomic manager supports different types of management algorithms, such as rule-based systems or neural networks. The policy framework and contract repository are currently not implemented within the prototype.

7.6.2 Context dissemination scalability

This section studies the amount of context that is received by AEs throughout the hierarchy and determines how the hierarchical structure can be exploited to improve scalability. At the lowest level of the hierarchy only a single server is managed by the AEs. As such, the amount of generated context depends on factors such as the number of hosted virtual machines and received service requests. As the number of virtual machines that can be hosted on a single physical server is limited, this is not expected to severely impact scalability. On the other hand, at the data center and cloud levels, the amount of context depends on the number of servers within a data center and the number of data centers respectively. When

[9] http://clarkparsia.com/pellet
[10] http://owlapi.sourceforge.net/

the data center is managed by a single AE, it eventually becomes overloaded with context as the number of servers within grows. Therefore, a large data center should be managed by multiple AEs, which are also structured in a hierarchy.

This section presents an analytical model that calculates the amount of context that needs to be processed at different levels within the data center layer. The cloud level model can be similarly defined, but is omitted for brevity. Subsequently, the model is employed to show how scalability of the context dissemination process can be maintained by using hierarchies of AEs. The model is based on the resource allocation use case presented in Section 7.5 and the context requirements defined in Figure 7.3.

7.6.2.1 Analytical formulation

Consider a data center with S servers that host, on average, V virtual machines each. Additionally, R different resources are monitored (e.g., CPU or memory). The data center AE hierarchy consists of n levels, with level 1 the lowest and n the highest data center level. Level 0 thus represents the server level and $n+1$ the lowest cloud level. T_i is defined as the total number of AEs at level i. As such, $T_0 = S$, as every server is managed by a unique AE. C_i is defined as the number of child AEs governed by every AE at level i, while M_i is a configurable parameter that determines the maximum number of child AEs at level i. Based on the input variables S, V, R, n, and M_i, the amount of context that is processed by every AE can be calculated. The context calculation is based on the value of C_i, which in turn depends on T_i. Therefore, T_i is first defined:

$$T_i = \left\lceil \frac{T_{i-1}}{M_i} \right\rceil \tag{7.1}$$

Plainly, the total number of AEs at level i equals the number of AEs at level $i-1$ divided by the maximum number of child AEs at level i, rounded up. Similarly, C_i is defined as follows:

$$C_i = \frac{T_{i-1}}{T_i} \tag{7.2}$$

In other words, the number of child AEs of every level i AE equals the total number of AEs at level $i-1$ divided by the total number of AEs at level i.

Now let N_i represent the total amount of context received by every AE at level i per time period during normal operations. The amount of context is measured using an abstract unit, which counts the pieces of information that are disseminated, such as a single server or virtual machine resource measurement. During normal operations, every AE receives a single aggregated measurement for each resource type every time period from each of its child AEs. N_i is thus calculated as follows:

$$N_i = R \times C_i \tag{7.3}$$

When an overload is detected, additional context is generated. At the lowest data center level, the AEs are in direct contact with server AEs. Therefore, when the load of one of those servers exceeds X_2, it starts requesting one additional measurement for each resource type for each of the server's virtual machines, or a total of $R \times V$. On higher levels, additional context is only requested of servers with load higher than X_2, from child AEs whose managed cluster's aggregated average load exceeds X_1. However, here not only virtual machine resource information is requested but also of the server itself, or a total of $R + R \times V$. Now let P equal the percentage of servers whose measured load exceeds X_2. The amount of additional generated context O_i at level i when a single child cluster becomes overloaded can then be calculated as follows:

$$O_i = \begin{cases} R \times V & i = 1 \\ (R + R \times V) \times P \times \prod_{j=1}^{i-1} C_j & i > 1 \end{cases} \tag{7.4}$$

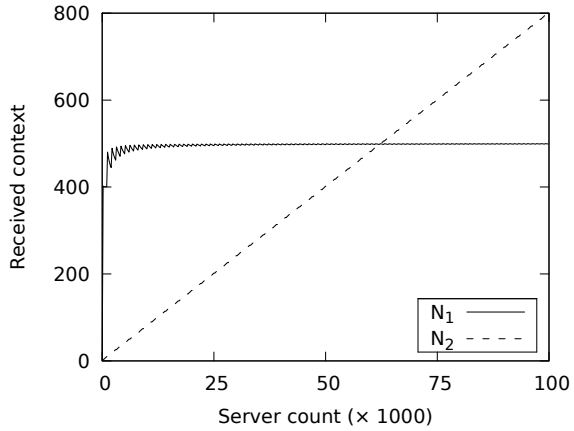
In the second case, the total number of servers within a child cluster is calculated by $\prod_{j=1}^{i-1} C_j$. The percentage of overloaded servers within a child cluster therefore equals $P \times \prod_{j=1}^{i-1} C_j$.

7.6.2.2 Evaluation

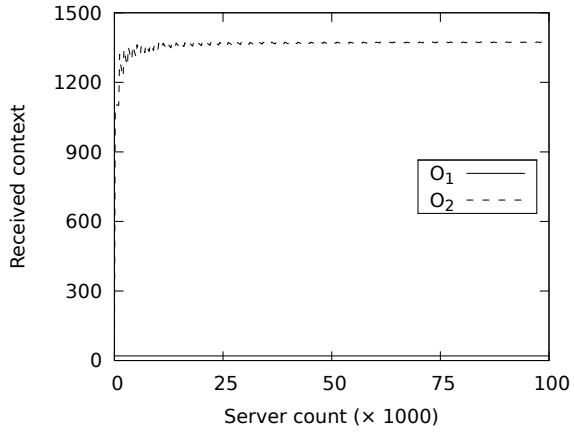
To evaluate the proposed hierarchical model, the parameters, introduced in the analytical model, need to be quantified. We assume $R = 2$, as two resources, CPU and memory, are measured. The average number of virtual machines adds only a constant factor to the results and its value therefore does not matter. For reference, a value $V = 10$ was arbitrarily chosen. The percentage of overloaded servers was set to $P = 25\%$. The evaluated data center consists of up to $100,000$ servers.

Figure 7.4 shows the amount of received context per AE for a two level data center hierarchy, with $M_1 = 250$. Figure 7.4a depicts received context under normal operations, when no child cluster of the AEs is overloaded. When a child cluster of an AE becomes overloaded, additional context is generated. Figure 7.4b shows the amount of additional generated context per overloaded child cluster. Under normal circumstances, the amount of generated context on the lowest level is obviously limited, as there is a maximum limit of 250 server AE children, which limits the received context to $R \times S = 2 \times 250 = 500$ units. In contrast, the second level contains only a single AE, which governs all level 1 AEs within the data center layer. Consequently, its received context increases linearly with the size of the data center, which clearly presents a scalability bottleneck once the data center grows too large.

The depicted results are only valid when context filtering takes place. On the other hand, if all context is propagated upwards, the amount of context received by every AE is significantly higher. For example, in a data center with $100,000$



(a) during normal operations



(b) for an overloaded cluster

Figure 7.4: The amount of context received by data center AEs for a 2 level data center hierarchy

servers, AEs on the first data center level would have to process 5500 instead of 500 units of context. Even worse, the single AE on the second data center level, would need to process detailed context information about all servers at all times, which comes down to $2,2$ million units of context. This clearly illustrates the need for intelligent context filtering.

As shown in Figure 7.4b the additional generated context when a cluster becomes overloaded has an upper limit. However, the generated context reaches that limit as soon as $s \geq M_1$. This results in a high context load even for relatively small networks. To solve this issue, only detailed context information about the

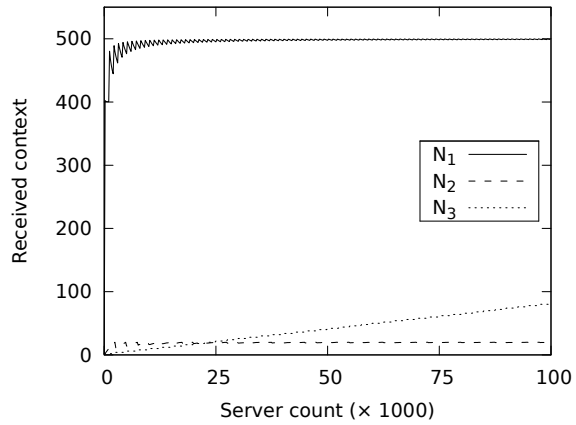
subset of overloaded servers nearest their resource limit is propagated to the top of the hierarchy. At the top level, resource migrations is then performed to solve the overload of these servers. Recursively repeating this process down to the bottom of the management hierarchy for the other overloaded servers solves the entire overload, while circumventing the described bottleneck. For example, if, instead of detailed context about the 25% overloaded servers, only context about the 100 most overloaded servers is propagated upwards, O_2 is reduced from 1375 to 550.

As shown, the top data center level presents a scalability bottleneck in terms of generated context, as an ever-increasing number of servers is governed by a single top-level data center AE. However, this problem can be reduced by adding additional levels to the hierarchy. Figure 7.5a shows how context overhead per AE is significantly reduced by introducing a third data center level into the hierarchy, with $M_1 = 250$ and $M_2 = 10$. The second data center level is now limited to 10 child AEs per AE, thus limiting the overhead generated under normal circumstances. Again, the number of children of the top level (in this case the third), is theoretically unlimited. Therefore, the scalability bottleneck shifts there. However, due to the introduction of an additional level, ten times as many servers are necessary to generate an additional child for the top level AE. Consequently, the received context increases much slower. Once the context received at the top level AE reaches a prespecified limit, a new level can be introduced, effectively allowing unlimited scalability of received context as a function of server count. The problem related to introducing additional hierarchical levels becomes apparent in Figure 7.5b. In this case, the top level AE's children are each responsible for up to 1000 servers, instead of 100 in the 2 level hierarchy. As such, the generated context when an overload occurs has also increased tenfold. Again, a significant reduction can be achieved by limiting the number of servers that are returned to only a fixed amount with the highest load. The resource allocation algorithm could start by migrating virtual machines hosted by these most overloaded servers. The underlying AEs could then themselves solve the problems within their own sub-cluster.

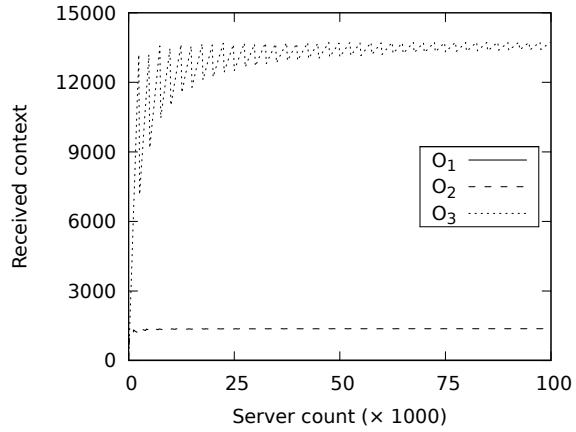
In summary, we have shown how changing the structure of the AE hierarchy can be leveraged to effectively limit the amount of context that must be processed by every AE. More specifically, by increasing the number of levels in the AE hierarchy, the load on the top level AE can be significantly reduced, greatly increasing the scalability of the context dissemination process. As a downside, increasing the number levels in the hierarchy has been shown to slightly decrease optimality of the management algorithms [25].

7.6.3 Semantic reasoning overhead

Semantic reasoning using ontologies is computationally hard. Specifically, it has been proven that the more expressive OWL 2 profiles have an NP-complete (or



(a) during normal operations



(b) for an overloaded cluster

Figure 7.5: The amount of context received by data center AEs for a 3 level data center hierarchy

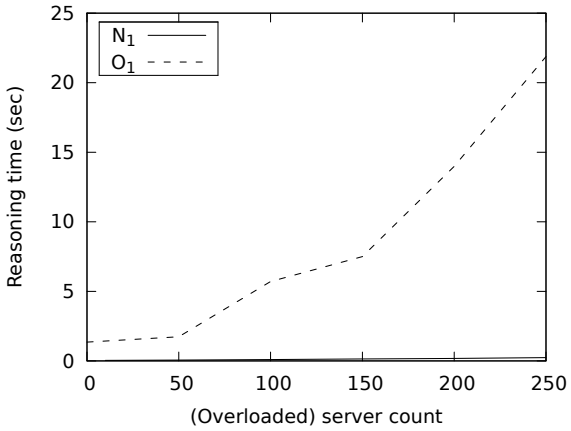
worse) reasoning complexity[11]. Therefore, the reasoning steps of the context dissemination process are expected to impact performance. This section studies the impact on performance by semantic reasoning in the filter rule generation and context filtering steps. More specifically, the total delay introduced by the reasoning processes was measured, which allows us to determine whether or not semantic reasoning is feasible when managing large scale networks. As in the previous section, we focus on the data center level. Determining the reasoning time in a

[11] http://www.w3.org/TR/owl2-profiles/#Computational_Properties

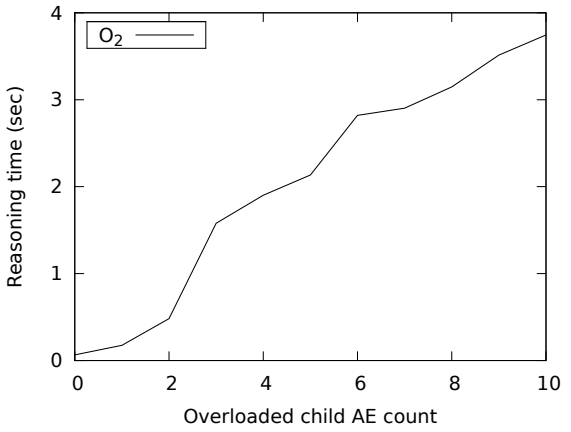
$100,000$ server data center, with a two-level data center management hierarchy, where the lowest level AEs manage up to 250 server AEs and a single top level data center AE manages up to 400 low level data center AEs. In the presented results, Jena rules were used during the filtering process. This choice was made for two reasons. First, Jena and SWRL rules offer greater expressive power than pure OWL 2 class definitions. Second, unlike SWRL in Pellet, the Jena reasoner allows rules to be used without OWL inferencing. Therefore, Jena rules offer better performance for the price of decreased inferencing capabilities. All tests were performed on a server with two dual-core AMD Opteron 2 Ghz processors and 4 GB memory, running Debian 5.0 and Linux kernel 2.6.30.

Figure 7.6 depicts the reasoning time for the filter rule generation process for both levels of the data center hierarchy. Filter rules are generated by the AE itself and then distributed to the child AEs. As such, the bottom data center AEs generate filter rules which are then sent to the server level AEs. The filtering itself is performed by the server AEs. Figure 7.6a presents the reasoning time for generating filter rules for up to 250 server-level child AEs. The curve labelled N_1 shows the reasoning time under normal circumstances (i.e., when none of the servers are overloaded). In this case, the reasoning process takes less than 250ms for up to 250 servers. In contrast, curve O_1 depicts the reasoning time for a growing number of overloaded servers and a total of 250 servers. Even when all servers within the cluster are overloaded, and consequently a huge amount of context is received, determining which filter rules to activate is feasible in less than 30s. However, this is highly unlikely to occur, as the overload should be solved when it starts occurring and not after all servers have become overloaded. In case the overload is limited to a fraction of all servers, filter rule generation takes only a few seconds at most. Figure 7.6b depicts the reasoning time for generating filter rules at the top data center level, for 400 child AEs and as a function of the number of overloaded child cluster. Every cluster is assumed to contain 250 children, of which 25 are overloaded for overloaded clusters. The figure shows that the reasoning time remains under 1s for up to 3 overloaded cluster, or 75 overloaded servers. For 10 overloaded clusters, and thus 250 overloaded servers, the filter rule generator is still capable of generating new filter rules in less than 4s. The increasing reasoning time in both scenarios is due to the increasing amount of knowledge that is being stored into the ontological context model, thus complicating the reasoning process.

The reasoning time for the actual semantic filtering process is shown in Figure 7.7. As stated, the generated filter rules are forwarded to the underlying child AEs, where the actual filtering takes place. The figure thus depicts results for filtering at the server level and the bottom data center level, which forward context to the bottom data center level and top data center level respectively. The server AE maintains a single server. Therefore, Figure 7.7a depicts the reasoning time as a function of the number of virtual machines hosted on the server. The number of



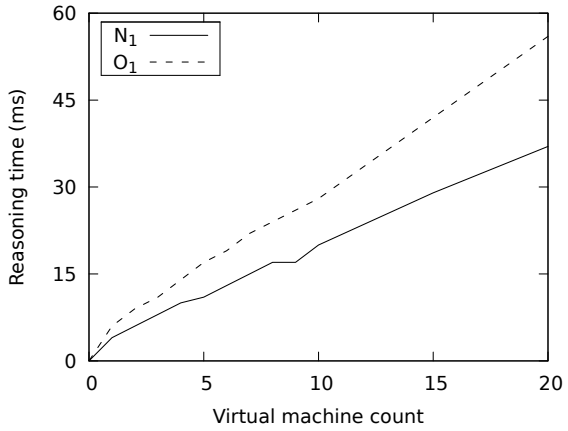
(a) bottom data center AE level



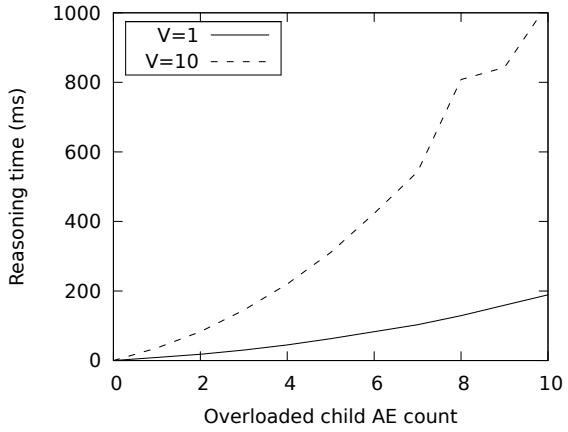
(b) top data center AE level

Figure 7.6: The reasoning time for generating filter rules

virtual machines directly influences the amount of generated context. In an overloaded scenario, with 20 virtual machines, it takes the context disseminator less than 60ms to match the produced context with the defined filter rules. The reasoning time for the bottom data center AEs is shown in Figure 7.7b. At this level, AEs must process and disseminate detailed context of up to 250 servers. Under normal circumstances, only a few aggregated context values are disseminated. However, for every overloaded server within the cluster, the parent AE adds additional filter rules. The figure shows that when 10 servers within the cluster are overloaded, which each host 10 virtual machines, it takes up to 1s to match all produced con-



(a) server AE level



(b) lowest data center AE level

Figure 7.7: The reasoning time for semantically matching filter rules with context

text with the generated filter rules. However, the figure also shows that when these servers only host a single virtual machine, and thus a lot less context is produced, the reasoning time is reduced to 200ms. This means that reasoning time is much more heavily influenced by the amount of produced context than by the number of filter rules.

In summary, the presented intelligent context filtering approach has been shown to be capable of significantly reducing the amount of disseminated context during normal operations. Therefore, the reasoning time of the proposed semantic context dissemination processes can be kept in the order of tens of milliseconds. As

problems occur within the network, more context needs to be distributed throughout the hierarchy, causing an increase in reasoning time. The presented results depict the context dissemination process in an AE hierarchy responsible for managing a $100,000$ server data center. As long as the number of overloaded servers, and thus the amount of additional generated context information, remains somewhat low (i.e., below 50 per cluster), the reasoning time for generating new filter rules becomes at most a few seconds. If an entire cluster becomes overloaded, the reasoning time increased significantly up to 25 seconds. However, under these circumstances it is no longer necessary to generate additional filter rules, as all possible context is already being requested. The filtering process itself is more efficient, and even when up to 250 servers are overloaded, applying the generated filters takes less than 1 second. The results show that it is feasible to execute the filter rule generation process often (i.e., several times per second) during normal operations, and less often (i.e., once every few seconds) when a limited overload occurs, as the filter rule generation process only starts slowing down once these additional filters are in place. Additionally, the presented results show that even for a data center with up to $100,000$ servers, of which a large portion is overloaded, the context filtering process can easily be executed once every second throughout the AE hierarchy.

7.7 Conclusion & future work

This chapter presents a hierarchical framework for efficient and scalable context dissemination in large-scale computing and communications systems. It focuses on intelligent and efficient collaboration and communication between distributed management components, referred to as Autonomic Elements (AEs). Semantic models and reasoners are introduced to facilitate correct understanding and interpretation of information, goals and actions. More specifically, we have shown how semantics can be employed to efficiently filter context, based on its meaning and the management goals of the AEs. As the managed environment changes, context filters are automatically and dynamically adapted by the presented framework. This dynamic process makes sure that the right context, is delivered to the right place, at the right time. Consequently, context-related overhead is significantly reduced, as superfluous information is not distributed among AEs.

The presented approach was applied to the management of large-scale cloud computing data centers. Managing such large-scale data centers presents a significant challenge, as huge amounts of context are generated by the underlying resources. The presented intelligent context filtering framework is therefore well suited to improve management scalability and efficiency in such a scenario. Additionally, the specific context requirements throughout the management hierarchy of this scenario were identified and it was shown how the framework can be applied

to meet them. Finally, an analytical model was introduced to demonstrate how the proposed combination of AE hierarchies and intelligent context dissemination and filtering can be used to significantly reduce context overhead and support significantly increased scaling in terms of data center size.

A prototype implementation of the presented context dissemination framework was evaluated in order to estimate the effects of semantic reasoning on performance. The reasoning time of the filter rule generation and context filtering steps was measured in order to determine the limits of the context dissemination process. It was shown that under normal circumstances, both semantic generation and filtering take only a few tens of milliseconds in a data center of up to $100,000$ servers. As more and more servers become overloaded, the filter rule generation process causes more detailed context to be disseminated throughout the hierarchy. Consequently, the reasoning time increases. However, even in a highly overloaded network, where detailed context about up to 250 servers is distributed, filter rule generation takes only up to 25 seconds and actual filtering less than one second. It was thus shown that detailed context updates can be disseminated at one second intervals. Filter rule generation can be performed several times per second during normal operations (i.e., when it is most useful), and several times per minute once every few seconds during an overloaded period.

This chapter described the fundamental concepts of an intelligent context dissemination and filtering framework. In a fully autonomic context, the framework needs to automatically derive the dynamic context requirements at every level within the management hierarchy. These requirements depend on the specific management processes and algorithms used throughout the hierarchy. In future work, we are planning to close the gap between these management processes on one hand, and the context dissemination framework on the other, thus creating an integrated and fully autonomic solution.

Acknowledgment

Jeroen Famaey is funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT-Vlaanderen) under grant no. 73185; Steven Latré is funded by the Fund for Scientific Research Flanders (FWO-Vlaanderen).

References

[1] B. Jennings, R. Brennan, W. Donnelly, S. Foley, D. Lewis, D. O'Sullivan, J. Strassner, and S. van der Meer. *Challenges for federated, autonomic network management in the future internet*. In 1st IFIP/IEEE International Workshop on Management of the Future Internet (ManFI), pages 87–92, 2009. doi:10.1109/INMW.2009.5195942.

[2] L. Baresi, A. D. Ferdinando, A. Manzalini, and F. Zambonelli. *The CASCADAS framework for autonomic communications*. In Autonomic Communication, pages 147–168, 2009. doi:10.1007/978-0-387-09753-4_6.

[3] J. Strassner, J.-K. Hong, and S. van der Meer. *The design of an autonomic element for managing emerging networks and services*. In International Conference on Ultra Modern Telecommunications (ICUMT), pages 1–8, 2009. doi:10.1109/ICUMT.2009.5345533.

[4] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. *How to enhance cloud architectures to enable cross-federation*. In 3rd IEEE International Conference on Cloud Computing (CLOUD), pages 337–345, 2010. doi:10.1109/CLOUD.2010.46.

[5] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow. *Blueprint for the intercloud – protocols and formats for cloud computing interoperability*. In Fourth International Conference on Internet and Web Applications and Services (ICIW), pages 328–336, 2009. doi:10.1109/ICIW.2009.55.

[6] J. Strassner, J. de Souza, D. Raymer, S. Samudrala, S. Davy, and K. Barrett. *The design of a novel context-aware policy model to support machine-based learning and reasoning*. Cluster Computing, 12(1):17–43, 2009. doi:10.1007/s10586-008-0069-4.

[7] J. Famaey, S. Latré, J. Strassner, and F. De Turck. *A hierarchical approach to autonomic network management*. In 2nd IFIP/IEEE International Workshop on Management of the Future Internet (ManFI), pages 225–232, 2010. doi:10.1109/NOMSW.2010.5486571.

[8] W. K. Chai, A. Galis, M. Charalambides, and G. Pavlou. *Federation of future internet networks*. In 2nd IFIP/IEEE International Workshop on Management of the Future Internet (ManFI), pages 209–216, 2010. doi:10.1109/NOMSW.2010.5486573.

[9] M. Serrano, S. van Der Meer, V. Holum, J. Murphy, and J. Strassner. *Federation, a matter of autonomic management in the Future Internet*. In

12th IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 845–849, 2010. doi:10.1109/NOMS.2010.5488357.

[10] K. Feeney, R. Brennan, J. Keeney, H. Thomas, D. Lewis, A. Boran, and D. O'Sullivan. *Enabling decentralised management through federation*. Computer Networks, 54(16):2825–2839, 2010. doi:10.1016/j.comnet.2010.07.006.

[11] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan. *The RESERVOIR model and architecture for open federated cloud computing*. IBM Journal of Research and Development, 53(4):1–11, 2009. doi:10.1147/JRD.2009.5429058.

[12] R. Buyya, R. Ranjan, and R. Calheiros. *InterCloud: Utility-oriented federation of cloud computing environments for scaling of application services*. In 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP), pages 13–31, 2010. doi:10.1007/978-3-642-13119-6_-2.

[13] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. *A view of cloud computing*. Communications of the ACM, 53(4):50–58, 2010. doi:10.1145/1721654.1721672.

[14] J. Keeney, D. Lewis, and D. O'Sullivan. *Ontological semantics for distributing contextual knowledge in highly distributed autonomic systems*. Journal of Network and Systems Management, 15(1):75–86, 2007. doi:10.1007/s10922-006-9054-5.

[15] J. Keeney, K. Carey, D. Lewis, and V. Wade. *Ontology-based semantics for composable autonomic elements*. In 19th International Joint Conference on Artificial Intelligence (IJCAI), 2005.

[16] J. Famaey, S. Latré, J. Strassner, and F. De Turck. *Semantic context dissemination and service matchmaking in future network management*. International Journal of Network Management, 2011. doi:10.1002/nem.805.

[17] J. Wang, B. Jin, J. Li, and D. Shao. *A semantic-aware publish/subscribe system with RDF patterns*. In 28th Annual International Computer Software and Applications Conference (COMPSAC), pages 141–146, 2004. doi:10.1109/CMPSAC.2004.1342818.

[18] M. Petrovic, H. Liu, and H.-A. Jacobsen. *G-ToPSS: Fast filtering of graph-based metadata*. In 14th international conference on World Wide Web (WWW), pages 539–547, 2005. doi:10.1145/1060745.1060824.

[19] J. Skovronski and K. Chiu. *An ontology-based publish-subscribe framework*. In International Conference on Information Integration and Web-based Applications Services, 2006.

[20] H. Li and G. Jiang. *Semantic message oriented middleware for publish/ subscribe networks*. In Sensors, and Command, Control, Communications, and Intelligence (C3I), volume 5403, pages 124–133, 2004. doi:10.1117/12.548172.

[21] M. Klusch, B. Fries, and K. Sycara. *OWLS-MX: A hybrid semantic web service matchmaker*. Web Semantics: Science, Services and Agents on the World Wide Web, 7(2):121–133, 2009. doi:10.1016/j.websem.2008.10.001.

[22] G. Shen, Z. Huang, Y. Zhang, X. Zhu, and J. Yang. *A semantic model for matchmaking of web services based on description logics*. Fundamenta Informaticae, 96(1):211–226, 2009. doi:10.3233/FI-2009-175.

[23] M. L. Sbodio, D. Martin, and C. Moulin. *Discovering semantic web services using SPARQL and intelligent agents*. Web Semantics: Science, Services and Agents on the World Wide Web, 8(4):310–328, 2010. doi:10.1016/j.websem.2010.05.002.

[24] A. B. Bener, V. Ozadali, and E. S. Ilhan. *Semantic matchmaker with precondition and effect matching using SWRL*. Expert Systems and Applications, 36(5):9371–9377, 2009. doi:10.1016/j.eswa.2009.01.010.

[25] H. Moens, J. Famaey, S. Latré, B. Dhoedt, and F. De Turck. *Design and evaluation of a hierarchical application placement algorithm in large scale clouds*. In 12th IFIP/IEEE International Symposium on Integrated Network Management (IM), 2011. doi:10.1109/INM.2011.5990684.

[26] J. Strassner. *Policy-based network management – solutions for the next generation*. Morgan Kaufman, 2004.

[27] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. *Towards a complete OWL ontology benchmark*. In 3rd European conference on The Semantic Web: research and applications (ESWC), pages 125–139, 2006. doi:10.1007/11762256_12.

[28] D. Harrington, R. Presuhn, and B. Wijnen. *An architecture for describing simple network management protocol (SNMP) management frameworks*, 2002. http://www.ietf.org/rfc/rfc3411.txt.

[29] J. Famaey, S. Latré, J. Strassner, and F. De Turck. *An ontology-driven semantic bus for autonomic communication elements*. In 5th IEEE International Conference on Modelling Autonomic Communication Environments (MACE), pages 37–50, 2010. doi:10.1007/978-3-642-16836-9_4.

[30] S. Latre, S. van der Meer, F. De Turck, J. Strassner, and J.-K. Hong. *Ontological generation of filter rules for context exchange in autonomic multimedia networks*. In 12th IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 575–582, 2010. doi:10.1109/NOMS.2010.5488448.

[31] S. Davy, B. Jennings, and J. Strassner. *The policy continuum – policy authoring and conflict analysis*. Computer Communications, 31(13):2981–2995, 2008. doi:10.1016/j.comcom.2008.04.018.

[32] M. Charalambides, P. Flegkas, G. Pavlou, and J. Rubio-loyola. *Dynamic policy analysis and conflict resolution for diffserv quality of service management*. In 10th IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 294–304, 2006.

[33] F. Wuhib, R. Stadler, and M. Spreitzer. *Gossip-based resource management for cloud environments*. In 6th International Conference on Network and Service Management (CNSM), pages 1–8, 2010. doi:10.1109/CNSM.2010.5691347.

[34] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. *Pellet: A practical OWL-DL reasoner*. Web Semantics: Science, Services and Agents on the World Wide Web, 5(2):51–53, 2007. doi:10.1016/j.websem.2007.03.004.

[35] M. J. Serrano, J. Serrat, J. Strassner, and M. Ó Foghlú. *Management and context integration based on ontologies, behind the interoperability in autonomic communications*. In International Conference on Complex Open Distributed Systems (CODS), 2007.

[36] J. Strassner, J. N. Souza, S. van der Meer, S. Davy, K. Barrett, D. Raymer, and S. Samudrala. *The design of a new policy model to support ontology-driven reasoning for autonomic networking*. Journal of Network and Systems Management, 17(1):5–32, 2009. doi:10.1007/s10922-009-9119-3.

8

Conclusions and research perspectives

"Everything should be made as simple as possible, but not simpler."

– Albert Einstein (1879–1955)

Since its inception, the Internet's underlying infrastructure has evolved significantly in terms of size and complexity. Additionally, the end-to-end Quality of Service (QoS) requirements of its services have become exceedingly more stringent. These factors have greatly contributed to the ever-increasing complexity and cost to manage and configure the Internet and its services. This dissertation aims to advance research, in the area of network and service management, on addressing the ever-growing management complexity of multimedia services with stringent end-to-end QoS requirements. The problem is tackled in two steps. First, the problem of offering end-to-end QoS guarantees on the Internet is explored. Second, a solution for the increased management complexity, resulting from the Internet's evolution in size and service requirements, is investigated.

8.1 End-to-end Quality of Service guarantees

The Internet is often referred to as a *network of networks*. However, with the exception of inter-domain routing, there is little to no interaction between these interconnected network domains. Multimedia content delivered over the Internet usually traverses several such domains. In order to guarantee the end-to-end

QoS requirements of modern multimedia services, there is however need for coordination and collaboration between the traversed domains. Federated network management aims to support such inter-domain coordination. A major goal of this dissertation is to offer a set of architectural and algorithmic components to facilitate federations for the delivery of QoS-constrained multimedia services across the Internet.

Chapter 2 investigates state of art research on federated management of the Future Internet and identifies several related challenges. A detailed survey on research efforts towards tackling these challenges is provided, which allows to us identify open issues. The following identified open issues are most relevant to this dissertation:

- The scalable identification and discovery of the network domains and shared resources that need to be incorporated into a federation, based on the federation's dynamic goals.
- Automated mechanisms that facilitate the semantic interoperability between network domains, without the need to align intra-domain models and semantics.
- Multilateral negotiation protocols, that allow federal agreements to be negotiation between many parties simultaneously, without the need for a trusted centralized negotiation authority.
- Distributed conflict detection and resolution algorithms to find and solve conflicts between intra-domain policies and federation-wide goals.

Chapter 3 presents a significant step towards supporting federated multimedia service delivery on the Internet. A framework for the negotiation and configuration of federations between content providers, core Internet routing domains, cloud-based storage sites, and service providers is presented. The negotiated federal agreements allow the content provider to guarantee the end-to-end QoS requirements of the service providers. The chapter additionally presents an algorithm that determines the optimal set of network domains and resources to include in the federation, aiming to minimize the total cost for the content provider, while satisfying the QoS goals set by the service providers. This algorithm thus answers three important questions related to federations: who will take part in the federation, what capabilities should be shared and how should they be configured? Results show that the presented approach is capable of significantly reducing delivery costs compared to traditional end-to-end QoS reservation mechanisms. It achieves this by the inclusion of intermediary storage sites within the delivery chain. The use of dynamically deployed content caches within these sites results in a cost reduction of up to 80% in the evaluated scenarios. The significance of the cost reduction

does depend on external factors, such as the relative cost for storing an item compared to the cost for transmitting it and the vicinity in the network of storage sites to the service providers. Additionally, as the caches are deployed inside the network, they can be shared among different service providers. The evaluation proves that cache sharing can result in an additional significant cost reduction for service providers positioned near each other in the network. In the investigated scenarios, cache sharing resulted in an additional reduction of up to 46%.

Chapter 4 further elaborates on the caching aspect itself. A novel predictive cache replacement strategy is presented. The strategy's cache replacement decisions are based on the predicted future popularity of content, instead of directly on historical information. An in-depth evaluation of the use of prediction in caching, based on simulation results, is conducted. The simulations use request traces from an actual Video on Demand (VoD) service, adding to its realism. The results show that the proposed algorithm's prediction accuracy is impacted by the length of the historical trace, which hinders the accurate prediction of new content. Additionally, the theoretical performance bounds of predictive caching are compared to the theoretical optimum, as well as the traditional Least Frequently Used (LFU) strategy. Assuming that the future can be perfectly predicted, the proposed predictive strategy's performance is between 4 and 25% lower than the global optimum and between 17 and 22% higher than LFU. Taking into account prediction errors, the reduction compared to the optimum increases to between 12 and 33%, while the gain compared to LFU is reduced to between 5 and 10%.

8.2 Management complexity of the Internet

The ever-growing management complexity of modern communications networks makes it increasingly more difficult and costly for human network operators to efficiently and optimally manage the underlying resources. In a quest to resolve this issue, the autonomic network management paradigm has been proposed. It aims to reduce the management and configuration complexity for human operators, by introducing self-governing management components, called Autonomic Elements (AEs), in the network that take over part of the management responsibilities. To ensure scalability, it is expected a huge number of distributed AEs will need to communicate and cooperate in order to achieve the Internet's management goals. The second major goal of this dissertation, is to devise a scalable collaboration framework to facilitate the interaction between AEs, both within and across network domain boundaries.

A first step towards a scalable AE collaboration framework is presented in Chapter 5. It proposes to structure AEs in a hierarchical fashion. The layers of this hierarchy map well to the management responsibilities of AEs. At the bottom of the hierarchy, AEs directly manage and configure a small set of resources.

As they have a very narrow view, they can maintain detailed context information without endangering scalability. They summarize and aggregate this information before forwarding it to their parent AE. Higher up in the hierarchy, the view of the network becomes wider, but the information less detailed. This ensures scalability throughout all layers of the architecture. Additionally, it enables both small-scale detailed configurations, as well as large-scale management optimizations. The quantitative merits of the proposed hierarchical architecture are proven based on the evaluation of an analytical model. The evaluation shows that organizing AEs in a hierarchical topology significantly increases scalability compared to the use of a flat distributed management structure. Specifically, the amount of management information that needs to be processed by a single AE is significantly lower in the hierarchical topology. Moreover, the increase in terms of management overhead as a function of size of the managed environment is considerably higher when using a flat topology. Concretely, in the evaluated scenario, the overhead per AE in a flat architecture was up to 8 times as high as in the novel hierarchical framework.

Chapter 6 focusses on a different aspect of the communication between distributed AEs. It presents the Semantic Communications Bus (SCB), which facilitates the scalable dissemination and intelligent filtering of semantic information. Additionally, it incorporates an algorithm to match AE requirements with management functionality offered by other AEs. Finally, it contains a set of semantic models, in the form of ontologies, which aid in the semantic interoperability of AE interactions, within and across network domain boundaries. A prototype implementation of the SCB is used to determine the overhead of using semantics on the performance of information exchange and service matchmaking. It is evaluated in combination with several semantic reasoning methods. Results show that depending on the expressiveness of the method, generated overhead can indeed significantly impact performance. As such, it is important to correctly determine the required level of expressiveness and use less expressive approaches when possible. Concretely, the prototype is capable of matching a message with a filter rule in a matter of milliseconds when disabling advanced inferencing, while it takes several hundred milliseconds when using advanced ontological reasoning techniques. Additionally, the matchmaking algorithm can determine compatibility between a management goal and a service description in less than 10 milliseconds.

Chapters 5 and 6 focus on the scalable interactions between AEs in a generic network management scenario. In contrast, Chapter 7 combines the proposed generic concepts into a concrete solution for the scalable management of large-scale federated cloud computing data centers. Specifically, the chapter presents a hierarchical version of the previously presented SCB. An analytical model and a prototype implementation are used to validate the proposed solution. The results of this evaluation prove the viability of our approach to overcome the increasing management complexity of the current Internet. Concretely, the analytical model

shows that the amount of information that needs to be processed by every AE can be effectively limited by increasing the number of layers in the hierarchy. This ensures scalability under an ever-growing number of managed servers and services within the data center. This is achieved through a combination of the hierarchical nature of the architecture as well as the intelligent filtering component of the SCB. The prototype implementation is used to explore the viability of using semantics in the information exchange process. It was shown that both generating filter rules and performing the actual semantic filtering can be done in a matter of milliseconds, for data centers containing as many as $100,000$ servers.

8.3 Research perspectives

This dissertation offers several contributions towards an autonomic and federated network management framework for the Future Internet. However, research is still ongoing, and several open issues remain to be solved.

8.3.1 Discovery of network domains and capabilities

This dissertation answered two important questions concerning the negotiation of federations: "Who to include in the federation, and what capabilities to share?" However, the presented framework implicitly assumes the set of candidate network domains and shareable capabilities are known. As the Internet consists of ten thousands of interconnected network domains and each of them potentially contains huge amounts of resources, it is infeasible to assume information about all these network domains is readily available. As discussed in Section 2.4.4, to enable dynamic and automated negotiation of federation agreements, a mechanism is needed to discover a subset of relevant network domains in a scalable and distributed manner. The mechanism should be able to select candidate network domains based on the goals of the federation. For example, in the case of end-to-end QoS guarantees, the network domains along a selected set of end-to-end paths through the Internet should be returned. Additionally, in this dissertation, the capability reservation problem was solved specifically for the end-to-end delivery of multimedia content. The necessary capability types were thus known in advance. This is not the case in a generic scenario. As such, algorithms are needed to automatically map generic federation goals to a set of capabilities offered by the federation partners.

8.3.2 Inter-domain information models

Information models facilitate the unambiguous interpretation and understanding of information between AEs, as well as across management domains. The proposed

SCB uses ontology-based information models to achieve this. This allows AEs, distributed across different network domains, to communicate and collaborate with each other. It was implicitly assumed that these models can be easily made available throughout all federated network domains. As stated in Section 2.4.2, this is far from trivial and mechanisms are needed to map diverging semantic information models unto each other. Only then can true collaboration across independent administrative domains be achieved.

8.3.3 Optimized delivery of multimedia services

This dissertation presents novel cache replacement strategies in order to reduce the bandwidth requirements of modern multimedia services. The amount of Internet traffic originating from multimedia services is only expected to increase even more in the future. In order to limit their bandwidth requirements, further optimization of their delivery process is thus necessary. Current PhD and project research within our group is exploring the optimization of video delivery on the Internet, based on the combination of HTTP Adaptive Streaming (HAS) and Scalable Video Coding (SVC). The ultimate goal of this work is to increase the number of admitted video streams, while maintaining acceptable Quality of Experience (QoE) for all users.

8.3.4 Generic self-governance

The work on autonomic network management presented in this dissertation focussed on the interaction between distributed AEs necessary to achieve global self-governance. The actual self-governing behaviour was outside the scope of our research, but is nevertheless a very important step towards a truly autonomic network management architecture for the Future Internet. Self-governance is achieved through autonomic control loops, which monitor the environment, identify problems and sub-optimal behaviour, formulate a solution and convert it to actual configuration changes in the underlying resources. Existing work on self-governance has always focussed on autonomic algorithms for very specific management problems. However, these specific algorithms need to be combined into a generic self-governing management entity. It needs to be able to determine which management algorithms to utilize, based on the current state of the environment. This would facilitate global optimizations, transcending specific management problems. Ongoing PhD research within our research group therefore focusses on the design of generic autonomic control loops and the interactions between them.

8.3.5 Policy refinement

The autonomic control loops of AEs are governed by a set of management policies. These policies are defined by human operators and specify the bounds within

which the network should operate. A single policy can be defined in different ways. For example, the same policy could be specified as a high-level business goal, or a set of low-level device configurations. The proposed hierarchical network management architecture is well equipped to handle these different representations of the same policy. AEs at the top of the hierarchy are guided by the high-level business goal representation, while bottom level AEs use low-level device configurations. As the goal of autonomic network management is to reduce complexity for human operators, they should not have to specify all representations of a single policy. Instead, they should be able to define the high-level business goals, which are then automatically refined and translated into other representations. Although some research exists on this topic, a generic policy refinement and translation methodology is yet to be invented. This topic will be further investigated in a recently started research project. The results will be reported upon in future publications.

A

A novel context authoring process for federated autonomic management

S. Latré, J. Famaey, J. Strassner, and F. De Turck

The stringent Quality of Experience requirements of services in the Future Internet trigger the need for an intelligent network management approach that is able to provide an end-to-end Quality of Service guarantee. Since service provisioning typically spans multiple network domains, there is a need for a federation of management components. While a federated solution offers important advantages in terms of scalability, it complicates the exchange of context between different nodes, as each node requires contextual data from other nodes to reason upon and perform its management tasks. This is further complicated by the fact that the required contextual data of a management function can change depending on the overall context. Therefore, there is a need to govern the communication of contextual data between the network components in a federated environment. In this appendix, we propose a context authoring process that is able to automate the context exchange between nodes in a federated management environment. The context authoring process enables the automatic generation of filter rules that define where, how and when contextual data needs to be requested from other nodes in a federation. This is achieved through an ontological approach where a semantic reasoner can deduce a new set of filter rules when the context changes.

A.1 Introduction

In recent years, the Internet has evolved from a best-effort packet forwarding service towards a service-oriented delivery that may require strict service delivery quality requirements for certain applications. New advances, such as cloud computing and multimedia services, have resulted in more advanced uses, including migrating the processing power of encoding and transcoding videos to data servers, which can also stream the videos later on. Together with these more advanced services comes an important increase in management complexity. These new services have larger quality requirements and require a minimum Quality of Service (QoE) level to be constantly maintained, even when anomalies occur in the network or at the server level.

To deal with this growing complexity, various approaches are adding more intelligence into the network, either in the form of autonomic communications [1] or cloud computing [2]. The goal is to provide differentiated Quality of Service (QoS) a self-governing network that is able to react to changes in the network as well as in the environment by reconfiguring appropriate network elements. Automation is one way to manage this growing complexity but, as the behaviour of the network components themselves becomes more automated, so does the communication between these components. Therefore, there is a need to govern the communication between the network components as well as with external management elements to provide and share information required for performing autonomic network management.

Network management components, which in the autonomic networking paradigm are often called Autonomic Elements (AEs), are typically distributed amongst different physical machines or even management domains. Together, these management components form a federation that is responsible for providing QoS guarantees to the services it manages. In a federation, these loosely coupled AEs need to communicate with each other to make sure that they can provide the desired QoS levels. Each AE governs the managed resources using their own control loop, and communicates its decisions to other interested elements in a federated autonomic network (e.g., when one autonomic element wants to inform other elements in the federation of its decisions). The communication between components can also be local: several autonomic element architectures [3, 4] are centred around a bus that forwards information from one component in the element to the other interested elements. In all these examples, the communication exchange consists of getting pieces of information, which we call context, from one element to the other.

Exchanging context in a federated management environment is typically performed through the publish-subscribe paradigm, which allows consumers to express their interest in context through filter rules [5]. A producer should only send

context to a consumer if the context complies with the consumer's filter rules. These filter rules define what type of context producers need to be sent to the consumers, including when it is sent (e.g., when a threshold is exceeded) and how (e.g., if it should be averaged). These filter rules are normally static and defined by the consumer interested in the context. In an autonomic management scenario this static definition of filter rules is sub-optimal: manually defining filter rules is a time-consuming task and, moreover, in an environment where the behaviour of network components can change, so can the required filter rules. The challenge lies in automating the exchange of context through the dynamic generation of (context-sensitive) filter rules.

In this chapter, we address the following research questions: (i) Can we describe the context requirements of federated autonomic components in such a way that the generation of filter rules can be automated? (ii) Can this generation benefit from the actual context in order to minimise the overhead of context exchange? and (iii) What is the introduced reasoning overhead of such an automated generation? Consequently, the contributions of this chapter are threefold: we propose a context authoring process that enables the automation of context between federated network components. This context authoring process uses semantic reasoning, so that policies can be defined that govern the context exchange process. Finally, the context authoring process supports several summarisation mechanisms that reduce the amount of data needed for reasoning, and consequently reduce the reasoning overhead.

The remainder of this chapter is structured as follows: Section A.2 discusses similar work in the field of context modelling in network management and the use of publish-subscribe paradigms. The role of exchanging context in a component-based autonomic architecture is discussed in Section A.3, and the need for automating this context is described through an illustrative use case of multimedia video processing and delivery in Section A.4. Section A.5 discusses the context authoring process itself, while Section A.6 presents detailed evaluation results of the performance of the proposed approach.

A.2 Related work

We use the following definition of context from DEN-ng [6]: The Context of an Entity is a collection of measured and inferred knowledge that describe the state and environment in which an Entity exists or has existed. In particular, our definition emphasises two types of knowledge: facts (that can be measured) and inferred data, which results from machine learning and reasoning processes applied to past and current context. It also includes context history, so that current decisions based on context may benefit from past decisions, as well as observation of how the environment has changed.

The inclusion of context can increase the flexibility and intelligence of many applications. This is argued by [7], which describes the development of a software architecture to support the building of context-aware applications by defining a conceptual framework. While the framework simplifies the task of acquiring and delivering context to applications, it does not provide an information model to visualise or define how context is used, nor does it describe how policy is used. In [8] a first step towards such an information model is proposed through an ontology for describing concepts for context-aware applications. However, it is designed to support context negotiation, not general purpose analysis.

The authors of [9] propose an integrated framework for defining domain-specific constraints using an ontology for constructing association rules. Their approach features pruning and generalisation. Our approach is different, in that (1) it uses data from models as well as ontologies, (2) their approach does not model the antecedent and consequent, whereas ours does, and (3) their constraints are not directly integrated into the mining algorithm, whereas ours are. [10] describes a context-aware policy management abstraction called Context-Aware Management Domains, which define a common set of policies to apply to a grouped set of entities based on their context. In this sense, it is very similar to FOCALE's context-aware policy rules [11]. The work presented in this chapter uses the principles of [11], but adds a more specific ontology to produce filter rules.

In [12], a model is proposed for representing context based on a 4-tuple (who, what, where, and when), which they call knowledge atoms that exist as tuples in tuple spaces. This is in reality very similar to how the DEN-ng information model was designed, as it uses software patterns to represent context and different aspects of context that define intelligent containers for storing facts and inferences from external sources [3, 11, 13]. However, the DEN-ng approach uses all six questions from the Zachman Framework to describe knowledge [14] (What, Where, When, Why, Who and How), and is thus more robust than [12].

Specifically to the exchange of context in a federated management environment, the use of the publish-subscribe paradigm is used by work in [15], where filter rules are used as the basis to form a Knowledge Based Network (KBN). We believe the work presented in this chapter complements this approach: while the KBN work focuses on semantic clustering of ontologies [16] and extending current solutions with semantic constructs [17], we focus on the automatic generation of filter rules through an ontological approach to provide information for KBNs.

A.3 Exchanging context in a federated architecture

In this section, we discuss the role of exchanging context in a network management environment. Figure A.1 illustrates how, in the autonomic communications paradigm, more intelligence is introduced into the network and data center side

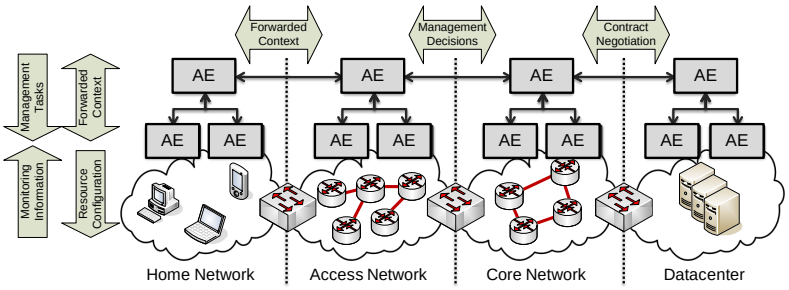


Figure A.1: Overview of a hierarchical network management environment. Arrows represent context exchange between the components.

through a hierarchical organisation of autonomic elements (AEs) [18]. Each AE is composed of a set of managed entities, which are either managed resources or AEs themselves. Managed resources are oblivious to the autonomic nature of the network, and fully depend on their parent AE to govern their management decisions. Child AEs are guided by their parent, but also have some limited autonomic decision-making capabilities of their own. The different parent AEs form a federation and need to agree on how they will manage the network as a whole [19].

It is clear that an efficient orchestrated communication between AEs is crucial for a well performing autonomic management system. Context is a vital part of any autonomic system. It is used to model the current state of the managed entities, which in turn allows the system to adapt to changes when necessary. We define the context of an entity as a collection of measured and inferred knowledge that describe the state and environment in which the entity exists or has existed. As such, a variety of context can be exchanged between remote components at every level of the hierarchy. First, at the lower levels of the hierarchy, the managed resources will be monitored; the resulting monitoring information can be regarded as context. Also, instructions to change the configuration of a specific resource can be seen as context. Second, between parent and child AEs, the parent AEs are responsible for governing the behaviour of their child AEs. As such, a parent AE can delegate management tasks to a specific child AE: these instructions are context that can also be of interest to other child AEs (e.g., to know what neighbouring AEs need to do). Additionally, context information from higher level AEs can be propagated down to lower level AEs. In the opposite direction, child AEs have the responsibility of forwarding context to their corresponding parent AEs. Finally, on the highest layer, there is no hierarchical relationship between AEs, which means that peering AEs need to mutually agree on the management behaviour they utilise. This happens through contract negotiation [19] between peering AEs: the execution of these contracts also involves the exchange of context such as monitoring

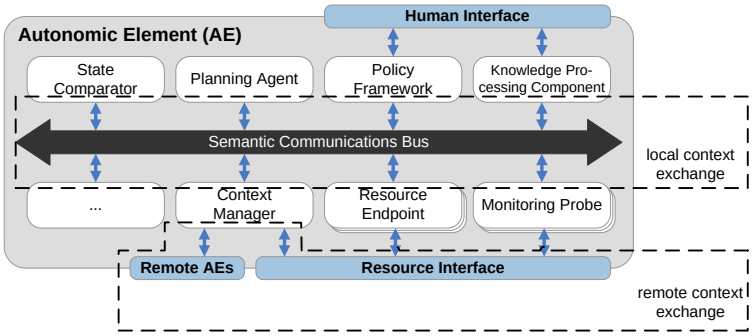


Figure A.2: A bus-driven autonomic element. The components connected to the bus are loosely coupled, and can communicate through the exchange of context.

information of services in a peering domain.

Context exchange can also occur inside an autonomic element; Figure A.2 shows the inner architecture of a bus-driven autonomic element. The functionality implemented by the example elements is roughly identical to that of the Focale [3] or Cascadas [4] control loops. The contextual exchange process that takes place inside an AE can either be internally or externally originated. Externally originated context is context that is produced by remote AEs and that is of interest to one or more components in the local AE. Internally originated context is context that is produced by one component of the local AE and is subsequently forwarded to other components inside the same AE. In this chapter, we focus on the context exchange process between AEs. However, the same concepts can be applied for intra-AE context exchange as well.

It is the role of the context manager to govern both the internal and external context exchange processes by instantiating the most appropriate filter rules on the appropriate locations. These filter rules define what type of context a subscriber is interested in, when this context is needed (e.g., when a threshold is exceeded) and how it should be delivered (e.g., averaged, or as a data series). Figure A.3 shows the internals of the context manager. As discussed, the context manager orchestrates the context exchange process by generating the appropriate filter rules using a publish-subscribe paradigm. Instead of requiring the manual specification of these filter rules, the context manager uses a context authoring process that exploits knowledge from the information model as well as applicable data models and ontologies to automatically generate these filter rules.

The context manager is responsible for generating both local and remote filter rules. The context manager instantiates filter rules to process information received by components that produce contextual data, which then push filtered context once it becomes available. When a push mechanism is not possible, a pull method can also be used as an alternative. The generated filter rules are defined in a

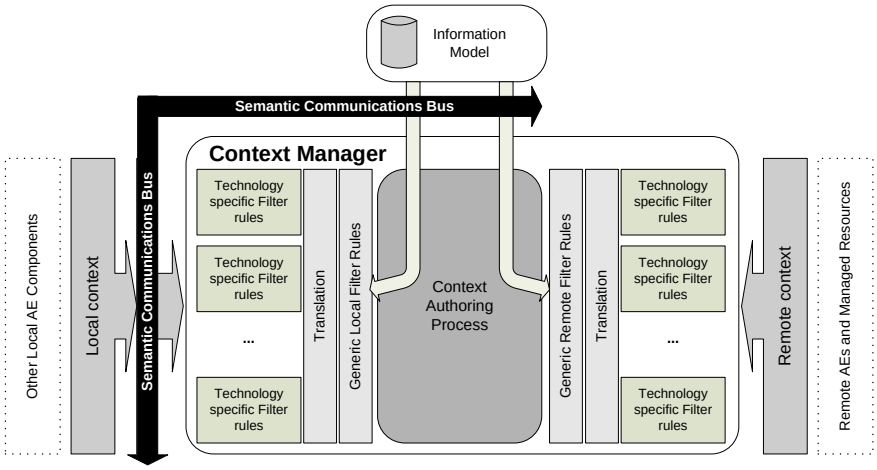


Figure A.3: Details of the context manager within a bus-driven autonomic element illustrated in Fig. 2. By exploiting knowledge from the information model, a context authoring process may generate both local and remote filter rules to enable the context exchange.

technology-neutral language, and can be later on translated to multiple technology-specific mechanisms depending on the implementation.

A.4 Use case: multimedia video delivery

To provide a better understanding of how the context manager is used in federated managed network environments, we describe the context exchange process for an important use case, which will be used throughout the remainder of this chapter. In this use case we assume that a set of critical cloud applications, offering multiple multimedia services to its end users, are hosted on a remote data server. These multimedia services consist on one hand of the capturing, encoding and transcoding of videos and on the other hand of the streaming of the videos in various quality levels to end users. Such multimedia services [20] are already being offered in cloud environments as of today. This use case introduces several management challenges: as video encoding requires considerable resources, cloud infrastructure needs to be managed to enforce load balancing and adequate application placement. Moreover, as multimedia services typically have strict quality requirements in terms of packet loss, delay and jitter, federated end-to-end management is needed to guarantee optimal delivery from the cloud-based multimedia servers to its end users.

To offer such a federated end-to-end network management, we assume that a hierarchical autonomic element architecture is used as illustrated in Figure A.1.

This enables the use of different administrative policy rules that are used to govern different domains to be enforced by a similar topology of autonomic elements. In our example, the top-level domains are the home network, access network, core network and cloud infrastructure. We focus on the context exchange process of the cloud infrastructure's root AE. This AE communicates with the root AEs of other management domains and its child AEs.

Management tasks that we will focus on include load balancing of the cloud servers and path reservation in the network that connects the data server with the home network. Load balancing in the dataserver is achieved by migrating processes to other servers when servers become overloaded. The path reservation is triggered by the cloud's root AE: additional resources on a path can be requested from other network AEs, which may reply either positively or negatively.

We assume that a set of management algorithms is deployed on the cloud's root AE that performs these management tasks. In terms of context, this management algorithm initially only requires basic monitoring information describing the overall status of each management domain. This context is used to trigger the detection of a lack of resources in the network or overloaded servers in the cloud. The context manager translates these initial context requirements to filter rules based on a description of the various entities (e.g., both management and operational data as well as definitions of managed resources) in the information model that are used in the management algorithms. The translated filter rules specify that the monitored video quality score on the home network and the network and server load need to be forwarded to the cloud infrastructure's AE on regular intervals (e.g., every second).

There are many circumstances in which these initial filter rules may change: one of them is a change in context triggered by either a locally or remotely monitored value. For example, suppose that an application server in a cloud becomes overloaded. Figure A.4 illustrates the sequence of events that occur, starting from the initial change in context up to the deployment of a new set of filter rules. As stated, the cloud infrastructure's AE receives regular context updates of each of the AEs that it communicates with (steps 1-4). The context that is contained in this context update is received by the cloud infrastructure's AE Knowledge Processing Component, which examines the context and forwards it to the control loops and context manager. In this example, these components have previously expressed their interest in remote monitoring updates that have been processed using local filter rules. We assume that, at a given time, the home network's AE detects bad video quality offered from one of the multimedia applications in the cloud (step 5). This bad video quality score triggers a reactive reasoning process in the knowledge processing component that is implemented in one or more control loops that tries to solve this problem (step 6). In this case, the problem is solved by migrating one of the applications to a second cloud server. The process migration is initiated

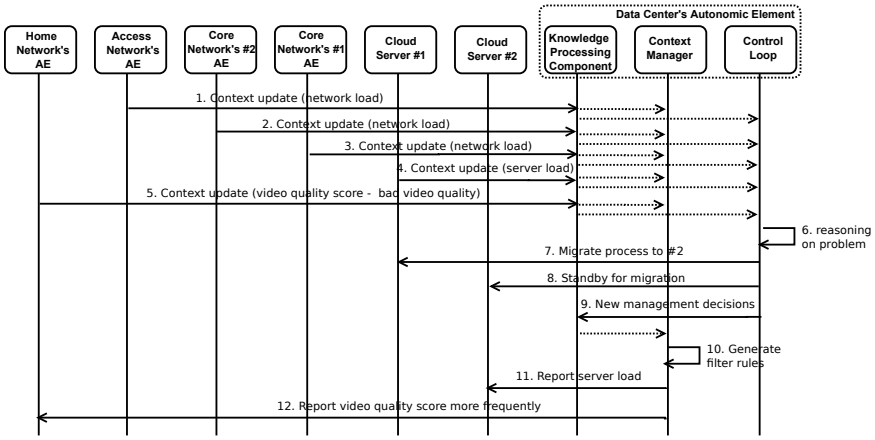


Figure A.4: Sequence diagram illustrating the exchange of context in a federation with two core networks an access network and home network. The sequence diagram illustrates how a change in filter rules is triggered.

from server #1 (step 7) to server #2 (step 8) and at the same time, the knowledge processing component is informed about this change (step 9). The context manager also receives this information, which triggers a reasoning task in the context authoring process of the context manager. Through reasoning, new filter rules are generated based on the change in the management decisions (step 10). In this case, two new filter rules are generated. First, the context manager informs the server about all changes and also asks the server to start reporting its load as well, as it wants to monitor its performance (step 11). Second, because a problem has occurred in the near past, the knowledge processing components want to closely monitor the delivery quality (through adjustments using the control loop) and instruct the home network's AE to send updates about the video quality score at a higher frequency (step 12).

A.5 Context authoring process details

In this chapter, we focus on the filter rule generation process (step 9 in Figure A.4). We use an ontological context model to describe, in formal terms, the contextual requirements of the management components and the applications that they are governing. Furthermore, we provide a three step context authoring process that (1) deduces information from typical reasoners, such as rule-based reasoners, and stores them in the context portion of the model, (2) performs intelligent filtering on the stored data to minimise the overhead and (3) automatically generates the required filter rules based on the context model and additional policies that are

applicable for that particular context.

A.5.1 Detailed process requirements

Context authoring processes in autonomic elements are responsible for orchestrating responses to changes in local and remote context in an autonomic network management environment. It uses the knowledge available in the information model, as well as instance data stored in data models derived from the information model and inferencing from one or more ontologies, to generate filter rules. In our approach, we use an information model to define knowledge in a technologically neutral form; this is then instantiated in one or more data models, as explained in [3]. We augment this with ontological data to be able to reason about facts captured in the models. In this sub-section, we present a number of requirements for the proposed context authoring process.

First, as the context authoring process must generate the filter rules, it must be able to effectively characterise the requirements AEs have in terms of context exchange. This is the basis for the context authoring process: the components within an autonomic element must be able to specify what type of context they need, how they want it to be delivered and under what circumstances. To this extent, the context authoring process relies on an ontological model, which is described in Section A.5.2.1.

Second, there is no use in designing a context authoring process that automates the generation of the filter rules if the translation from the contextual requirements to the model used to generate the filter rules is not automated as well. Therefore, the context authoring process must be able to interpret the contextual requirements of typical components that reside in an autonomic element, such as a rule-based reasoner, neural networks, or plain function calls. While the goal is to automate this process as much as possible, there might be specific cases that justify a (partial) manual specification of contextual requirements into the context model. For example, when a component uses a straightforward and dedicated algorithm, it might be easier to specify its contextual requirements by hand. In Section A.5.2.2, we propose an algorithm for translating the contextual requirements of a rule-based reasoner into the context portion of the model.

Third, the context authoring process must be able to cope with changes in these contextual requirements and act accordingly by altering the set of generated filter rules. The set of generated filter rules may be changed on two distinct levels: due to (1) a change in context and/or (2) a change in an external policy. The first has been illustrated in the previous section, where a change in the algorithmic details of the control loop resulted in a change in contextual requirements and ultimately in a change in the filter rules that were generated. The latter can be illustrated as follows: a network operator might constrain the context exchange process be-

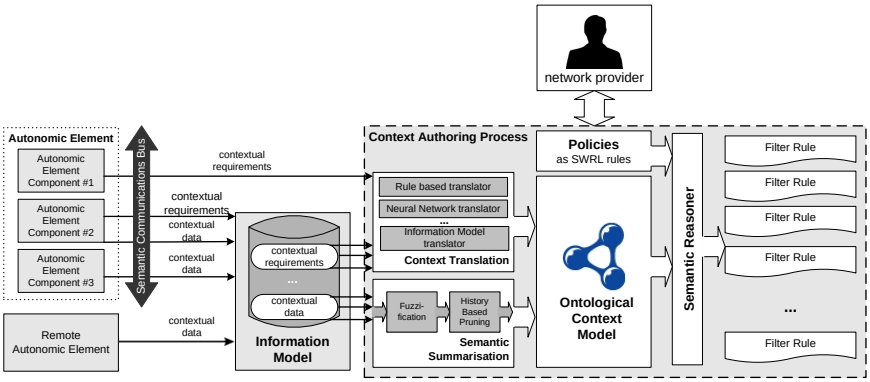


Figure A.5: Computational overview of the context authoring process.

tween two components through one or more policy rules. An example of such a policy is that the estimated overhead of exchanging context must always stay below a threshold defined by the operator. The context authoring process must then take into account this limitation in the generation of filter rules (e.g., by lowering the update frequency of a filter rule, or by other means, such as aggregating data). In both cases, the context authoring process must be able to reason over the knowledge available to adapt the filter rules. To that extent, we propose a semantic reasoner in the context authoring process in Section A.5.2.3.

Fourth and finally, the reasoning steps in the context authoring process must be able to be executed in their allotted time. More specifically, the process must react in a timely manner to a change in context so that the newly generated filter rules are constructed in time to act on context, as well as to request new context. Also, as the context authoring process is responsible for generating the filter rules for all components inside an autonomic element, it is important that the reasoning delay scales with the number of components inside the autonomic element. While exact values of this design goal depend on the employed use case, this means that the reasoning delay must be in the order of milliseconds as opposed to seconds (or greater). In Section A.5.2.4, we present a number of actions that reduce the amount of data required by the reasoning process.

A.5.2 Computational overview

An overview of the context authoring process itself is shown in Figure A.5. As illustrated, the proposed context authoring process uses an ontological approach where all required information is available in *the context model*. This ontological context model receives its information either directly from other components inside the same autonomic element or from appropriate data models derived from the

information model, as well as from applicable ontologies. Two types of data are stored: the contextual requirements of the components and a subset of the context that is available in the data model.

The contextual requirements of the components of each AE are stored into the context model through the context translation step. Depending on the type of management component, specific algorithms exist that are able to extract the contextual requirements from that specific management component and translate them into a format suitable for the context model. *The context translation* step thus contains an extensible repository for these algorithms; we present an algorithm for an illustrative management component (i.e., a rule-based reasoner) in Section A.5.2.2.

As the set of filter rules can also depend on the context, context can be stored into the context model as well. To minimise the complexity of the filter rule generation process (through semantic reasoning) the context model is kept as small as possible and only context that can trigger a change in the set of filter rules is stored in the context model. We call this reduction of context *semantic summarisation*.

Once the required information is stored into the context model, the filter rules are then generated through a semantic *context reasoning* step that reasons over the knowledge available in the context model. This reasoning process can use additional policies, in the form of SWRL rules, that can influence the filter rule generation process. These policy rules need to be manually defined by a knowledge expert (e.g., the network or service provider), and provide a simple but effective way to govern the exchange of context.

A.5.2.1 Ontological context model

Figure A.6 provides an overview of the main concepts in this context model. The basic notion of a context type is modelled in our ontology using the ContextType concept. A ContextType represents different types of context that can be requested from other components. Typical examples of context types are the memory load of a server or the packet loss of a router.

The ContextType concept also has some descriptive relationships such as the hasName relationship that provides a name for this context type (e.g., PACKET _LOSS) and other optional relationships (e.g., the hasSource and hasTarget relationships that provide information where the context originated from and what entity it describes, respectively). Through these optional relationships context can be described as broadly or narrowly as required by a specific task. For example, to describe all packet loss related context, it suffices to define a packet loss concept without a hasSource or hasTarget relationship. If a specific packet loss measurement needs reference, the hasSource and hasTarget relationships can be defined.

These context types are static and can often be directly fetched from a data model. In our prototype, we use DEN-ng as the information model [14], and instantiate a custom data model from applicable portions of the DEN-ng information

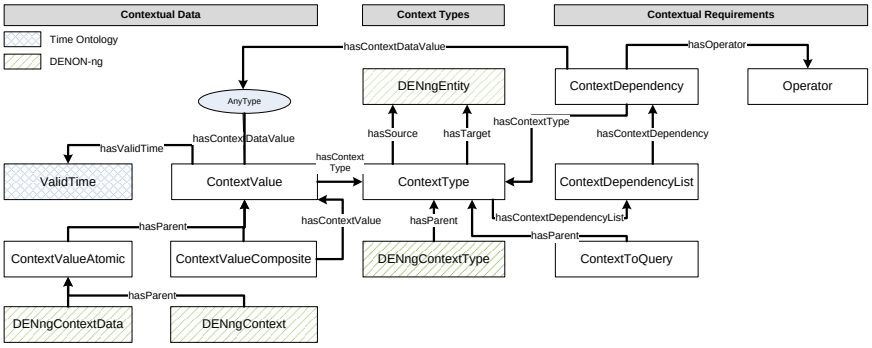


Figure A.6: Overview of the ontological context model, highlighting the main concepts and the interaction with related ontologies such as DENON-ng and the Time ontology.

model. In DEN-ng, there are a number of context classes [13] that each have attributes. The DENON-ng ontology [21], an ontology that complements the DEN-ng information model, was defined to reason about facts defined in the DEN-ng information model. Accordingly, the DENngContextType concept is defined as a sub-concept of our ContextType concept. We use the DENngContextType concept to import data from the applicable DEN-ng Context classes.

The context itself is modelled through the ContextValue concept, which has a relationship with the ContextType concept that defines the specific type that this particular instance of ContextValue belongs to. Moreover, the context model allows assigning it a particular data value (of type AnyType) and a time value at which this value occurred. For the definition of time, we use the standard Time ontology [22]. Since the context value can be a single value or a complete series of data, we use the composite pattern to model both concepts as instances of ContextValueAtomic and ContextValueComposite. Similar to the design of the ContextType concept, we leveraged classes from the DEN-ng information model. In DEN-ng, the ContextData class is used to model different aspects of context, and the Context class is used to aggregate each of the ContextData hierarchies into a single model of Context [13]. Each of the DEN-ng classes use the composite pattern to provide a rich and extensible hierarchy of modelling contextual data. Hence, we linked the Context and ContextData classes of DEN-ng to the ContextValue class of our ontology.

The exchange of a ContextType can be restricted by introducing dependencies between context types. We define a context dependency as follows: a ContextType X depends on another ContextType Y if data from X depends on values from Y. To model a context dependency, the ContextType concept has a hasContextDependencyList relationship that links a ContextType with one or more sets of context dependencies (modelled through the ContextDependency concept). The

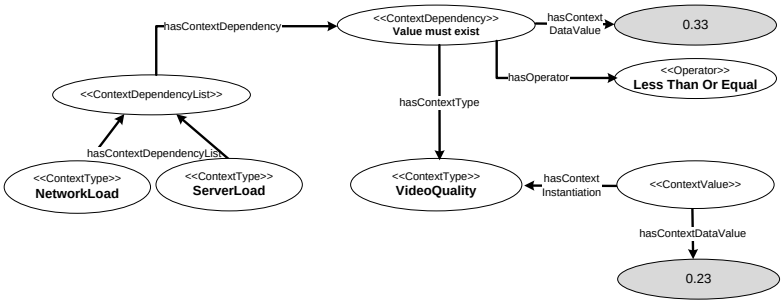


Figure A.7: An example of an instantiation of the context model for a cloud environment scenario.

context dependency has relationships with both simple data values and context types, which enables the value of a context type to be compared with a data value or another context type. A ContextType can be modelled with multiple ContextDependencyLists, meaning that it depends on multiple sets of context types.

To illustrate the use of a context dependency, Figure A.7 shows an example of how data can be stored in the context model. In this case, we use three context types: the maximum reported server load, the maximum reported network load and the average video quality of a network AE. We assume that the load types should only be requested if there is an indication of an actual problem (i.e., a drop in video quality). This problem indication is modelled through a ContextDependency that introduces a less than or equal comparison between the video quality score and a constant equal to 0.33. Hence, the loads will be requested only if the value of the video quality is lower than 0.33. As can be seen, in this example, this is indeed the case, as there is a context value instance with value 0.23. Note that not all relations of the context model are shown here for the sake of simplicity.

Instances of the ContextType concept model possible pieces of context that can be requested from other nodes in the federation. ContextType concepts with the location relationship (i.e., the hasSource relationship) can be considered as possible filter rules for a a particular router or server in the network. If the location relationship is not defined, the corresponding possible filter rule is more broad and applicable to all entities in the federated environment. Note that ContextType instances only model possible filter rules: whether they are actually generated into filter rules depends on, amongst others, the stored context values and the hasContextDependencyList relation, the definition of ContextToQuery and optional SWRL rules. The ContextToQuery concept is used for the semantic reasoning process and is described in Section A.5.2.3.

A.5.2.2 Context translation

While the context model provides an ontological basis for defining and reasoning about contextual data which in turn enables us to automatically generate the filter rules later on, it introduces the need for a process to translate the available knowledge into a technology-neutral form that facilitates decision making. In this section, we discuss how the necessary data can be translated into the context model. The context model is made up of static context information such as context types and contextual requirements (modelled through the ContextDependency concept) as well as dynamic context (modelled through the ContextValue concept).

Translating the context types The context types that are available in a management domain are static, as they only represent the type of data that can be requested from remote components. Moreover, they are typically already modelled in an information model, such as DEN-ng or a technology-specific data model, such as a vendor's Management Information Base (MIB) data. For this reason, the DENngContextType is already modelled as a sub-concept of the more general ContextType concept. By using this link, every DEN-ng context type is automatically translated to the ContextType itself through the DENON-ng ontology. This translation is simple [5]: both the attributes and associations are mapped to the corresponding ontology relationships.

Translating the context values The translation of the context values into the ontology for storage is also straightforward, as there is a direct link with its DEN-ng variants (i.e., the DEN-ng Context and ContextData classes). When new context is received, it can thus be automatically mapped to the appropriate concepts, such as a ContextValueAtomic concept. Note that, as will be discussed in Section A.5.2.4, we perform a filtering of the context beforehand so that only the required context is stored into the context model.

Translating the contextual requirements The most important challenge is translating the contextual requirements from the different components inside an autonomic element into the context model. These contextual requirements, modelled as a ContextDependency concept in the context model, depend on the actual implementation used by the autonomic element's management algorithms. As such, they require the input of a knowledge expert who is familiar with the details of the management algorithms to semantically define the contextual requirements. A complete automation of this process is therefore often not feasible. On the other hand, many management algorithms use the same technologies and - depending on the nature of the technology - a translation algorithm can be constructed that automates the translation process. This simplifies the translation process to a semi-

```
translateRuleBasedSystem(graph, rulebase) ≜
  let contextset = φ
  ∀rule ∈ rulebase :
    ∀andclause ∈ rule.conditions :
      ∀orclause ∈ andclause :
        ∀atomicclause ∈ orclause :
          ∀co ∈ getAllContextOperands(atomicclause) :
            contextset = contextset ⊔ createContextType(c,co.ContextType)
            contextset = contextset ⊔ getDependencies(c,atomicclause,graph)
  return contextset

getDependencies (c, clause, graph) ≜
  let dependencylist = φ
  let dependencyVertices = getDependencyFromClause(graph, clause)
  ∀andclause ∈ dependencyVertices :
    let dependencies = createContextRequirementList(l)
    ∀orclause ∈ andclause :
      ∀atomicclause ∈ orconditions :
        ∀co ∈ getAllContextOperands(atomicclause) :
          found = true
          dependencies = ⊔ createContextRequirement(req,co.ContextType)
          dependencies = ⊔ hasContextRequirement(l,req)
    if found ≡ true
    then
      dependencylist = dependencylist ⊔ dependencies
    endif
  return dependencylist
```

Figure A.8: The translation of the contextual requirements of a generic rule-based system to the context model.

automated step that needs supervision by the network provider. This is justified, as translating the contextual requirements is a static process and the dynamicity of changing context is handled through the semantic reasoner.

To illustrate the feasibility of this approach, we present an algorithm that automates the translation process of a management algorithm containing a rule-based reasoner (cf. Figure A.8). This algorithm is described in the Vienna Development Methodology (VDM) [23] notation, which is an implementation independent model for describing the structure and operation of software systems based on the mathematical theory of sets and maps.

The translation algorithm transforms the rules in a rulebase to contextual requirements, taking into account dependencies between rules. Applying this algorithm to other type of reasoners, such as a neural network-based reasoner, is performed in an analogous manner. The output of the algorithm is a context requirement set, which is modelled as a set of ontology axioms that define the overall context model.

We assume that all rules are rewritten in Conjunctive Normal Form (i.e., as a conjunction of disjunctions). The contextual requirements of a rule-based system

are in essence determined by the conditions of the rules in the rule set. If we take all context types described in the conditions and add them as context types, we have a basic set of contextual requirements. This is illustrated in Figure A.8, which queries context types - as part of a context operand in the rule's condition - and adds them to the context model.

We can further refine this set of contextual requirements by introducing dependencies between different context types (described in the getDependencies function). To do this, we first examine the rule base and build a condition dependency graph, which is a directed graph in which the vertices are conditions. An edge from vertex A to vertex B means that condition A is needed to trigger condition B. For example, the following three rules:

$$A \wedge B \Rightarrow C \wedge D \tag{A.1}$$

$$X \wedge Y \Rightarrow F \tag{A.2}$$

$$C \wedge F \Rightarrow G \tag{A.3}$$

result in the following condition dependency graph

$$A \wedge B \rightarrow C \wedge F \tag{A.4}$$

$$X \wedge Y \rightarrow C \wedge F \tag{A.5}$$

The construction of this dependency graph is straightforward. Since the execution of rule A.3 (with condition $C \wedge F$) can only be fired if rule A.1 and rule A.2 are fired previously, there is a dependency between the conditions of rule A.1 and rule A.3 on one hand and rule A.2 and rule A.3 on the other hand.

Based on this dependency graph, dependencies between context types are constructed by building a ContexDependencyList in the context model. First, a list of conditions is computed that define the dependencies of the given atomic clause (i.e., if the atomic clause is part of a vertex in the dependency graph). Second, the context dependency list is constructed by adding every context type that is described in the atomic clause to the list. The output of the getDependencies function is one ore more context dependency lists.

A.5.2.3 Context reasoning

The context reasoning step in the context authoring process automatically generates the appropriate filter rules. This generation is based on the knowledge available in the context model. The structure of the context model lends itself to efficiently generate filter rules since the ContextType concept and its associated relationships already model candidate filter rules. The goal of the context reasoning step is to select a specific set of context types that need to be requested. Afterwards, the translation of these types to filter rules is straightforward.

To perform the reasoning, an additional concept - called ContextToQuery - is defined in the context model. The filter rule generation process is governed based on the definition of this ContextToQuery concept. Only the context types that actually comply with the definition of ContextToQuery will be retained as filter rules. The context reasoning works thus as follows: each time the generation process is started, a semantic reasoner will perform an ontological subsumption to check which instances of the ContextType concept are also instances of the ContextToQuery concept. Hence, the ontological reasoner selects the appropriate context types from all available context types (modelled as instances of the ContextType concept) based on the given definition of ContextToQuery.

A network or service provider can provide his own definition of ContextToQuery to introduce additional policies that control the context exchange. By adjusting the ContextToQuery definition, the context exchange can thus be governed. For example, to state that only the context of one particular IP address should be requested the following straightforward definition can be used.

```
ContextType
AND hasSource
(DENngEntity AND hasAddress value "192.168.0.1")
```

Also more advanced definitions are possible: to make the filter rule generation process dependent on the context, the definition of the ContextToQuery concept should refer to the context itself. As explained in the previous section, the required context is stored in the context model and can be referenced in the definition of the ContextToQuery concept. For example, the dependencies that were introduced between context types in the translation of a rule in a rule-based reasoner can be exploited by introducing a definition that states that an instance of ContextType only belongs to the ContextToQuery concept if there is at least one ContextDependencyList of the given context types in which all ContextDependency instances have a corresponding link with a context instance. This can be expressed through the ontology language OWL2 [24]. An illustrative definition applied to the described use case is proposed in Section A.6.

Once the appropriate context types are selected as part of the ontological reasoning process, the translation of these context types to actual filter rules is straightforward. This is illustrated in Figure A.9. As shown, the filter generation process simply queries all ContextToQuery instances and translates them into a set of filter rules, bundled by their source. The generation algorithm constructs a technology-neutral list of filter rules, which does not depend on the syntax of a specific query language. Here, a FilterRule is described as a triple consisting of an identifier, source and target. The list of filter rules is then sent to every corresponding location, so that the required context can be pushed when needed.

getFilterRules$(ont) \triangleq$
  $let\ \ nodes = getNodes(ont)$
  $let\ \ rulemap = \phi$
  $\forall n \in nodes:$
    $let\ \ rules = getFilterRulesByNode(ont, n)$
    $rulemap = rulemap \sqcup (n \rightarrow rules)$
  return $rulemap$

getFilterRulesByNode $(ont, node) \triangleq$
  $let\ \ rules = \phi$
  $let\ \ types = getContextToQuery(ont)$
  $\forall i \in types$
    if $hasSource(i, node)$
    then
      $rules = rules \cup (name(i), source(i), target(i))$
  return $rules$

Figure A.9: The algorithm for automatically generating the filter rules based on the context model.

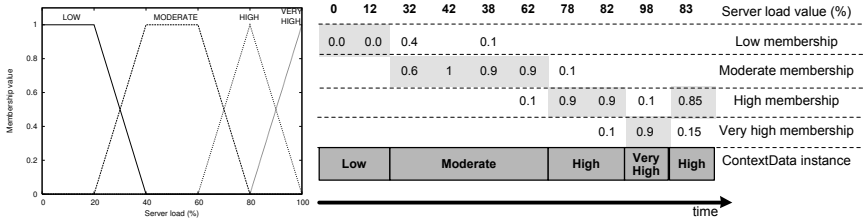
A.5.2.4 Semantic Summarisation

The context authoring process described until now stores all context in the ontological context model. While this allows all context to be queried from the definitions contained in the ContextToQuery instance and thus provides a large expressivity, this also introduces a significant performance penalty. The time needed for performing reasoning in ontologies is known to scale exponentially with the number of instances in the ontology [25]. In order to control the performance of the context reasoning, it is thus important to limit the number of instances contained in the context model as much as possible.

As the storage of context takes up the most room we try to limit the amount of context in the context model without losing the expressiveness required by the automatic filter rule generation process. We refer to this process as semantic summarisation, as we try to summarise the context that needs to be stored into the context model. The semantic summarisation process consists of two distinct algorithms: a fuzzification algorithm and a history based pruning algorithm. We explain both in the remainder of this section.

Context fuzzification: The different autonomic elements in an autonomic management domain will typically generate a continuous stream of context. This continuous stream of context can be summarised for two reasons.

First, the context authoring process often does not need detailed values of every context type to base its decision on. While detailed values may be needed in the decision function of a management control loop, the context authoring process is only used to determine which context needs to be requested from an autonomic



(a) Used fuzzy variables (b) Fuzzification process for illustrative server load values

Figure A.10: Overview of the fuzzification process for illustrative server load values. The fuzzification results in a significant reduction of the number of instances being stored in the ontological context model.

element in the management federation. The actual requested context will then be forwarded to the corresponding control loop. As such, the context authoring process can cope with much coarser values in the context.

Second, changes in the context are especially of importance, as they can potentially lead to a change in the generation of filter rules. If the autonomic elements forward context that provide no major updates in values, there is no use in creating a new ContextValue instance. However, if there is an update, then it is better to summarise the data by grouping the data values into one ContextValue instance that indicates that this value is generated over a given time period.

Figure A.10 illustrates the fuzzification process for an illustrative flow of context of the server load context type. For the server load, we defined 4 fuzzy variables, each with their corresponding membership functions: low, moderate, high and very high (Figure A.10a). Figure A.10b shows how each new value of context triggers the calculation of the 4 membership functions (membership values of 0 have been omitted for simplicity), a labelling to a fuzzy variable and a modification to the context model, either by changing an existing ContextValue instance or by creating a new one.

History based pruning: the context fuzzification will significantly decrease the number of instances of ContextValue that are created in the context model. However, without removing data from the context model, the ontology will keep growing, leading to a significant overhead. To tackle this issue, we propose a simple data pruning algorithm that only stores the context values that are not older than HWi seconds. The higher this time window HWi is, the more history can be taken into account in the context exchange process. We characterise the impact of this time window HWi value on the reasoning performance in Section A.6.

There is an important trade-off in the level of semantic summarisation that can be applied. If too much summarisation is applied, the filter rule generation process

obviously loses its flexibility and accuracy. For example, if it is important to only request context if another type of data is very high (i.e., higher than 0.95) it is useless to define membership functions that can only distinguish between values lower than 0.5 and higher than 0.5. Hence, it is important to tailor the membership functions for each context type to the appropriate scenario. This consists of assigning a linguistic term to different ranges of values, which is often a straightforward task for a knowledge expert. Also, in defining the membership functions, the knowledge expert can take a cautious approach by defining the membership functions in a sufficiently broad manner. This can lead to the exchange of context which, eventually, is not used in the management function, but it ensures that context that needs to be requested is effectively requested. Although this increases the contextual overhead, the increase can be limited depending on the scenario, as only a subset of the original context will be requested.

A.6 Performance evaluation

A.6.1 Experimental setup

We implemented a prototype of the context authoring process and integrated it with the autonomic architecture illustrated in Figure A.3. The context authoring process was implemented as an OSGi bundle in the OSGi framework [26] and was added to an already existing prototype implementation containing a semantic communications bus, which enables sending messages between components inside an autonomic element [18]. The context model was constructed using the OWL API [27]; the Pellet library [28] was used as an OWL2 reasoner. The implemented prototype allows context to be fetched from remote parties and uses this context in a management algorithm.

A.6.2 Network model

To evaluate the performance of the context authoring process, we modelled a cloud environment scenario as described in Section A.4. In this network model, a hierarchy of autonomic elements ensures an end-to-end QoE optimisation of a video service from a cloud environment to multiple clients in the home network. This end-to-end QoE optimisation can be achieved by changing configurations (e.g., reserving resources) both in the network (governed by network AEs) and in the cloud environment (governed by server AEs). We focus on the cloud infrastructure's root AE, and discuss its context exchange process. Table A.1 provides an overview of the different context types that are available in this network model, and Figure A.11 illustrates where context data are generated and how they are linked with each other. As shown, each context type, except the service information type, has both an aggregated view (i.e., an average over an AE) and a local view. The

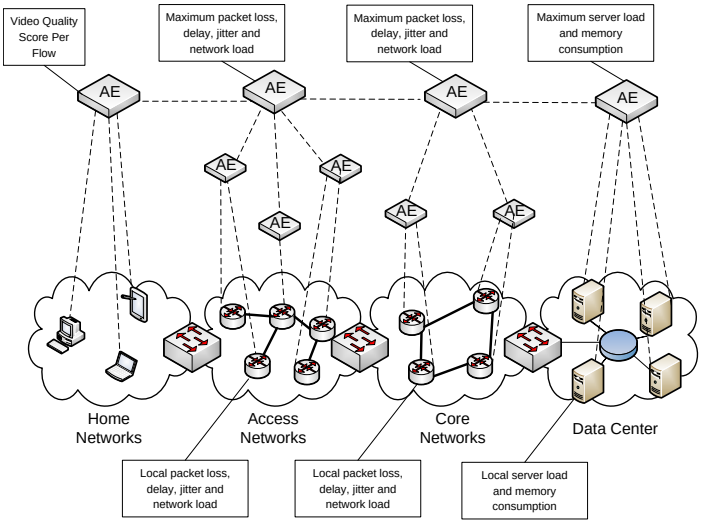


Figure A.11: Schematic overview of the use case's topology, including which types of context are generated where. Local devices such as routers and servers generate local monitoring information, whereas AEs generate derived and/or aggregate context by forwarding the maximum of local measurements.

aggregated view always consists of the maximum of the context it aggregates (e.g., the maximum reported packet loss on all routers managed by a single AE).

We assume that every context type provides an updated value of its contextual status every second. We modelled a network of 10,000 nodes assigned to 2 core networks, 1 aggregation network, 1 access network and 100 home networks. Based on this network size, in total, 150,000 possible context types are continuously generated. This thus leads to a huge amount of context that is being requested if the context communication is not carefully controlled. By using the context authoring process, only a subset of this context will actually be requested. As discussed in Section A.4, in this AE, we used a rule-based reasoner as a management algorithm that checks the status of the aggregated context types of its neighbouring network AEs and children server AEs and requires local, and thus more detailed, context types, if one of the aggregated context types exceed a threshold stated in the rules (e.g., to zoom further in on the problem).

We will now discuss how the context authoring process can be applied to support the scenario depicted in Section A.4. We applied the context translation algorithm described in Figure A.8 to the management algorithm's rule base. The generation algorithm used four different configuration sets. In the first configuration set (OWL), the ContextToQuery concept was defined as OWL restrictions and no semantic summarisation occurred. In the second configuration set (Summarised OWL), OWL restrictions were still used but the semantic summarisation

Table A.1: Overview of the available context types and their granularity. The aggregated context types denote the maximum value reported by the corresponding local context types that are controlled by the AE.

Name	Aggregated	Local
Service information	N/A	1 per Flow
Server CPU load	1 per Server AE	1 per Server
Server memory consumption	1 per Server AE	1 per Server
Video quality score	1 per Network AE	1 per Flow
Network load	1 per Network AE	1 per Router
Packet loss	1 per Network AE	1 per Router
Jitter	1 per Network AE	1 per Router
Delay	1 per Network AE	1 per Router

step was also enabled. The third and fourth configuration sets (SWRL and Summarised SWRL) again disabled and enabled the semantic summarisation process, but replaced the ContextToQuery definitions with SWRL rules.

As output of the context translation algorithm, the rules that require more local context types when the aggregated context types exceeded a threshold, resulted in multiple context dependencies (modelled through the ContextDependency concept) being stored in the context model. For example, a rule that required the CPU load per server when the maximum reported server load is higher than 0.9, resulted in a context dependency between the MAX_CPU_LOAD and CPU_LOAD context types as also previously illustrated in Figure A.7. This particular context dependency resulted in the following (partial) definition of the ContextToQuery concept when using the Summarised OWL approach.

```
ContextType AND hasName value "CPU_LOAD"
AND hasContextDependency
   (ContextDependency
       AND hasContextType (ContextType
            AND hasName value "MAX_CPU_LOAD")
       AND hasContextValue
          (LatestContextValue
              AND hasContextDataValue
  value "VERY_HIGH"))
```

In the case of a configuration with SWRL rules, the following SWRL rule was generated:

```
ContextType(?t) AND hasName(?a,"CPU_LOAD")
```

```
   AND hasContextDependency(?t,?d)
   AND ContextDependency(?d)
   AND hasContextType(?d,?t2)
   AND ContextType(?t2)
   AND hasName(?a2,"MAX_CPU_LOAD")
   AND hasContextValue(?t2,?v)
   AND LatestContextValue(?v)
   AND hasContextDataValue(?v,"VERY_HIGH")
==> ContextToQuery(?t)
```

What these definitions in essence say is that, if the latest aggregated maximum CPU load value has been classified as very high by the fuzzification algorithm, the server specific CPU load context types should be requested as well. In this case, the latest CPU load value is defined as an instance of the LatestContextValue concept, which is a sub-concept of ContextValue in the context model, and defined through a separate OWL restriction or SWRL rule that performs a timestamp comparison with the current time. Note that these examples are used when the semantic summarisation step is enabled. Without semantic summarisation, the definitions are slightly changed to incorporate the crisp value as threshold (i.e., 0.9).

Similar OWL restrictions and SWRL rules were used for the other context types. In this network model, the behaviour is thus as follows: we start out with filter rules that only request aggregated context types. Depending on the generated context, other context types can be requested for a period of time, again depending on the subsequent generated context. To evaluate the scalability of the approach, we increased the number of AEs that are involved in the context authoring process. This resulted in a variation on the number of context types that are initially requested from 1 to 250, the corresponding context values that need to be stored into the context model thus also increase with this setting.

A.6.3 Evaluation result details

As ontological approaches are known to have scaling issues in terms of reasoning time we use this reasoning time as performance metric. The time needed for the other steps in the generation process, being the time needed to store any new context into the context model and instantiate the generated filter rules on the remote parties, after they were generated, are negligible.

We investigated the impact of the different configuration sets (A.6.3.2) and configuration of the semantic summarisation step (A.6.3.3) with the goal of deriving guidelines on the necessary amount of summarisation. Furthermore, to investigate the scalability of the approach we characterised the reasoning time for an increasing amount of context types involved, triggered by an increase of the number of server and network AEs in the network model (A.6.3.4). Each experiment

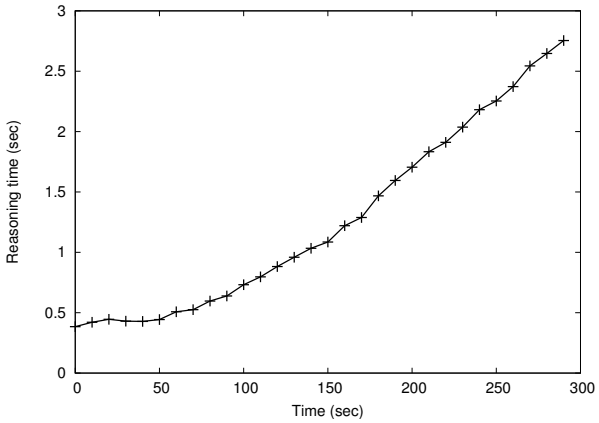


Figure A.12: Influence of OWL reasoning on the reasoning time as a function of time.

was repeated 20 times: thus we present average values together with their standard deviation. The experiments were carried out on a Dual-Core 2Ghz AMD machine with 3.5 GB of RAM memory.

A.6.3.1 Reasoning time without semantic summarisation

In this experiment, we investigate the impact of the standard ontological approach, using OWL reasoning and without any semantic summarisation. We emulated the scenario described in Section A.4, with a network topology containing 6 network AEs and 10 children server AEs. Thus, this means that in total 50 context types continuously generate context and send it to the cloud's AE. Figure A.12 illustrates the reasoning time as a function over time as more and more context is being stored into the context model. As shown, as time elapses, the reasoning time quickly increases. As more context is being stored in the context model, the reasoning time increases in the order of seconds. After 5 minutes of operation, it takes 2.82 seconds to generate the appropriate filter rules, while at startup only 400 milliseconds were needed. Furthermore, the reasoning time follows an exponential increase, which renders this approach infeasible for any real-time operation. This significant increase in the reasoning time of the OWL reasoning configuration set is due to the poor scalability of ontological reasoning as the number of instances in an ontology increases. After 5 minutes more than 15,000 context values have been added to the ontology, as 50 context types generate an updated context value every second. This results in an explosion of the size of the context model and consequently of an explosion of the reasoning time.

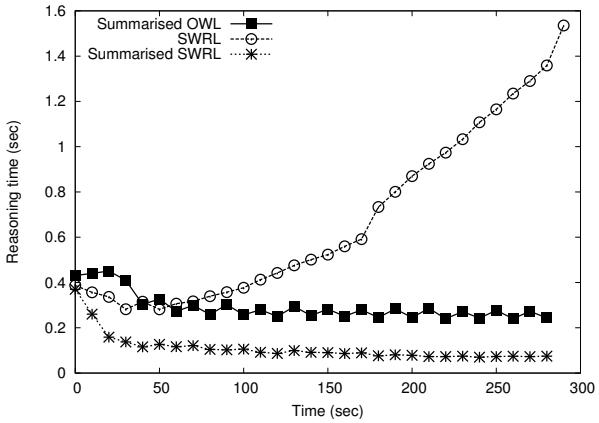


Figure A.13: Influence of the Summarised OWL, SWRL and Summarised SWRL reasoner configuration sets on the reasoning time as a function of the time.

A.6.3.2 Influence of applying semantic summarisation and SWRL inferencing

The impact of applying semantic summarisation on one hand and introducing SWRL inferencing on the other hand is illustrated in Figure A.13. Here, the reasoning time is shown for 3 different configuration sets: Summarised OWL, SWRL and Summarised SWRL. For the summarisation algorithms, we defined 5 fuzzy variables per context type in the fuzzification algorithm, and chose to keep the latest 30 seconds of context in the history-based pruning algorithm. Similar to the previous experiment, initially 50 context types generate context as the context exchange is performed with 6 network AEs and 10 server AEs. As can be seen, there is a significant difference between the standard OWL reasoning without summarisation, discussed in the previous section, and these 3 configuration sets.

A number of important observations can be made from these results. First, the summarisation process clearly improves the performance of the context authoring process, regardless if OWL or SWRL-based reasoning is used. Where the OWL reasoning without semantic summarisation introduced reasoning times of 3 seconds after 5 minutes, enabling the semantic summarisation process keeps the obtained reasoning times around 240 milliseconds. A similar behaviour can be observed when enabling semantic summarisation for the SWRL-based reasoning. As the semantic summarisation process limits the context that is being stored into the context model, we are able to decrease the required reasoning time considerably to reasoning times of only 74 miliseconds.

Second, enabling the summarisation also stabilises the obtained reasoning time. This is clearly visible in Figure A.13 where the obtained reasoning times do not

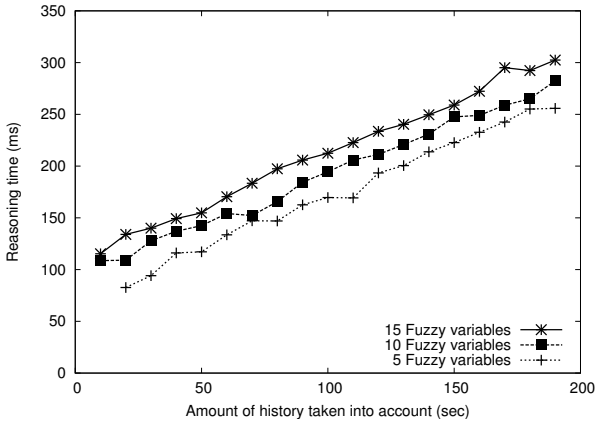


Figure A.14: Influence of an increasing time window HWi in the history based pruning algorithm, defining the history that is stored in the context model, on the reasoning time. Different fuzzification configurations are investigated.

increase for the Summarised OWL and Summarised SWRL configuration sets as the time increases. Even better: the reasoning time slightly decreases in the first 30 seconds because of the reduced initial start-up overhead of the reasoner. The standard SWRL reasoning on the other hand does result in an increasing reasoning time as the time elapses: over the course of 5 minutes the reasoning time of the SWRL configuration set increases again exponentially from 400 ms to 1580 ms and results in reasoning times that are even higher than that of the Summarised OWL configuration set, even after 50 seconds. As the summarisation process limits the ontology's size through fuzzification and pruning, the obtained reasoning times are stabilised. This is certainly an important aspect. In order to be feasible in an on-line deployment, the performance of the generation process should remain stable as a function of the time. As illustrated in Figure A.13, only the semantic summarisation approaches feature this behaviour, as they limit the ontology size by removing old and irrelevant data.

Third, regardless of the summarisation process, the use of SWRL inferencing as definition of the ContextToQuery concepts also improves the performance of the context authoring process. SWRL inferencing is typically known to have a better reasoning performance because the reasoning itself does not assume an open-world, which is the case for standard OWL reasoning. Therefore, the combination of both approaches, the summarisation algorithm and SWRL inferencing, yields the best results. It does not only have the lowest reasoning times (approximately 74 msec), it is also able to stabilise the obtained reasoning times as more context is being generated. Furthermore, reasoning times of 74 msec are certainly small enough to be deployable in an on-line management environment.

A.6.3.3 Configuration of the semantic summarisation algorithm

The previous results showed the advantage of summarising the context before it is stored in the context model. In this section, we investigate the impact of the number of fuzzy variables in the fuzzification algorithm and the time window HWi of the history based pruning algorithm that defines the amount of history that is stored in the context model. Both parameters influence the ontology's size, and also the reasoning time. Similar to the previous section, 50 context types were initially requested in this experiment.

Figure A.14 illustrates the reasoning time as a function of this HWi parameter for various configurations of the fuzzification algorithm. These various configurations are differentiated by a different number of fuzzy variables that are defined. As can be expected, increasing the time window leads to an increase in overall reasoning time: as more history is being stored into the ontology and not removed by the pruning algorithm, the ontology increases, as does the reasoning time. However, the increase in reasoning time is still linear because of the fuzzification algorithm. As the fuzzification algorithm groups context with the same value, increasing the history that is stored for one particular context type does not always lead to an additional context value instance being stored into the context model. This allows us to avoid a typical exponential increase in scalability of the ontology's performance. Instead, increasing the amount of history that is taken into account results in a linear increase from 70-120ms, if 5 seconds of history is taken into account, up to 250-300 ms, if 3 minutes of history is taken into account.

As shown, the configuration of the fuzzification algorithm has a small but clear influence on the reasoning time. As more fuzzy variables are defined, potentially more context value instances are added to the ontology, especially if the context features a strong oscillation in one or more data values. Consequently, the higher ontology size results in a higher reasoning time. However, the experienced increase in reasoning time is rather small: increasing the number of fuzzy variables from 5 to 15 only leads to a 40 ms increase in reasoning time. As the use of 15 fuzzy variables allows differentiating between 15 levels for a particular context value, this already offers a considerable amount of expressivity.

A.6.3.4 Scalability evaluation results

Although the context exchange process of the emulated scenario already consisted of a federation of moderate size (i.e., 16 participating AEs), we investigate the scalability of the context authoring approach by increasing the number of AEs that are involved in the context exchange process up to 75 AEs. As these AEs represent autonomic elements that each manage a subset of the network, that many AEs involved in the context exchange corresponds to a network of hundreds of thousands of nodes. For this experiment, 5 fuzzy variables were defined.

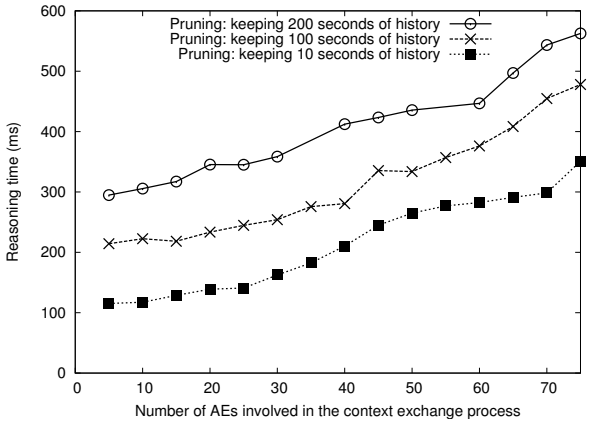


Figure A.15: Impact of an increasing number of AEs on the reasoning time. A context exchange with 75 AEs, which represents a network topology of hundreds of thousands of nodes, results in a reasoning time of 478 msec.

Figure A.15 illustrates the obtained reasoning time for an increasing number of AEs involved in the context exchange. Also, different configurations of the history based pruning algorithm are illustrated. Although increasing the number of AEs does lead to an increase in reasoning time, the experienced increase is still tolerable. If the number of AEs involves is increased from 5 to 75, the reasoning time approximately doubles (e.g., if the last 100 seconds of context is kept the reasoning time increases from 210 msec to 478 msec).

In general, the obtained reasoning times show that the context authoring process scales well as a function of the size of the federation. For context exchange processes of moderate size, it is possible to keep the reasoning times in the order of 200 msec and less and still store a considerable amount of history in the context model. As both the history and the federation's size are increased, reasoning times in the order of 500 msec are obtained. While these latter values correspond to a considerable increase in reasoning time, it is safe to assume that a management federation of 75 AEs will perform complex management tasks, whose execution is in the order of seconds, so that high reasoning times for the context exchange process are allowed.

A.7 Conclusions

In this chapter, we proposed a context authoring process for autonomic federated management nodes. The context authoring process provides an automated context exchange process through an ontological approach where knowledge about the contextual requirements is stored in an ontological context model. Based on this

context model, the appropriate filter rules, which define the context that needs to be requested from remote nodes, can be automatically generated.

The context authoring process allows a service provider to delegate the contextual requirements of a management algorithm to the context manager component, which is responsible for automating the context exchange process. The context authoring process itself features a number of contributions: (i) it contains translation algorithms that allow typical management algorithms, such as a rule-based reasoner, to automatically store their contextual requirements into the ontological model; (ii) through semantic reasoning, the context exchange process is dependent on the contextual data itself, which means that the set of generated filter rules can be automatically adapted when the context changes; (iii) the context authoring process can be significantly improved by including a fuzzification and pruning algorithm which summarise the semantic information that is stored in the ontology and significantly decrease the time needed to generate the filter rules.

A performance evaluation of a prototype of the context authoring process showed that a combination of SWRL inferencing as semantic reasoning method and the enabling of the proposed semantic summarisation algorithms provide the best results. Furthermore, the results showed that the semantic summarisation algorithms successfully limited the ontology size as more data was being stored. The obtained reasoning times are in the order of tens of milliseconds for the considered federation sizes. For example, a federation with 16 AEs involved in the context exchange process are able to generate the filter rules in approximately 72 msec. As such, the results show that the context authoring process is able to react in a timely manner to context changes, which makes it deployable in on-line federated management environments.

References

[1] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. *A survey of autonomic communications*. ACM Transactions on Autonomous and Adaptive Systems, 1(2):223–259, 2006. doi:10.1145/1186778.1186782.

[2] M. Murphy, L. Abraham, M. Fenn, and S. Goasguen. *Autonomic clouds on the grid*. Journal of Grid Computing, 8(1):1–18, 2010. doi:10.1007/s10723-009-9142-3.

[3] J. Strassner, N. Agoumine, and E. Lehtihet. *FOCALE – a novel autonomic networking architecture*. International Transactions on Systems Science and Applications, 3(1):64–79, 2007.

[4] L. Baresi, A. D. Ferdinando, A. Manzalini, and F. Zambonelli. *The CASCADAS framework for autonomic communications*. In Autonomic Communication, chapter 6, pages 147–168. Springer, 2009. doi:10.1007/978-0-387-09753-4_6.

[5] S. Latré, S. van der Meer, F. De Turck, J. Strassner, and J. Won-Ki Hong. *Ontological generation of filter rules for context exchange in autonomic multimedia networks*. In 12th IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 575–582, 2010. doi:10.1109/NOMS.2010.5488448.

[6] J. Strassner. *Policy-based network management: Solutions for the next generation*. Morgan Kaufmann Publishers Inc., 2003.

[7] T. Gu, X. Wang, H. Pung, and D. Zhang. *An ontology-based context model in intelligent environments*. In Communication Networks and Distributed Systems Modeling and Simulation Conference, 2004.

[8] M. Khedr and A. Karmouch. *Negotiating context information in context-aware systems*. IEEE Intelligent Systems, 19(6):21–29, 2004. doi:10.1109/MIS.2004.70.

[9] A. Bellandi, B. Furletti, V. Grossi, and A. Romei. *Ontology-driven association rule extraction: A case study*. In Context & Ontologies: Representation and Reasoning, 2007.

[10] R. Neisse, P. Costa, M. Wegdam, and M. van Sinderen. *An information model and architecture for context-aware management domains*. In IEEE Workshop on Policies for Distributed Systems and Networks (POLICY), pages 162–169, 2008. doi:10.1109/POLICY.2008.31.

[11] J. Strassner, J. de Souza, S. van der Meer, S. Davy, K. Barrett, D. Raymer, and S. Samudrala. *The design of a new policy model to support ontology-driven reasoning for autonomic networking*. Journal of Network and Systems Management, 17(1):5–32, 2009. doi:10.1007/s10922-009-9119-3.

[12] G. Castelli, M. Mamei, and F. Zambonelli. *Engineering contextual knowledge for autonomic pervasive services*. Information and Software Technology, 50(1):36–50, 2008. doi:10.1016/j.infsof.2007.10.009.

[13] J. Strassner, J. de Souza, D. Raymer, S. Samudrala, S. Davy, and K. Barrett. *The design of a novel context-aware policy model to support machine-based learning and reasoning*. Cluster Computing, 12(1):17–43, 2009. doi:10.1007/s10586-008-0069-4.

[14] E. Power, Z. Boudjemil, and S. Fox. *Architecture and implementation of a testbeds repository*. In International Conference on Telecommunications and Multimedia (TEMU), 2010.

[15] D. Lewis, D. O'Sullivan, K. Feeney, J. Keeney, and R. Power. *Ontology-based engineering for self-managing communications*. In 1st IEEE International Workshop on Modelling Autonomic Communications Environments (MACE), pages 81–100, 2006.

[16] J. Keeney, D. Jones, D. Roblek, D. Lewis, and D. O'Sullivan. *Knowledge-based semantic clustering*. In ACM Symposium on Applied Computing (SAC), pages 460–467, 2008. doi:10.1145/1363686.1363801.

[17] J. Keeney, D. Roblek, D. Jones, D. Lewis, and D. O'Sullivan. *Extending siena to support more expressive and flexible subscriptions*. In Second International Conference on Distributed Event-Based Systems (DEBS), pages 35–46, 2008. doi:10.1145/1385989.1385995.

[18] J. Famaey, S. Latré, J. Strassner, and F. De Turck. *An ontology-driven semantic bus for autonomic communication elements*. In Proceedings of the 5th IEEE international conference on Modelling Autonomic Communication Environments (MACE), pages 37–50, 2010. doi:10.1007/978-3-642-16836-9_4.

[19] J. Famaey, S. Latré, J. Strassner, and F. De Turck. *A hierarchical approach to autonomic network management*. In 2nd IFIP/IEEE International Workshop on Management of the Future Internet (ManFI), pages 225–232, 2010. doi:10.1109/NOMSW.2010.5486571.

[20] I. Amazon.com. *Netflix selects amazon web services to power mission-critical technology infrastructure*. News Release –

http://www.businesswire.com/news/home/20100507005180/en/Netflix-Selects-Amazon-Web-Services-Power-Mission-Critical, last accessed at 2011/01/04.

[21] M. Serrano, J. Serrat, J. Strassner, and M. O Foghlu. *Management and context integration based on ontologies, behind the interoperability in autonomic communications*. In SIWN International Conference on Complex Open Distributed System (CODS), pages 435–442, 2007.

[22] J. R. Hobbs and F. Pan. *An ontology of time for the semantic web*. ACM Transactions on Asian Language Information Processing, 3(1):66–85, 2004. doi:10.1145/1017068.1017073.

[23] D. Bjørner and C. B. Jones, editors. *The vienna development method: The meta-language*. Springer-Verslag, 1978.

[24] P. Hitzler, M. Krtzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph. *OWL 2 web ontology language primer*. http://www.w3.org/TR/owl2-primer, 2009.

[25] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. *Towards a complete OWL ontology benchmark*. In The Semantic Web: Research and Applications, Lecture Notes in Computer Science, pages 125–139. Springer Berlin / Heidelberg, 2006. doi:10.1007/11762256_12.

[26] OSGi Alliance. *OSGi service platform release 4 version 4.2 core specification*. http://www.osgi.org/Download/File?url=/download/r4v42/r4.core.pdf, 2009.

[27] M. Horridge and S. Bechhofer. *The OWL API: A Java API for working with OWL 2 ontologies*. In OWLED, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[28] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. *Pellet: A practical OWL-DL reasoner*. Web Semantics: Science, Services and Agents on the World Wide Web, 5(2):51–53, 2007. doi:10.1016/j.websem.2007.03.004.

B

Design and evaluation of a hierarchical application placement algorithm in large scale clouds

H. Moens, J. Famaey, S. Latré, B. Dhoedt, and F. De Turck

Published in the proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management

As the requirements and scale of cloud environments increase, scalable management of the cloud is needed. Centralized solutions lack scalability and fully distributed management systems only have a limited overview of the system. One of the often-studied problems in cloud environments is the application placement problem, used to decide where application instances are instantiated and how many resources to allocate to the instances. In this appendix a general approach is introduced for using centralized cloud resource management algorithms in a hierarchical context, increasing the scalability of the management system while maintaining a high placement quality. The proposed method uses aggregation techniques to generate input values for the hierarchical application placement algorithm. Subsequently, performance of hierarchical application placement method is compared to that of a fully centralized algorithm. Results show that the hierarchical algorithm's solution is only 5% worse than that of the centralized algorithm, but takes 25% less time to calculate and scales significantly better.

B.1 Introduction

In recent years the adoption of cloud computing has increased greatly. The increasing scale of clouds complicates management, which leads to to scalability issues of the management system itself, compromising the scalability of hosted applications. Centralized management systems are being replaced by distributed management infrastructures, that are often fully decentralized, lacking a full overview of the system, and making it more difficult to achieve a global optimum.

Our previous theoretical work [1] indicates that a hierarchical structuring of control nodes enables good sharing of context, information of the current state of the system, with a relatively small communications cost, while at the same time having a high scalability.

In this appendix, we investigate how centralized algorithms can be modified to work in large-scale environments by using a hierarchical management system. The different levels of the hierarchy have a different view of the system, with highest-level controllers having an overview of the system based on aggregated values, increasing the scalability of a hierarchy compared to a purely centralized solution. To this end, we evaluate how centralized solutions for one specific problem, the cloud application placement problem, can be incorporated into a hierarchical management system.

One of the key challenges in cloud management is quickly adjusting application resource allocation in the face of changing demands, and doing this in a scalable way. Determining which servers in the cloud need to execute which applications is done by means of solving the application placement problem. This problem is NP-hard, and many different solutions have been proposed [2–5].

Currently the two common approaches to application placement are centralized and fully decentralized solutions. In centralized solutions, a controller gathers monitoring information, calculates an optimal placement, and enforces the configuration. These algorithms tend to be highly complex and slow to execute, which makes scalability an issue. Decentralized approaches on the other hand optimize using only local information, leading to suboptimal placement.

Our results demonstrate that a hierarchical approach leads to scalable and fast cloud application placement, as the structure scales better than the centralized approach, and has a higher-level overview of the total system compared to the fully decentralized model. A generic template for using centralized application placement algorithms in a hierarchical fashion is presented, based on information aggregation and decoupling of management levels. The aggregation and decoupling techniques demonstrated in this appendix will enable the use of various centralized algorithms in a hierarchical fashion, greatly increasing scalability of existing cloud management solutions.

In the next section, we will discuss related work. Afterwards, in Section B.3

we will give an overview of the system architecture. Section B.4 contains a generalized formal description of the application placement problem. Following this, we discuss the hierarchical management itself in Section B.5, where both the creation of a management hierarchy and the modification of centralized algorithms to function in the system are described in-depth. In Section B.6 we evaluate the proposed solution. Finally, Section B.7 contains our conclusions.

B.2 Related work

Much work has been done concerning the application placement problem. Most of the work can be divided into two categories: centralized and fully distributed.

One of the first papers on the subject of application placement was published in 2003 [5]. A solution is generated centrally using linear programming and genetic algorithms. The solution is more geared towards consolidation of datacenter resources rather than dynamic cloud provisioning in large scale clouds, as is the case here. Our solution is designed to scale where centralized solutions no longer function, be it because of CPU limitations, which would be the bottleneck when using linear programming, or bandwidth bottlenecks, which would occur when using genetic algorithms.

Multiple centralized solutions have been proposed [2, 6–8], and many of the algorithms use similar principles. The centralized solution proposed in [2] uses multiple min-cost max-flow problems to generate a suitable solution and has a complexity of $O(n^{2.5})$, better than older solutions. The algorithm was further expanded in [8], yielding slightly better performance and results. As centralized solutions have access to all information concerning managed nodes, the placement quality is generally very good. These solutions work well for smaller datacenters, but do not scale well for large datacenters. Our solution uses centralized algorithms on clusters which are limited in size, leading to a much higher scalability at a cost to placement quality due to the smaller overview of the system resources.

A distributed peer to peer system, used for cloud management was demonstrated in [9]. In this solution, every node contains a database of management information, which is selectively flooded towards other nodes. This ensures every node has the relevant information for its management, but due to its design, no single node has a full overview of the network. In our work, we focus specifically on a subproblem, the application placement problem, and we examine how application placement can be made to scale while maintaining a global overview.

In [3] a fully decentralized approach is used, based on a gossip-protocol. Here individual nodes manage themselves and continually exchange information. Nodes continually improve their configuration by exchanging information and shifting load between them. It is shown that this leads to an optimal configuration if memory constraints are omitted. This decentralized approach has a very high scalabil-

ity, but convergence to an optimum is slower when compared to the centralized approach, and as each step leading to a configuration causes the migration of applications, these steps are expensive. Another decentralized approach, proposed in [4] uses an economical approach in which every actor tries to maximize its own gain. In doing so, a good global solution is obtained, but like all fully decentralized solutions there is no higher-level overview of the network. In contrast to these fully decentralized solutions, ours executes application placement on different hierarchy levels in which higher-level nodes have better overview of the system and are able to achieve good placement quality, while still maintaining good scalability.

While most application placement approaches are based on CPU and memory requirements, [10] executes application placement based on the physical location and bandwidth requirements of the servers, trying to put as many application components as possible close to each other and taking into account the physical system configuration. The solution is based on a central placement manager of which multiple, synchronized instances can exist. It combines properties of both the centralized approach and decentralized approaches, but needs much synchronization between management node instances. In this appendix we propose a management system in which the different management nodes are more loosely coupled, needing only limited amounts of communication between nodes.

Hierarchical techniques in provisioning were used in [11], but this system is dependent upon hierarchical application and system descriptions and executes placement using only bandwidth information. By contrast, our solution uses hierarchies for the management itself without needing additional descriptions.

Automatic hierarchical node ordering in peer to peer systems was demonstrated in [12]. TreeP is used to hierarchically structure nodes in peer to peer systems, mainly used to structure object lookup, and its structure is inspired by that of B-Trees. Each node can occur at multiple levels in the tree. Our solution uses a similar approach, but has different goals and thus uses a different structure. We use dedicated management nodes, without strict ordering. Because of this our solution can grow and recover faster, as no node ordering is needed in our management system.

B.3 System architecture

The cloud computing datacenter consists of multiple servers. All the servers have two applications installed on them: a cloud environment, such as OpenNebula, and management middleware, the functionality of which will be explained later in this section. Most of the servers are *execution servers*, used to execute application instances. Some servers are used by the management system to control the execution servers. These servers, used to manage the cloud system are *management servers*. They execute a dedicated management application in their cloud envi-

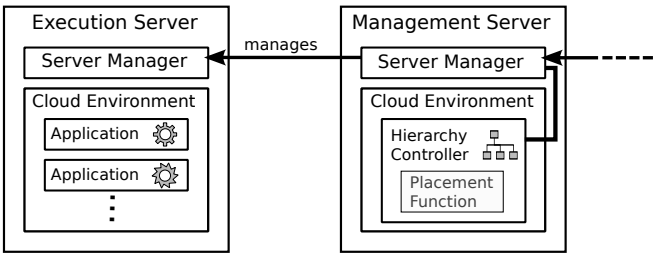


Figure B.1: The system components on management and execution servers

ronment instead of regular applications. This approach allows every server to be either a management server or an execution server, enabling dynamic scaling of the management system itself. The servers of the datacenter are connected hierarchically, where the execution servers act as leaves and the management servers are nodes of the management tree.

The different system components are shown in Figure B.1. Each server in the system contains a *server manager*. This is a lightweight middleware component, responsible for maintaining a relationship with the servers parent, gathering management information and sending it to the parent management server. Its main function is abstracting the cloud infrastructure present on the server. As this component is present on all servers, it can also be used to increase robustness of the system by monitoring its parent, and initiating leader election should one of the management nodes fail. A cache of neighbours, nodes with which the server has communicated with recently, can be used to reconnect to the management system.

A second component, the *hierarchy controller* is only present on management servers and gathers service performance information of their children. These children are either application servers or other controllers. The controller executes application placement based on the locally present management information. It also aggregates it and forwards it to its own parent. The parent node executes application placement on a higher level, determining which clusters execute which applications, and how much of each application to execute.

This hierarchy controller, unlike the server manager is a heavyweight component, which is executed like the other applications in the cloud (e.g.: in the case of OpenNebula as a Virtual Machine (VM)). It is responsible for executing the application placement algorithm. As every server in the cloud can be used as a controller, the management system itself can scale when needed and can be made more reliable.

The server manager and hierarchy controller form the backbone of the architecture, but to enable full cloud functionality, additional supporting components would be needed: An *application request router*, responsible for routing successive application requests to application instances and session management, an *ap-*

Table B.1: Symbols

Symbol	Description
A	The set of all applications.
S	The set of all servers.
Γ	The various resources considered by the system.
Ra	The resource availability of the various servers. Ra_s^r determines the available amounts of resource r on server s.
Rd	The resource demand of the applications. Rd_a^r is a value containing the demand of application a for resource r.
M	The placement matrix, $M_{s,a}^r$ contains the amount of resource r allocated to server s for application a.
Γ_s	Resources for which the demand is strict. They have a fixed demand per-instance and without this amount the placement is invalid.
Γ_l	Resources for which the demand is loose. The goal of the management system is to maximise loose demand fulfilment.
Rd_l	Resource demands for loose resource types.
Rd_s	Resource demands for strict resource types.

plication image repository containing images of the various hosted applications, and a *policy repository* containing placement restrictions. In this appendix we focus on the application placement problem itself, so these components were not implemented.

B.4 Formal problem description

The application placement problem itself has previously been described formally [2–4, 9]. In this section we formally describe and generalize common inputs and outputs of centralized application placement algorithms [2, 8, 10]. An overview of the symbols used here is shown in Table B.1.

A cloud consists of a set of servers S on which a set of applications A are executed. The cloud management system considers a set of multiple resources types Γ, such as memory, CPU and bandwith. For each resource type $r \in \Gamma$, every server $s \in S$ has available resources Ra_s^r, and every application $a \in A$ has a resource demand Rd_a^r. A centralized application placement algorithm takes a set of inputs and delivers as output a placement matrix M. For application a, server s and resource r, $M_{s,a}^r$ contains the amount resource r to allocate on server s for application a. The inputs generally contain server resource information Ra, the current placement M', and application resource demands Rd. When it comes to resource demands, we differentiate between two resource types which we shall call *strict* and *loose*. The set of strict resources $\Gamma_s \subset \Gamma$ contains demands that are invariable and per-instance, such as memory use, and in some instances bandwidth. Loose demands $\Gamma_l \subset \Gamma$, such as CPU requirements and sometimes bandwidth, are total

demands. The goal of the application placement problem is to optimize the fulfilment of loose requirements, while respecting the strict application requirements. Generally, strict requirements of applications are considered fixed for every application whereas loose requirements are variable [7]. Strict resource demands are indicated by Rd_s, loose resource demands by Rd_l. The complete specification of the application placement function is:

$$applace : Ra \times Rd_s \times Rd_l \times M' \rightarrow M$$

B.5 Hierarchical management

In this section we will describe the hierarchical management structure. First we will describe the management hierarchy itself, after which we will explain how the centralized algorithm is used in individual management nodes.

B.5.1 Hierarchical management structure

A hierarchical management node organisation is dynamically created by executing "add node" operations for each of the servers to add them to an existing management hierarchy. As more nodes are added, the hierarchy will automatically restructure itself. The structure of the management hierarchy is inspired by that of B-Trees [13]. B-Trees are datastructures, used mainly for ordering large amounts of data. The most important characteristics of B-Trees are the large number of children for every node and an equal depth of tree leaf nodes. Every node in a B-tree, except for the root, contains between n and $n/2$ entries. The root itself is allowed to have any number of nodes between 0 and n.

In the hierarchical management scenario, less restrictions are needed as the hierarchy is only used to structure nodes, and not to order them. Adding nodes can be realised simply by adding them to any controller at the lowest level. Deleting execution nodes can be done trivially, whereas deleting management nodes is achieved by performing a leader election amongst its children. Furthermore, we change the restrictions at the root node: an imbalanced tree is allowed, but only at this level. This enables us to require a higher minimum node count in the root than in a regular B-Tree, as using a dedicated server to manage only a small number of child servers would be a waste of resources.

As controllers gain more and more children, the execution time of $applace$ continually increases. Eventually the execution time exceeds a given threshold, indicating that the node is overutilized. In the reverse case, if execution time gets too low, network delay overhead becomes a bigger concern than $applace$ execution time, which indicates the node is underutilized. By executing $applace$ for various server counts the number of children causing overutilization and underutilization can be determined. These values are C_{min} and C_{max}. If $C_{min} \leq \frac{C_{max}}{2}$, splitting

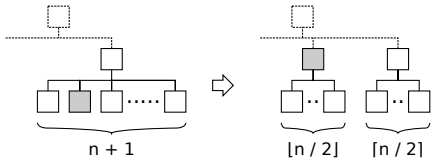


Figure B.2: Solving overutilization by splitting a node and promoting a child node (grey) to peer status.

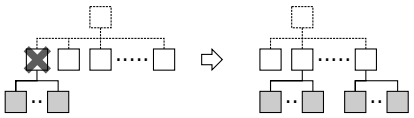


Figure B.3: Solving underutilization by removing a node and distributing its children (grey) amongst the node's peers.

and merging of nodes can be achieved using basic B-Tree operations. If $C_{min} > \frac{C_{max}}{2}$ techniques used in variations on B-Trees can be used where n nodes are considered and split into $n + 1$ nodes (as opposed to splitting a single node into two nodes in regular B-Trees).

The splitting of overutilized nodes is illustrated in Figure B.2. A node n_i chooses a node $n_j \in children(n_i)$, which it adds to $children(parent(n_i))$ it then chooses half the nodes remaining in $children(n_i)$, which it adds as children to n_j. The opposite situation is demonstrated in Figure B.3, where an underutilized node is deleted and its children are added to its neighbours.

In a bootstrap scenario server managers connect to each other and execute leader election to determine the root of the management hierarchy. Once this root is chosen the other nodes can be added to it using a default "add server node" operation. Nodes can find each other by providing an initial neighbour cache or by using separate approaches such as the Dynamic Domain Name System.

Because there are only a few restrictions, this structure can be created and updated swiftly, enabling dynamic restructuring of the management hierarchy. Furthermore, the equal depth of the various nodes ensures the various nodes at each level are either all management servers or all execution servers, making the set of managed servers at each level more homogeneous.

B.5.2 Algorithm details

We will now demonstrate how the general $applace$ function can be executed in a hierarchical structure. As servers are grouped hierarchically, each server s has a parent $parent(s) = p$, $p \in S$ and a set of children $children(s) \in S$, except for the root control node, which has no parent, and execution nodes, which have

no children. Every management server executes the $applace$ function. We will now consider a single cluster with management server m, which will execute the $applace$ function with modified inputs and outputs:

$M_m = applace(\hat{Ra}, \hat{Rd}_s, \hat{Rd}_l, M'_m)$

As Rd_s is generally taken to be static, it can be considered as application information, available to every management node involved in managing a specific application, so $\hat{Rd}_s = Rd_s$. M'_m is always present on each management node, except in bootstrap scenarios where the general bootstrap scenario of the specific $applace$ function needs to be used. This leaves $\hat{Ra}$ and $\hat{Rd}_l$, which need to be aggregated across the management tree.

B.5.2.1 Resource availability aggregation

Server resource information $\hat{Ra}$ at the lowest level can be observed from the child nodes and can be used directly, so if a child $c \in children(m)$ is an execution server, $\hat{Ra}_c^r = \hat{Ra}_c^r$ for all resource types r. At higher levels, in every management server, an aggregated value of child resource information must be determined. Whereas a single server has a definite resource availability Ra_s, a cluster of servers does not. As the placement algorithm can only function using definite values for Ra, a value must be determined for the clusters by aggregating the values of its children. As a result, aggregated resource information $\hat{Ra}$ needs to be determined which indicates how many resources its parent can realistically expect to allocate on the cluster. This estimation will have to be revised whenever underutilization of resources or unrealistic resource allocation occur. For a resource $r \in \Gamma_l$ and a cluster c an aggregated estimation $\hat{Ra}_c^r$ can be determined:

$$\hat{Ra}_c^r = wE_{r,c}^{low} + (1-w)E_{r,c}^{max} \tag{B.1}$$

$$E_{r,c}^{low} = C_c^r + U_c^r \tag{B.2}$$

$$E_{r,c}^{max} = \sum_{n \in children(c)} \hat{Ra}_n^r \tag{B.3}$$

The complete estimation is denoted in the first equation, with E^{low} an estimate of available resources that will generally be close, but slightly too low and E^{max} an upper bound which will be too high. The actual ratio will be determined by a weight w, which will be determined experimentally. E^{max} is the sum of all available resources of the children. Finally, E^{low} combines two values: C_c^r, the amount of resource r currently allocated in the cluster, and U_c^r, an estimate of the remaining usable space on all servers on which more application instances can be instantiated:

$$C_c^r = \sum_{s \in children(c)} \sum_{a \in A} (M'_c)^r_{s,a} \tag{B.4}$$

$$U_c^r = \sum_{s \in \Upsilon} (\hat{Ra}_s^r - \sum_{a \in A} (M'_c)^r_{s,a}) \tag{B.5}$$

$$\Upsilon = \{s \in S | \forall r' \in \Gamma_s : \hat{Ra}_s^r - (M'_c)^r_{s,a} < \min_{a \in A} \hat{Rd}_a^r\} \tag{B.6}$$

Equation B.4 determines the amount of resources allocated by the current placement matrix. This can be achieved by summing the allocations in the cluster's placement matrix for all servers and applications. Equation B.5 is used to evaluate the remaining amount of free space on the server. To this end, the remaining resource availability of all servers on which more applications can be executed, the set Υ, are summed.

To determine the set of usable servers Υ, minimal strict resource requirements for all applications are determined. If the resource availability on a server for a resource $r \in R_s$ is less than the minimal resource requirement, no additional applications can be instantiated on this server. Therefore, the remaining resources of this server are not added to the aggregated resource information. This is denoted formally in Equation B.6.

In a bootstrap scenario there is no current allocation, so $C_c^r = 0$. Hence, the first value of $E_{r,c}^{low}$ will be the sum of all available resources, the same as $E_{r,c}^{max}$ which will cause overallocation of applications on the selected cluster. The cluster will then place as much of the load as possible, maximizing the amount of allocated resources. A second execution will have much higher C_c^r, and severely reduced U_c^r, leading to a better second estimate. The value of C_c^r gives an accurate representation of the current resource allocation. The value of U_c^r only gives an upper bound on possible resource availability. As the share of U_c^r in the estimation decreases significantly after one allocation, the value of a second estimation will be much more accurate than the initial estimation. Consequently, doing multiple placements will increase placement quality.

B.5.2.2 Demand decoupling

The other input value $\hat{Rd}$ determines the application demand. Here, lowest-level clusters gather application demand information and send this to its parent nodes. Only the root has a total overview of the application demand Rd and schedules based on this information, so at the root $\hat{Rd} = Rd$. These scheduling requirements are passed on to its children, who can then start scheduling based on this information. This method has the disadvantage that all management levels need to be passed to enable scheduling.

These levels can be decoupled however, by changing the type of information passed on between these levels. Application placement yields a placement matrix M. Instead of directly passing on application demands as per M, two values are propagated across the tree for every application a: an application share $\sigma_a \in [0..1]$, and total application demand D_a^r for resource r. Server resource demand can then be calculated as $Rd_a^r = \sigma_a \cdot D_a^r$. At the root level, $\sigma_a = 1$ for all applications, ensuring the full application demand will be met, leading to $\hat{Rd} = Rd$ as expected. The advantage of this approach is that passing on σ is much cheaper than directly using $\hat{Rd}$ as the latter requires placement calculation at higher levels, while the former only distributes a single value across the hierarchy. It allows management nodes to work independently, instantly reacting to changing demands.

This still requires the highest level of the hierarchy to know every application active in the system, which makes application placement more expensive as an increase in applications leads to an increase in execution time. In realistic situations however, not every application needs to be known at every level as smaller applications can be managed by only a part of the management tree. We use application delegation to resolve this issue. A management node can delegate an entire application by assigning an application share of 1 to a specific application on single child. If a parent delegates an application, the child can remove all of the application's resources from its aggregated resource information, and the parent no longer needs to monitor the application's performance information and placement. If the child is no longer able to achieve the required placement, it can delegate the application upward towards its parent, once again making it responsible for the application's management.

B.5.2.3 Overview

In summary, to execute the centralized algorithm, we retain non-varying resource demand, we aggregate resource availability and we decouple the different management levels by using application shares instead of resource demand. Figure B.4 illustrates how the various inputs are combined in a management server. A single management server is shown, together with its various inputs and outputs.

B.6 Evaluation results

We modified an algorithm from the literature [2] in a hierarchical fashion. The original algorithm operates in a centralized fashion with a complexity of $O(n^{2.5})$ with n the number of servers $|S|$, causing scalability issues as the server count increases. The hierarchical use of this algorithm solves these issues by introducing clusters grouping servers. The centralized algorithm is then executed on smaller clusters. We compared the performance of the hierarchical system using multiple

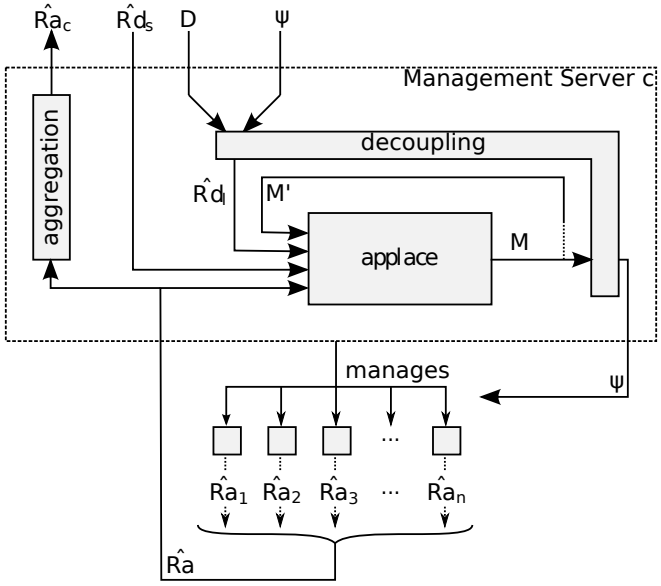


Figure B.4: The origin and destination of the different applace in- and outputs. A single management server, containing the applace-function, aggregation and decoupling mechanisms is shown.

values for C_{max}. We choose $C_{min} = \frac{C_{max}}{2}$. We considered application placement using memory and CPU as requirements with memory as loose requirement and CPU as strict requirement.

We used simulation of the cluster to evaluate the centralized and hierarchical algorithms where parallel tasks were executed sequentially and network overhead was simulated. Communication overhead between management nodes was simulated using a Gaussian distribution with mean 30ms and variance 10. Execution times were measured using a Linux server with an Intel Core i3 CPU (2.93GHz) with 4GiB of memory.

For every datapoint, multiple (20) random datacenters and sets of applications were generated, after which the measurements for the different datacenters were averaged. A random server has a CPU capacity, randomly picked from the set {1GHz, 1.6GHz, 2.4GHz, 3GHz} and a memory capacity from the set {1GB, 2GB, 3GB, 4GB, 8GB}. A set of random applications A is generated. Individual applications a are randomly generated by choosing a memory capacity from the set {400MB, 800MB, 1.2GB, 1,6GB} and allocating a random size $\theta_a \in [0..1]$ to it. The total application size, $\Theta = \sum_{a \in A} \theta_a$ can then be used to determine the total application share $\frac{\theta_a}{\Theta}$. Using a total CPU load for the entire datacenter L_{CPU} and individual application shares, we can then determine the demand for the application $Ra_a^{CPU} = L_{CPU} \frac{\theta_a}{\Theta}$. We used $w = 0.9$ as experiments have shown

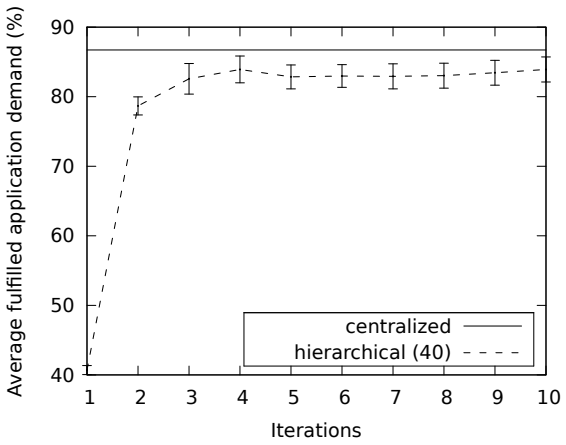


Figure B.5: The quality of the allocation after subsequent placement calculations ($C_{max} = 20$, $|S| = 50$). Standard errors are shown as well.

this weight yields good results. High weights lead to good estimations giving a higher weight to E^{low}, as discussed in Section B.5.2.1. A weight of 1 would lead to a too low estimate and would make it impossible to improve on bad allocations.

We evaluated a server and application configuration where application demands remain static. We evaluate the speed and quality of a single allocation on a datacenter with a heavy load ($L_{CPU} = 1$). We measure allocation quality by comparing the average application satisfaction $Q_a = \frac{\sum_{s \in S} M_{s,a}^{CPU}}{Ra_a^{CPU}}$ of the different allocation strategies. Allocation quality is measured by comparing the desired amount of resources with the allocated amount of resources. A satisfaction of 1 means all demands are satisfied. For this test we used an equal number of randomly generated applications and servers. As our architecture assumes that the management system itself is executed on a cloud instance, and we work using $L_{CPU} = 1$ achieving full satisfaction will be impossible as part of the capacity will be consumed by the management system itself, pushing the centralized algorithm to its limits.

In Section B.5.2.1 we mentioned that the quality of $\hat{Ra}_c^r$ increases as multiple placements are made. Figure B.5 illustrates the effect of this: subsequent placements increase in quality until a threshold is reached, after which placement quality stagnates. The first placement has a relatively bad quality, caused by the low quality of the initial estimate. The second placement greatly increases placement quality and after a third placement the quality of the placement has a value near its maximum. Because of this we will execute hierarchical placement three times in following experiments, leading to a high quality at the cost of higher execution times.

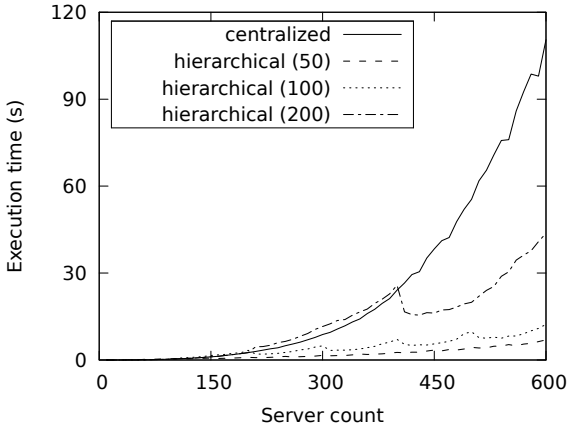


Figure B.6: Execution time of the hierarchical and centralized allocation strategies with varying server and application counts ($|A| = |S|$)

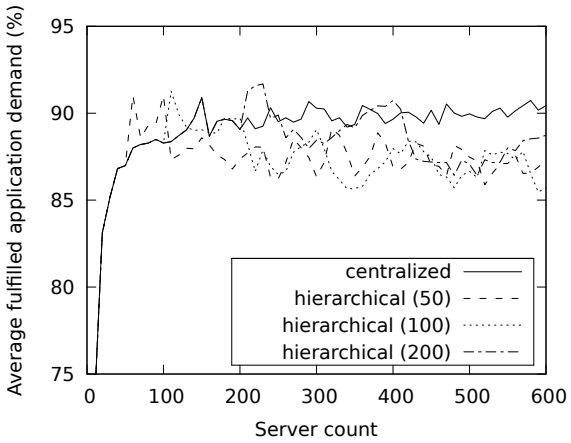


Figure B.7: Comparison of the average satisfied demand of the different allocation strategies ($|A| = |S|$)

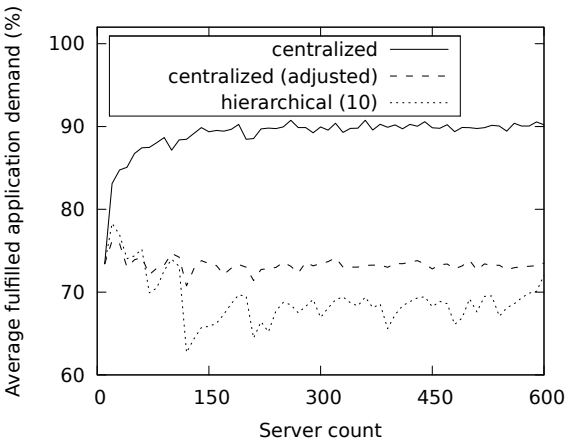


Figure B.8: Illustration of the management overhead and performance penalties induced by the hierarchy with a very low branching factor ($C_{max} = 10$)

We compared the performance of the centralized approach with that of five different variants of the hierarchical approach, with $C_{max} = 50$, $C_{max} = 100$ and $C_{max} = 200$. The allocation speed of the different techniques is illustrated in Figure B.6. As we expected, the hierarchical approach executes faster than the centralized approach. As indicated in Figure B.7, the average satisfied demand of applications is best provided for by the centralized algorithm once $|S| \gg C_{max}$. As long as $C_{max} > |S|$ the hierarchical approach is the same as the centralized algorithm. Once branching occurs a tree is formed. This temporarily *increases* both placement quality and allocation cost, as the management system repeats the allocation multiple times. As the datacenter size increases, allocation performance decreases while the difference between centralized execution times and hierarchical execution times increases. The higher C_{max}, the higher the placement quality, but the slower the execution.

Branching causes two types of performance penalties. As additional servers are used for the management process, they can no longer be used to execute applications, decreasing maximum achievable application satisfaction. Furthermore, information is fragmented by the process, allowing for over- or under-allocation of application demand on clusters.

The effects of the different performance penalties in the hierarchical approach are illustrated in Figure B.8. Here we compare the satisfied demand of the centralized algorithm with that of the hierarchical approach and that of an adjusted centralized algorithm, where servers used in hierarchical management are kept idle. In this case we used a management tree with an unrealistically low C_{max} to illustrate both types of performance penalties. While the centralized approach

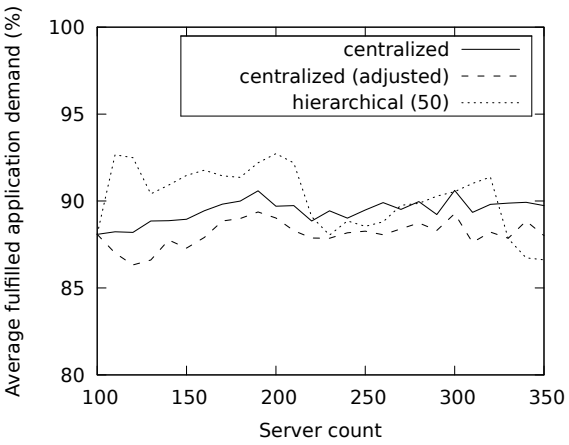


Figure B.9: Illustration of the management overhead and performance penalties with a higher branching factor ($C_{max} = 100$)

uses all servers but one, where the algorithm itself executes, the adjusted centralized algorithm does not use any of the servers used in the management hierarchy. Because less servers can be used for actual application execution, the achievable placement quality is lower. As the hierarchical placement uses the same servers as management servers, the adjusted centralized placement offers an upper bound for the placement quality. The quality difference between the centralized approach and the adjusted centralized approach (in this specific case $\pm 20\%$) is caused by the management hierarchy itself and stems from the choice to place the management infrastructure on the cloud itself. Not doing so would require a separate management infrastructure which would be less dynamic. The quality difference between the adjusted centralized approach and the hierarchical approach is caused by fragmentation of information and under-allocation. Picking higher a C_{max} causes performance penalties to decrease significantly while increasing allocation performance by repeated execution of the allocation algorithm, as illustrated in Figure B.9, where hierarchical execution at times even surpasses centralized execution due to repeated executions.

The performance of the hierarchical system is still impacted by the number of applications, as shown in Figure B.10, where a constant number of applications is used ($|A| = 50$) and significantly better performance is achieved. The cause of this is that the number of applications also has an impact on the performance of the application placement algorithm. When considering a first placement, all applications need to be taken into consideration, leading to very high placement times. After this, some application responsibilities are delegated to specific instances, decreasing costs for subsequent placements. This implies that when many

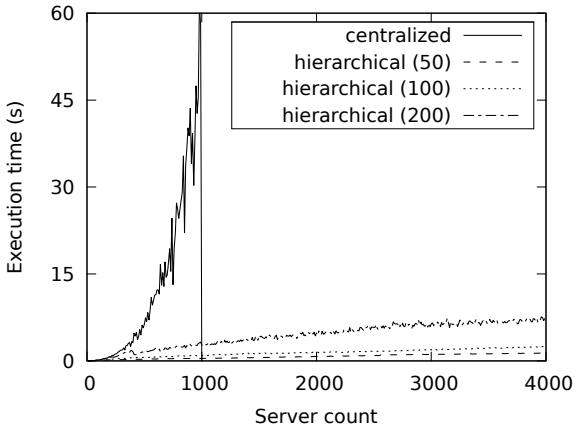


Figure B.10: Execution time of the hierarchical and centralized allocation strategies with fixed application counts ($|A| = 50$). Execution time of the centralized algorithm was measured up until 1000 servers.

applications are in use, a lower branching factor should be chosen as this leads to more management nodes and thus more delegation of applications.

Figure B.11 illustrates the number of applications used in application placement per level. The graph illustrates the maximum number of applications used in application placement at a given tree level. The lower this number, the faster a placement at this level can occur. From the graph, we see that a large part of applications continues to be managed at the root level, but at lower levels, significantly less applications are managed per-node. This implies lower-level nodes can execute placement much faster. This enables the system to react fast based on local data, swiftly yielding a local solution, while still maintaining a globally good solution once higher levels have executed application placement.

B.7 Conclusion

In this appendix we presented a hierarchical management system for cloud environments. Management nodes automatically order themselves in a structure inspired by that of B-Trees and each node executes a centralized placement algorithm for which inputs are generated by the management system.

The centralized approach leads to higher placement quality at the cost of higher execution times. As datacenters scale, the execution time also increases, leading to slow reactions to changing environments. At this point, using the hierarchical approach makes sense, as it is much faster and, thus, more scalable. The number of servers managed by each node has a large impact on the execution speed and

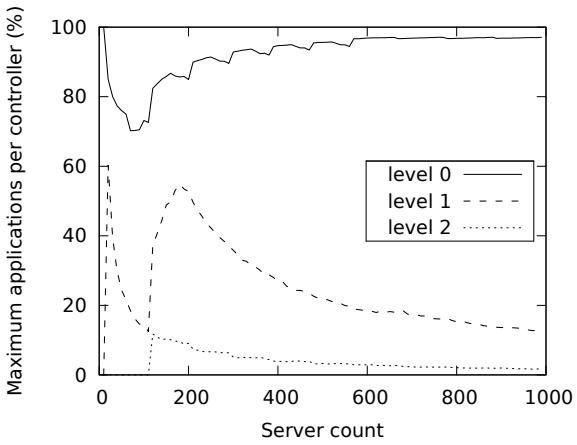


Figure B.11: The maximum number of applications known per node at different management levels ($C_{max} = 10$)

on the quality of allocation, as it directly influences both the number of servers used in the management system and the quality of the placement itself. These results are in accordance with our earlier work concerning hierarchical management. Introducing hierarchies increases scalability of management systems, and eases distribution of context, in this instance calculated application demands.

In future work, we intend to study and improve the robustness of the hierarchical management system. It would also be useful to add adaptiveness to the system by dynamically restructuring the management tree and adjusting system parameters, such as branching factor and estimation weights. These adjustments should lead to improved resource estimates, better resource allocations and faster placements. We also need to examine how the techniques demonstrated in this appendix can be applied to other centralized cloud management algorithms.

References

[1] J. Famaey, S. Latré, J. Strassner, and F. De Turck. *A hierarchical approach to autonomic network management*. In 2nd IEEE/IFIP International Workshop on Management of the Future Internet (ManFI), pages 225–232, 2010. doi:10.1109/NOMSW.2010.5486571.

[2] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. *A scalable application placement controller for enterprise data centers*. In 16th international conference on World Wide Web, pages 331–340, 2007.

[3] F. Wuhib, R. Stadler, and M. Spreitzer. *Gossip-based resource management for cloud environments*. In 6th International Conference on Network and Service Management (CNSM), pages 1–8, 2010. doi:10.1109/CNSM.2010.5691347.

[4] Y. Li, F.-H. Chen, X. Sun, M.-H. Zhou, W.-P. Jiao, D.-G. Cao, and H. Mei. *Self-adaptive resource management for large-scale shared clusters*. Journal of Computer Science and Technology, 25(5):945–957, 2010. doi:10.1007/s11390-010-1075-6.

[5] J. Rolia, A. Andrzejak, and M. Arlitt. *Automating enterprise application placement in resource utilities*. In 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM), pages 118–129, 2004. doi:10.1007/978-3-540-39671-0_11.

[6] T. Kimbrel, M. Steinder, M. Sviridenko, and A. Tantawi. *Dynamic application placement under service and memory constraints*. In International Workshop on Efficient and Experimental Algorithms, April 2005. doi:10.1007/11427186_34.

[7] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi. *Dynamic placement for clustered web applications*. In 15th international conference on World Wide Web, pages 595–604, 2006. doi:10.1145/1135777.1135865.

[8] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguadé. *Utility-based placement of dynamic web applications with fairness goals*. In 11th IEEE/IFIP Network Operations and Management Symposium (NOMS), pages 9–16, 2008. doi:10.1109/NOMS.2008.4575111.

[9] C. Adam and R. Stadler. *Service middleware for self-managing large-scale systems*. IEEE Transactions on Network and Service Management, 4(3):50–64, 2007. doi:10.1109/TNSM.2007.021103.

[10] C. Low. *Decentralised application placement*. Future Generation Computer Systems, 21(2):281–290, 2005. doi:10.1016/j.future.2003.10.003.

[11] S. Schneider, A. Meisel, and W. Hardt. *Communication-aware hierarchical online-placement in heterogeneous reconfigurable systems*. In IEEE/IFIP International Symposium on Rapid System Prototyping, pages 61 –67, 2009. doi:10.1109/RSP.2009.23.

[12] B. Hudzia, M.-T. Kechadi, and A. Ottewill. *TreeP: A tree based P2P network architecture*. IEEE International Conference on Cluster Computing, pages 1–15, 2005. doi:10.1109/CLUSTR.2005.347022.

[13] R. Bayer and E. McCreight. *Organization and maintenance of large ordered indices*. In ACM SIGFIDET Workshop on Data Description, Access and Control, pages 107–141, 1970. doi:10.1007/BF00288683.

C

Ambient-aware continuous care through semantic context dissemination

F. Ongenae, J. Famaey, S. De Zutter, S. Verstichel, S. Latré, A. Ackaert, P. Verhoeve, and F. De Turck

The ultimate ambient-intelligent patient room contains numerous devices to automatically sense and adjust the environment, monitor the patient and support the staff members and patients in their day to day activities and tasks. This results in an overwhelming amount of data, which needs to be processed by the different applications. In this appendix, a context-aware, ambient-aware and pervasive framework is proposed which gathers, integrates, exploits and dynamically filters the relevant context information for the various applications by using a continuous care ontology. The platform allows applications to dynamically generate and register filter rules, such that they only receive contextual information, which is of interest at that time. To illustrate the benefits and performance of this platform, a proof of concept implementation is presented consisting of a sophisticated and person-oriented nurse call system supported by a localization and a home automation component.

C.1 Introduction

C.1.1 Background

In recent years, the complexity of institutional care settings has been increasing due to societal factors such as the growth of the care unit size and specialized care and the decline of staffing. This requires a more efficient scheduling and use of staff resources. In future years, these trends will accelerate due to an aging society.

To deal with these issues, information technology is often introduced. The current institutional care settings contain numerous devices to support caregivers in carrying out their everyday activities, e.g., electronic medical records and monitoring equipment. However, the high amount of technology further increases the complexity of the work flows, because the caregivers are directly faced with the complex technologies [1]. The caregiver has to use several sources and devices to consult and insert data even when carrying out a single task. This is very time-consuming. Due to this inadequate technology integration, the large amount of data being generated by the devices and the heavy workload of staff members, it is not rare for important events, e.g., early indications of declining condition of a patient, to be missed.

Consider for example a patient with a concussion, who needs to be in a dark environment to heal. Today, the staff members are responsible for dimming the lights each time they enter the room. Consequently, each staff member has to be aware of the patient's condition. If an uninformed person enters the room or a wrong button is pressed, this can cause physical pain for the patient. However, if the lighting control system would be aware of the patient's needs, it can automatically turn on the light to the correct level when it detects that the nurse enters the room. Moreover, a light sensor could be used to monitor the light intensity in the room and alert a staff member if a pre-defined threshold is crossed.

C.1.2 Ambient-aware continuous care

The ultimate ambient-intelligent care room of the future uses numerous devices to sense the needs and preferences of the caregivers and patients and adapt itself accordingly. This implies an emerging demand for the integration and processing of the heterogeneous data offered by the different technologies available in the room.

The ACCIO [2] (Ambient-aware provisioning of Continuous Care for Intramuros Organizations) project aims to realize this goal by developing a context-aware, ambient-intelligent, pervasive and semantic platform which exploits, integrates and filters the available heterogeneous data. The platform enables technology to blend into the background, using sensors and actuators to sense and adapt the environment. This frees the caregiver from the cumbersome task of manag-

ing the different technologies. However, to achieve this goal, the platform must be able to interpret the meaning and adequately filter the relevant information out of the huge amount of heterogeneous care data provided by the sensors. Unorganized data is voluminous, but has no meaning on itself as it has no relationships or context. Information is data that has been given meaning by defining relational connections. For this, the platform uses an ontology, which is a semantic model that formally describes the concepts in a certain domain, their relationships and atrributes. In this way, an ontology encourages re-use and integration. By managing the data about the current context in an ontology, intelligent algorithms can be defined that take advantage of this information to automate, optimize and personalize the continuous care of patients. Referring back to the previous example, this means that the ontology models the patient's condition and the nurse's location. Algorithms are defined that automatically put the light to the correct level based on the context information in the ontology. Afterwards, the nurse can decide to overrule this decision by adjusting the light level in the room manually.

C.1.3 Related work

A number of generic context platforms [3] have been developed to relieve application developers from the aggregation and abstraction of context information, and the derivation of high-level contexts. At the same time there has been a trend towards developing context platforms with formal context models based on ontologies allowing the integration and exploitation of more specific context knowledge with high-level context information using reasoner components. Most of these approaches assume as central component a knowledge component, which integrates and gathers all the context information. Various applications can then be built by querying this context model. Our approach uses as central component a semantic publish/subscribe system that uses a set of core ontologies. This component forwards the gathered context information to the different applications, but does not retain this information. Applications have their own knowledge component, which is a domain-specific extension of a subset of the core ontologies. The core ontologies guarantee that information can be exchanged between the different applications and that the context information gathered can be distributed to all the interested applications. This decreases the ontological commitment from the different applications and increases the overall performance as the different applications each only retain the context they are interested in and perform their reasoning in parallel on smaller data sets and often smaller knowledge models.

Publish/subscribe systems have evolved from static topic-based to dynamic content-based systems. By augmenting the content with semantics, subscriptions can be created which take into account the actual meaning of the content. Several semantic publish/subscribe systems have been proposed in literature [4] which dif-

fer in the method proposed to relate subscriptions to messages, namely based on Resource Description Framework (RDF) graph-matching, ontological inferencing and attribute-value pair matching. Our approach is most closely related to semantic publish/subscribe systems that use OWL ontological inferencing. These systems represent subscriptions as OWL concepts and messages as concept instances. Inferencing engine is used to determine if a message instance satisfies the constraints of a subscription class. This approach is more expressive than the custom RDF graph-matching algorithms as it allows new, non-asserted knowledge to be inferred. Moreover, it does not limit the format that messages are allowed to take as is the case in systems based on attribute-value pairs. Additionally, our approach differs from other OWL inferencing publish/subscribe systems as it also allows the use of Jena rules and Semantic Web Rule Language (SWRL) to define subscriptions. These rule languages support a wide range of built-in operators which greatly increases expressiveness. Jena rule inferencing exhibits the best scaling behavior as the the amount of subscription rules and message complexity increases [5], but is less expressive than SWRL or OWL inferencing. The choice between these three reasoning approaches allows balancing expressivity and performance according to the use case. Finally, our approach allows applications to automatically generate and register new filter rules based on the current context.

This generic context-aware platform with semantic publish/subscribe mechanism has already been applied to several autonomic network management scenarios such as the management of a multimedia access network and the management of a cloud data center [5]. A last contribution of this appendix is how the generic platform can easily be applied to tackle several challenges in ubiquitous healthcare such as filtering large amounts of context data, knowledge management and exploitation. Context-awareness is a hot research topic within the healthcare domain [6, 7]. The healthcare scenario has some specific implications which differentiate this scenario from other scenarios to which the platform can be applied. An often overlooked fact is that the strength of the context-aware platform is heavily dependent on the correctness and completeness of the used knowledge model. Constructing this model is a difficult task in domains such as continuous care where most knowledge is implicit and best practices are not rigorously documented. Therefore, we propose a participatory ontology engineering methodology which promotes user participation, while not overloading the involved stakeholders as time is a valuable resource within the eHealth domain. Another challenge in the healthcare scenario is the fact that wrong decisions made by the system can have severe implications. However, context data delivered by sensors is very unreliable. Decisions made based on wrong or incomplete sensor data might thus not be correct. Therefore, the developed platform provides techniques to detect unreliable sensor observations and annotate them. This allows developing Quality of Context-aware (QoC-aware) algorithms that take the reliability and correctness

of the context data into account. Additionally, the users are always able to overrule the decisions of the platform. This also improves the acceptance rate of the new technology as the users feel in control of the system.

C.1.4 Objective & organization

The goal of this appendix is to formulate an answer to the following research questions: (1) How do we dynamically filter the large amount of data such that the different components only receive the data that is relevant to them at that moment? (2) How do we model the continuous care data gathered and communicated between the different components in a formal and semantic manner? (3) How can intelligent algorithms and applications that optimize continuous care processes be built based on this developed model?

The remainder of this appendix is structured as follows. In Section C.2, the architecture of the ACCIO platform is presented. Section C.3 elaborates on the specifics of the platform using an illustrative example, namely realizing a sophisticated nurse call system supported by a Localization and Home Automation Component. Section C.4 evaluates the performance of the platform and discusses its potential impact on the delivery of continuous care. Finally, the conclusions are highlighted in Section C.5.

C.2 Architecture of the ACCIO platform

The ambient-intelligent care room consists of various devices, e.g., sensors and nurse call buttons, and intelligent applications that process the generated data. A communication substrate is needed to glue these components together and orchestrate collaboration. For this, the *Semantic Communication Bus (SCB)* [5] is designed, as visualized in Figure C.1. The SCB orchestrates the communication of semantically enriched data. This allows filtering data based on meaning rather than on string patterns. The SCB interprets the data by using *core ontologies* which model the information being exchanged in a continuous care domain. For example, the ontologies model that the environment contains light sensors, which make observations about the light intensity at a location. Although ontologies are widely accepted within the eHealth domain, most of them focus on biomedical research, e.g., Galen Common Reference Model or the Gene Ontology. Little work has been done on developing ontologies to support the continuous care of patients. Such an ontology has to contain information about the profile of staff members and patients, roles and responsibilities, administrative information, etc. To tackle this issue, continuous care core ontologies were co-created with the stakeholders using a participatory methodology developed within the ACCIO project. This is further detailed in Sections C.3.2 and C.3.3.

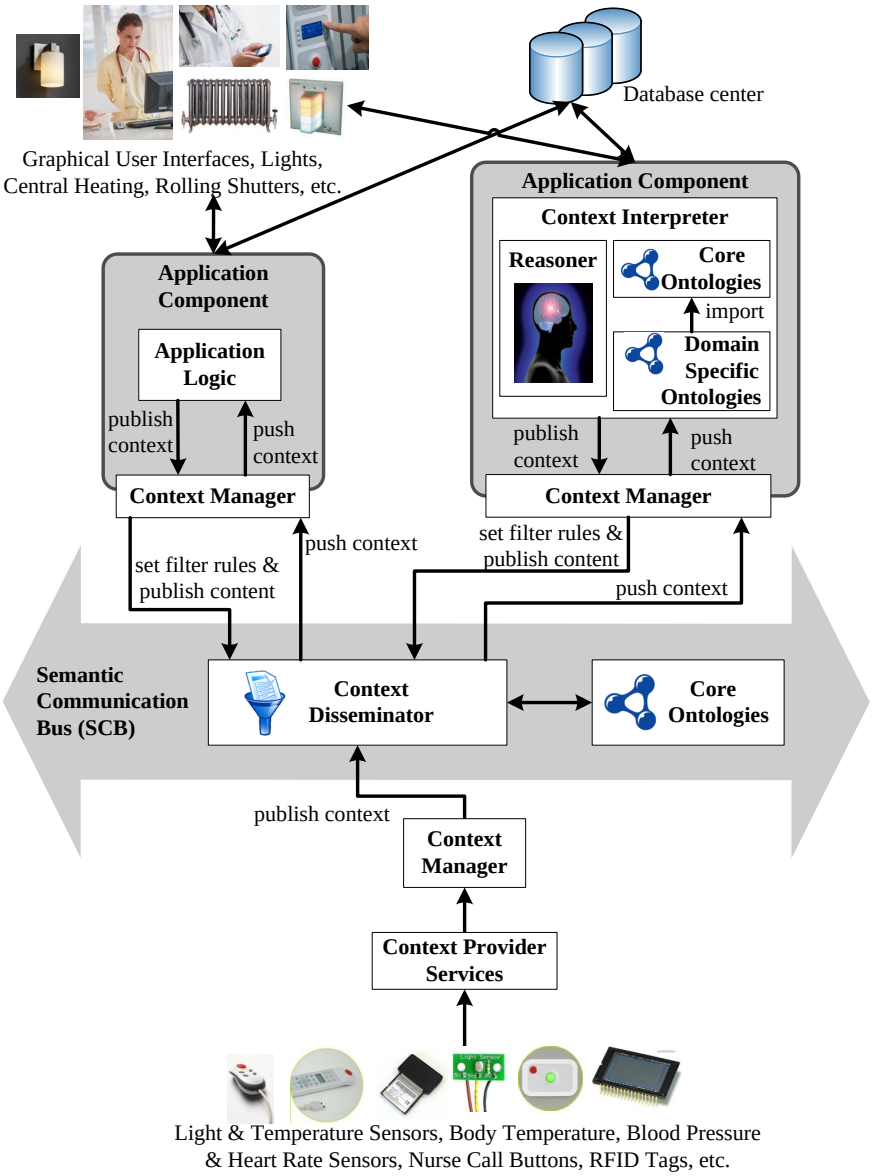


Figure C.1: Architecture of the ACCIO platform using a Semantic Cummunication Bus (SCB) for interaction, collaboration and orchestration

As depicted at the bottom of Figure C.1, the *Context Provider Services* receive data from the devices in the environment and transform it to context information which adheres to concepts in the core ontologies. This semantically enriched data

is forwarded to the *Context Manager*, which publishes it on the SCB. For example, the *Location Provider Service* is used to publish location information, e.g., about staff members or patients, on the SCB.

As is the case with classic publish-subscribe mechanisms, the SCB allows components to subscribe to context information, which is relevant to them at that moment, through the *Context Disseminator*. The components use a Context Manager, which contains (a subset of) the core ontologies used by the SCB, to specify the context they are interested in, by defining *filtering rules* and registering them with the Context Disseminator. For example, a nurse alerting component indicates that it is only interested in light intensity observations, crossing a particular threshold and coming from rooms with patients who suffer from a concussion. These rules are expressed using concepts from the core ontologies. When continuous care information is published on the SCB, the Context Disseminator matches this published event with the registered filter rules by reasoning on the information in the ontology. If a match is found, the information is forwarded to the components that subscribed to the filter rule. The use of these filter rules thus reduces the amount of care data that is forwarded to the applications. This prevents these components from being flooded with huge amounts of data generated by the sensors and devices in a truly pervasive and ambient-aware patient room. Moreover, components can register new filter rules on the fly based on the current context, which greatly improves the flexibility of the platform. This is further detailed in Section C.3.4.

As visualized in the top right of Figure C.1, the intelligent applications, receiving context information from the SCB, also use ontologies to model their specific domain and perform sophisticated reasoning. These *domain specific ontologies* extend concepts in the core ontologies, such that the context delivered by the SCB is directly understood by the application logic. Static information about the environment, e.g., names of patients or locations of sensors, is collected from databases. As a result of the reasoning, the applications can adapt the environment by controlling devices, e.g., lights or beepers. The applications can also publish their conclusions on the SCB through the Context Manager. This way, they can be picked up by other components to perform additional reasoning. For example, a first component computes the locations of people, based on the available sensor information, and publishes these locations on the SCB. A second component uses this location information to assign staff members to tasks, while a third component uses it to regulate the light level in a room.

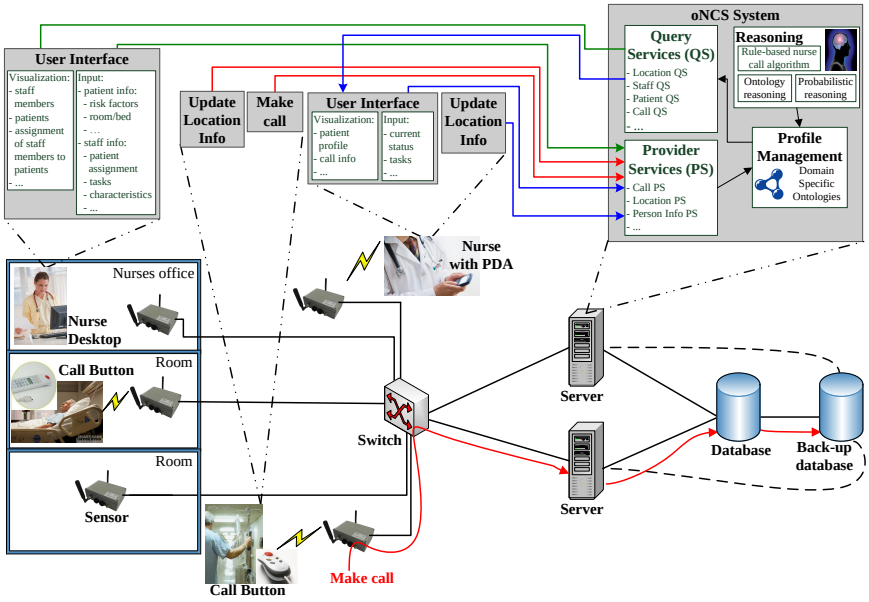


Figure C.2: General concept of the ontology-based Nurse Call Management System (oNCS) with probabilistic priority assessment and profile management

C.3 Use case: Ontology-based Nurse Call System

C.3.1 Scenario description

Nurse call systems are a very important fundamental technology in continuous care. However, the current systems are very static as call buttons are fixed to the wall of a room. There is an increased risk when patients become unwell inside a hallway, staircase or outside as they cannot call a nurse. Additionally, the nurse call algorithm consists of predefined links of beeper numbers to rooms. Consequently, the system does not take into account the various factors specific to a situation, such as the risk factors of a patient, e.g., heart patient or confused, or the competences of the staff, e.g., nurse or caregiver.

The increased introduction of electronic devices in continuous care settings facilitated the development of the ontology-based Nurse Call System (oNCS), visualized in Figure C.2, which allows patients to walk around freely with wireless nurse call buttons. Additionally, this platform manages the profiles of staff members and patients in an ontology. A sophisticated nurse call algorithm was developed by the authors. It first determines the priority of the call using probabilistic reasoning algorithms, which take into account the origin of the call and the risk factors of the patient. Next, the algorithm finds the most appropriate staff member to handle the call. It dynamically adapts to the situation at hand by taking into

account the context, e.g., location of the staff members and patients. A detailed description of this platform can be found in Ongenae, et al. [8].

To better illustrate the benefits of the ACCIO Platform, an extension of the oNCS system is presented in this appendix, which allows nurse calls to be automatically launched based on the data generated by the electronic equipment and sensors in the environment, e.g., alerting a nurse when the light intensity is too high in the room of a patient with a concussion. The proposed extension provides a solution to potential risky situations being missed because the caregivers are overloaded with constantly monitoring and orchestrating all the devices in the ambient patient room, making it more applicable to real-time scenarios.

To realize this extension, two other applications were designed, namely a Localization and Home Automation Component. The first determines the location of patients, staff members and devices. The latter automatically controls the ambient-intelligent activity in the room of the patient, e.g., switching on the lights at the appropriate level. The oNCS system, the Home Automation and Localization component each represent an *Application Component* which uses the SCB to filter the context information that is relevant to it at that moment. The architecture of these components is thus very similar to the architecture of the *Application Component* at the top right of Figure C.1. These three components, the SCB and the sensors form together a scenario-specific implementation of the ACCIO platform.

C.3.2 Co-creation methodology

The incorporation of ontology engineering tasks in knowledge-empowered organizations, such as hospitals, can prove to be a hindrance if not done in a way that is seamless to the day-to-day activities of the nurses, patients and doctors. To resolve this issue, the construction of the ontology should be user-driven. This will not only facilitate the acceptance of this new technology, but it will also empower the staff to make suggestions for changes and thus shape the common information space to their needs.

The existing ontology engineering methodologies are rather extreme in their choices to include domain experts [9]. There are methodologies, e.g., Tove, Enterprise and Methontology, that only discuss the scope and the requirements of the ontology with the domain experts. The rest of the ontology life-cycle is controlled by the knowledge engineer. Recently, a human-centered approach (HCOME) was proposed, which offers user-friendly and collaborative tools that allow domain experts to construct, merge and discuss their own ontologies. The knowledge engineer only delivers (technical) support in this process.

A methodology was developed that holds a middle ground between these two extremes. The ontology engineering process actively involves social scientists, ontology engineers and stakeholders. It promotes user participation while taking into

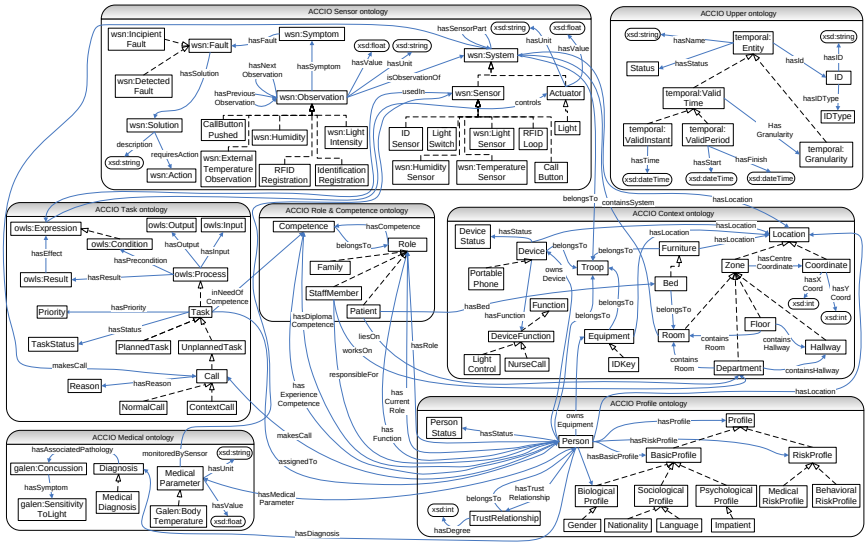


Figure C.3: Overview of the most prevalent classes and their relations of the seven core ontologies for the intelligent nurse call system use case. The dashed arrows depict subclass relationships. The blue arrows represent relationships between concepts and individuals.

account that time is a valuable resource within the eHealth domain. The methodology proposes several participatory methods and techniques to involve the users for example by performing observations, role-playing and discussing scenarios in hands-on workshops to capture the daily and preferred continous care practices and constructing a sunny day scenario and personas. A detailed discussion and evaluation of the methodology can be found in Ongenae, et al. [2].

C.3.3 Resulting continuous care core ontologies

The participatory methodology resulted in seven ontologies for the continuous care domain, of which the most important concepts and relations, with respect to the use case detailed in Section C.3.1, are depicted in Figure C.3. The Web Ontology Language (OWL) was used to develop the ontologies.

The *ACCIO Upper ontology* describes general classes, relations and axioms. Most importantly this ontology enables that data can be related with a unique ID. The classes preceded by the namespace prefix *temporal* are imported from the *SWRL Temporal Ontology* to model complex interval-based temporal information. All the other core ontologies import this ontology and define all their concepts as subconcepts of `temporal:Entity`. This is not shown on Figure C.3 to not overload the figure.

The *ACCIO Sensor ontology* is the most important ontology for the use case.

The concepts preceded by the *wsn* namespace prefix are imported from the *Wireless Sensor Network ontology*, which was developed by the co-authors and allows attaching meaning to data values monitored by sensors. The `System` concept models a system and its components. An `Observation` represents a data value monitored by a system. However, context information is often unreliable as it is gathered by sensors which can be imprecise or erroneous. Moreover, the context information can be ambiguous as information gathered by different sensors can conflict or the context information might even be incomplete if there is no sensor information available. As it is this context information that determines the behavior and the strategies of the different components, it is important to make the quality of the context data explicit to prevent error propagation in the upper context management. To support the development of QoC-aware (Quality of Context-aware) algorithms, this ontology contains axioms and rules, modeled as `Symptom` concepts, which allow detecting specific phenomena in the observations published to the SCB. For example, the `LightIntensityBelowZeroSymptom` detects light intensity observations that are below zero. Axioms are provided that reclassify these symptoms as `Fault` concepts, e.g., the previously mentioned symptom is reclassified as a `FaultyLightIntensitySensor` indicating that the sensor that made the measurement is faulty. Additionally, a fault can be reclassified as a `Solution`, e.g., the previous fault is reclassified as the `DoNotUseSensor` solution indicating that measurements from this sensor should not be used in the algorithms. Consequently, the components can take these classifications into account in their filter rules and algorithms. For example, on the one hand, the components can register filter rules that indicate that observations annotated as `FaultyLightIntensitySensor` concepts should not be forwarded to the component. On the other hand, components could choose receive these annotated observations and process them differently in their own algorithms. The WSN ontology was extended with systems, sensors, actuators and their associated observations, faults and solutions that play an important role in continuous care settings, e.g., nurse call buttons.

The *ACCIO Context ontology* models the contextual environment information. `Troop` is the most important concept. It represents a logical grouping of entities that belong together, e.g., a patient with all his/her devices, sensors, actuators, room, bed and equipment. The composition of a troop dynamically changes based on the context. This ontology also contains all the information related to localization. A `Location` can be a coordinate or a zone.

The *ACCIO Profile Ontology* models the profile information about staff members and patients that was indicated by the stakeholders in the workshops as important. Each `Person` is associated with a `Profile`, which consists of a basic and a risk profile. The latter is defined by axioms and rules, which allows obtaining the risk profile of the patient by reasoning on the information in the basic profile.

The *ACCIO Role & Competence ontology* defines each role by its competences through axioms. This allows writing algorithms that find the most appropriate staff members to fulfill a task based on the required competences. Each person is then associated with competences and roles through five relationships: `hasFunction`, `hasRole`, `hasCurrentRole`, `hasDiplomaCompetence` and `hasExperienceCompetence`.

The *Galen Common Reference Model*, of which the concepts are preceded by the *galen* namespace prefix, represents clinical terminology. The *ACCIO Medical ontology* adds axioms and constraints to this imported terminology that express relations between this medical knowledge and concepts in the other ontologies.

Finally, the *ACCIO Task ontology* models continuous care process workflows. For this, the *OWL-S Process ontology* was imported, of which the classes are preceded by the *owls* namespace prefix. The `Process` concept models a process, which can return information and produce a change in environment based on the information it is given and the context. This is described by `hasInput`, `hasOutput`, `hasPrecondition` and `hasEffect` relations. A process can be composed of several other processes. The *Accio Task ontology* extends this ontology. The `Task` concept, introduced as subclass of `Process`, represents the various continuous care tasks. Consider for example, the task of assigning a person to a call. A `Call` is modeled as an unplanned task. A `Normal Call` is then modeled as a `Call`, which has as precondition that a patient pushes a call button. This task takes the patient as input and has as output the assigned caregivers. The effect of the `Normal Call` is that the assigned caregivers' cellphones ring.

C.3.4 Flexible and semantic publish/subscribe mechanism

The SCB allows components to subscribe to relevant context information by registering filter rules with the Context Disseminator. When a context publisher forwards information to the SCB, the Context Disseminator reasons on the core ontologies to determine which subscribers are interested in this information by computing whether the forwarded data satisfies at least one filter rule defined by the subscriber. If it does, the information is forwarded to the component.

C.3.4.1 Publishing context

The Context Provider Services are used to semantically annotate data with concepts from the core ontologies such that it can be interpreted by the SCB. To achieve a flexible system, in which published context and filter rules can easily be matched, an `Event` concept is added to the *ACCIO Upper Ontology*, which has a `hasContext` relationship to `temporal:Entity` concepts. As all the other core ontologies import the *ACCIO Upper Ontology*, this `Event` concept and `hasContext` relationship essentially becomes a part of all the other core on-

tologies too. These core ontologies have defined all their concepts as subconcepts of `temporal:Entity`. As such, all their instances can occur as range of the `hasContext` relation. No other modifications to the ontologies are needed to support the management of these events.

When a device wants to publish context information, the Context Provider Service creates an event, containing the data it wishes to publish. For example, to publish that the light sensor with ID L101 measured a light intensity of 100 lumen, the following instances are created:

- Ob **owl:instanceOf** Observation **and** Ob hasValue 100 **and** Ob hasUnit 'lm'
- LsId **owl:instanceOf** ID **and** LsId hasID 'L101'
- Ls **owl:instanceOf** LightSensor **and** Ls hasId LsId **and** Ob isObservationOf Ls
- Ev **owl:instanceOf** Event **and** Ev hasContext Ob

Similarly, applications can publish the results of their reasoning. For example, the Localization Component collects all the context information that gives an indication of the locations of staff members, such as RFID tags, and calculates their current locations, e.g., room 101. These locations are published on the SCB to be used by other applications, e.g., the oNCS system, as follows:

- Pid **owl:instanceOf** ID **and** Pid hasID 'AB452487'
- P **owl:instanceOf** Person **and** P hasId Pid
- R101 **owl:instanceOf** Room **and** R101 hasNr 101 **and** R101 isLocationOf P
- Ev **owl:instanceOf** Event **and** Ev hasContext R101

To support the aggregation of context data and allow filter rules to process multiple observations simultaneously, more complex Context Provider Services can be written. For example, instead of publishing each light intensity observation to the SCB separately as in the first example, a Context Provider Service is developed to calculate the average of all the light intensity observations in a room within a minute and only publish this event on the SCB. This event can be published as a normal `LightIntensity` observation or the aggregation process can be made explicit to the SCB by creating new types of `Observation` concepts in the core ontologies, for example `AveragedLightIntensityObservation`. Filter rules can then be written to process these types of events. Similarly, Context Provider Services can be written to aggregate the values of different types of sensors.

C.3.4.2 Subscribing to context

Filter rules are expressed by defining subclasses of the `Event` class. This way, the Context Disseminator can determine if the published context matches a filter rule by asking an OWL Reasoner, such as Pellet, if the published event belongs to the `Event` subclass defined by the filter rule [5]. The filter rule is defined as necessary and sufficient conditions, which the published event must fulfill to belong to this class. Moreover, the Context Disseminator can use the OWL reasoner to check if the filter rule is satisfiable, i.e., does not contradict with the knowledge already defined in the core ontologies. If the filter rule is unsatisfiable, the subscription of the filter rule fails and the class is not added to the ontology.

For example, as the Home Automation Component regulates the ambient-intelligent activity in the room of a patient, it is interested in location information and light intensity, humidity and temperature observations. It registers the following filter rule:

Event **and** hasContext **some** (LightIntensity
or ExternalTemperatureObservation
or Humidity
or Location)

Note that the two `Event` examples match with this filter rule and will thus be forwarded to the Home Automation Component. The first example matches because the ontology declares that each `Observation`, which has a unit of type `lm`, is an observation of type `LightIntensity`. Similarly, the second event matches because the ontology states that each `Room` is a subclass of `Location`.

However, the information in which an application is interested changes based on the current context. Instead of filtering all the context that might be needed at some point in time, the application components are able to generate new filter rules when the context changes. The filter rule generation process is thus made context dependent. To enable this, the ontology allows defining context dependencies between context. A context dependency (X,c,Y) means that context Y only needs to be filtered if the condition c holds for context X. Optionally, a range d can also be specified for the values of this context parameter Y. For example, the domain specific ontology used by the oNCS system defines the dependency:

(`Person`, `hasDiagnosis some` (`hasAssociatedPathology some` (`hasSymptom galen:SensitivityToLight`)), `LightIntensity`)

This dependency indicates that the oNCS system is also interested in measurements of type `LightIntensity` when a patient is detected who is sensitive to light. Additionally, the domain specific ontology associates each light sensitivity symptom with a threshold, e.g., 100 lm. Consequently, the range d of Y is defined

as being greater than or equal to this threshold. This allows the oNCS system to alert a caregiver when the light intensity is too high at the location of this patient.

Given the context dependencies, the filter rule generation algorithm works as follows. A concept `ContextToQuery` is defined with the necessary and sufficient condition that the ontological instance must have the relationship X `hasContextDependency` Y and $c \equiv true$. Every time the context changes, the context dependencies are investigated by examining the membership of instances to the `ContextToQuery` concept through ontological inferencing. Once the membership is determined, the construction of the filter rules is straightforward. For example, consider that the oNCS system is alerted that the patient in room 101 is diagnosed with a concussion. The domain specific ontology of the oNCS system contains the knowledge that patients with a concussion have light sensitivity as a possible symptom. Consequently, the condition c of the context dependency holds and the following filter rule is generated:

Event **and** hasContext **some** (LightIntensity
and (hasValue **some** float[>=100])
and (isObservationOf **some**
(hasId **some** hasID 'L101')))

This filter rule disseminates light intensity observations in the room of the patient. The published context matched with this filter rule must be a `LightIntensity` measurement. Moreover, its current value must be of type float and higher or equal than 100. Finally, the observation must be observed by a system with ID 'L101'. This is the ID of the light sensor in room 101. As previously indicated, this kind of static information is stored in databases, which can be queried by the applications. Note that the first example of Section C.3.4 matches with this filter rule. More information about this filter rule generation algorithm can be found in [5].

C.4 Implementation details & results

Semantic reasoning and ontologies were adopted in this research to facilitate the exploitation and integration of heterogeneous context information delivered by the devices in an ambient-intelligent patient room. However, it is widely known that special attention should be paid to the number of instances in the ontology as this adversely affects the reasoning performance [10]. Consequently, if all the data from the ambient-intelligent environment would be delivered directly to the ontology-based applications, their performance would degrade drastically. By using the SCB filters can be defined such that the applications only receive relevant context information. Moreover, the SCB allows composing complex applications from a set of smaller applications, which perform specific reasoning tasks in parallel and forward their conclusions through the SCB.

Consequently, the effectiveness of the rules can be measured by the reduction in the amount of information that is forwarded to the components as it is important that only the necessary context information is received by the ontology-based components. However, the correctness of the rules is another important performance metric. The correctness is influenced by the amount of data that was wrongfully filtered. It is important not to filter too much information, as lack of information about the context might cause components to take incorrect actions or no action at all. The goal is to increase the effectiveness, while maintaining the correctness. Consequently, the goal is to maximize the amount of data that is not forwarded, while ensuring that all the information that influences the actions of components is correctly forwarded.

To evaluate the performance and benefits of the SCB, a prototype of the oNCS system, extended with the Localization and Home Automation Component, was implemented and tested in the Patient Room of the Future (PRoF). PRoF is an intelligent patient room, realized in Belgium, aimed to make a patient feel more like home by putting the patient and his needs first. For the prototype, PRoF was equipped with a TMote Sky sensor board, which contains a light, temperature and humidity sensor. Moreover, each patient and staff member carries an RFID tag to track his/her location. Finally, each patient wears a bracelet that monitors the patient's body temperature. Table C.1 gives an overview of the amount of data generated by these sensors.

The prototype is able to realize the example detailed in Section C.1.1. Figure C.4 visualizes the first three steps of this scenario. First, the components register filter rules with the SCB to receive context information they are interested in, independent of the current context. Second, the applications work together to detect the presence of a nurse in a patient's room and turn on the light at the appropriate level. Third, the applications are alerted that a particular patient has a concussion. Consequently, the oNCS system registers a filter rule to receive specific context information pertaining to the light intensity in this patient's room. Next, a visitor enters the room, which causes the light to be automatically turned on to a low level as this patient has a concussion. The actions taken to realize this are similar to step 2. However, it remains possible for people to overrule the system and turn up the light in the room. An event similar to the last event indicated in step 2 is published on the SCB. However, because of the new filter rule, this event is not only forwarded to the Home Automation Component, but also to the oNCS system. The Home Automation Component concludes that it cannot adjust the light level as it has been overruled. However, the oNCS system also reasons on this event and alerts a nurse of the situation.

In this scenario, the SCB enables filtering a large amount of the data from Table C.1. As the Home Automation Component is only interested in sensor observations about the light intensity, the external temperature and the humidity, 54% of

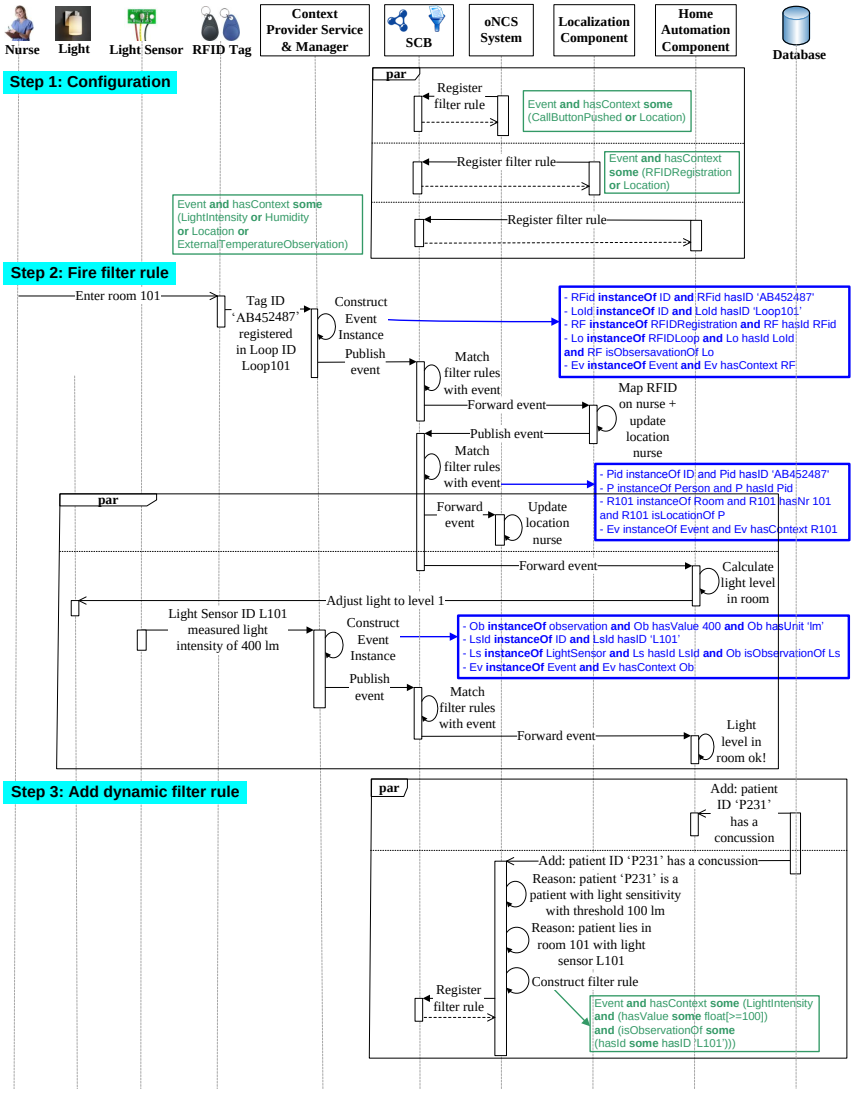


Figure C.4: Sequence diagram of the first three steps of the implemented scenario: Step 1: Configuration, Step 2: Turn on light when nurse enters the room, Step 3: Register a dynamic filter rule.

the generated sensor data is not forwarded to this component. Similarly, the Localization Component is only interested in RFID tag data, such that the amount of forwarded data is reduced by 77%. By taking advantage of the dynamic filter rule generation, none of the light intensity observations are forwarded to the oNCS system while none of the patients in the department have light sensitivity symp-

Table C.1: Amount of data generated by the sensors in a department with 20 rooms, 30 patients and 10 staff members.

Sensor	Nr. of sensors	Nr. of observations per sensor	Total nr. of observations per hour
Light	1/room	1/sec	72,000
Temperature	1/room	1/sec	72,000
Humidity	1/room	1/sec	72,000
RFID tag	1/person	1/sec	144,000
Body temperature	1/patient	1/sec	108,000

toms. When the department does have such a patient, only 8.3% of all the light intensity observations are forwarded to the oNCS system, assuming that it takes the nurse on average 5 minutes to respond to the context call, generated by the oNCS system because the light intensity is too high in the room. This way, the filter rule generation process allows dynamically adapting the amount of data that is filtered based on the context. It can be noted that both the oNCS and Home Automation Component never receive RFID tag observations anymore. They depend on the Localization Component, which processes these observations and publishes the resulting location information. However, these location updates are far less frequent than the RFID tag observations as only significantly changed locations, e.g., staff member is in another room, are published.

The prototype was built in Java, based on the Pellet OWL 2 reasoner and OWL-API. The evaluations were performed using the continuous care core ontologies needed to model the scenario, which consist of 142 classes, 42 object properties, 21 data properties and 556 asserted axioms.

The context dissemination of the SCB consists of two steps. First, filter rules are created by the applications and registered with the Context Disseminator. However, as filter rule registrations only occur occasionally, the introduced delay is negligible. Second, context is published to the SCB, matched with the filter rules and forwarded to the appropriate application if a match is found. The publication of context information happens frequently, as illustrated by Table C.1. As such, it is important that events are matched with filter rules swiftly. The performance of filtering, matching and forwarding an event with the SCB, is visualized in Figure C.5. The standard deviation is on average 4.65, 4.88 and 4.62 ms when respectively 0%, 50% and 100% of the filter rules match with the published event. The graph shows that the processing time is linear in the amount of filter rules and that the influence of the percentage of filter rules that match the event is negligible. Note that for the described scenario, which contains at most 4 filter rules,

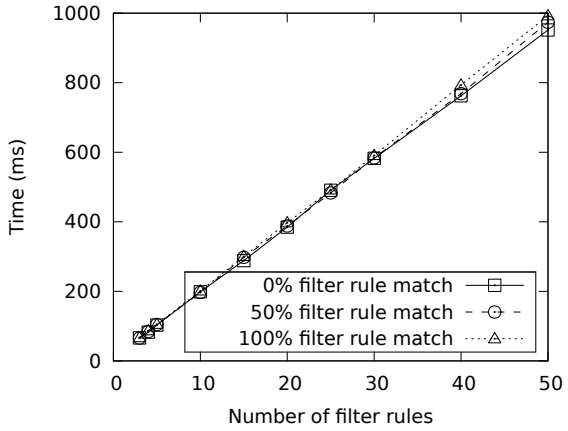


Figure C.5: Average reasoning time needed to publish, filter and forward one event as a function of the number of filter rules and the percentage of filter rules that match with this event, averaged over 30 iterations

an event is processed in on average 82.67 ms. However, the performance quickly decreases. For 50 filter rules, it takes on average 1 second to process an event. However, in an actual large-scale deployment, the SCB will most likely not correspond to a centralized physical process, but will be a virtual substrate distributed across the network. Consequently, the filter rules are distributed across the different dissemination instances, processing the event in parallel. The number of filter rules present in each context dissemination instance will thus be relatively low, which increases the throughput of the SCB significantly. Furthermore, it has been shown by Famaey, et al. [5] that the performance of the SCB depends significantly on the complexity of the used core ontologies. It is therefore important to achieve a good balance between the desired performance and the intelligence of the SCB, i.e., the expressiveness of the filter rules.

To optimize the correctness of the filter rules it is important to continuously evaluate the platform with the domain experts, both during the development process and after the system has been deployed. To make sure the system correctly reflects the continuous work processes of the caregivers, domain experts were constantly involved during the design and development process of the ACCIO Platform and used components. For example, observations were performed to investigate which information is taken into account to perform a certain task or make a decision and a participatory methodology was used to develop the ontology and the accompanying algorithms. Moreover, the developed system has been thoroughly evaluated with the various stakeholders by allowing them to play scenarios in the PRoF. When the system is deployed, techniques will also be used to contin-

uously monitor and improve the correctness, for example, by examining situations in which the decision of the platform was overruled by the users or by gathering intermediate feedback.

C.5 Conclusion

In this appendix, a context-aware and pervasive framework was presented, capable of disseminating the important care data for the different technologies available in an ambient-aware patient room by using a continuous care ontology. Depending on the context, the applications can register new filter rules on the fly to receive context information which is important to them at that moment. This way, the amount of data which needs to be processed by the applications is significantly decreased, which improves their performance. It was shown, that the delay introduced by the context dissemination is linear in the amount of filter rules and is negligible when 10 or less filter rules are registered. Moreover, the platform supports the composition of complex applications from a set of smaller applications. These perform specific reasoning tasks in parallel and notify their conclusions to other applications, which have expressed an interest in this kind of information. The strength of the platform is dependent on the correctness and completeness of the used ontologies. Therefore, a participatory ontology engineering methodology was presented which promotes user participation, while not overloading the involved stakeholders.

Acknowledgment

We would like to acknowledge that part of this research was supported by the IBBT Project ACCIO co-funded by the IWT, the IBBT and the following partners: Televic NV, Boone NV, Dominiek Savio Instituut and In-Ham. Part of this research was performed in the PRoF1.0 demo room, which was realized in Poperinge, Belgium by the PRoF consortium in 2010. F. Ongenae and J. Famaey would like to thank the IWT for financial support through their Ph.D. grant. S. Latré is funded by the Fund for Scientific Research Flanders (FWO-Vlaanderen). We thank all the participants in ACCIO for their valuable contribution to the project.

References

[1] M. Tentori, D. Segura, and J. Favela. *Monitoring hospital patients using ambient displays*. In Mobile Health Solutions for Biomedical Applications, pages 143–158. Medical Information Science Reference, 2009. doi:10.4018/978-1-60566-332-6.ch008.

[2] F. Ongenae, L. Bleumers, N. Sulmon, M. Verstraete, A. Jacobs, M. Van Gils, A. Ackaert, S. De Zutter, P. Verhoeve, and F. De Turck. *Participatory design of a continuous care ontology: Towards a user-driven ontology engineering methodology*. In International Conference on Knowledge Engineering and Ontology Development (KEOD), 2011.

[3] J. Hong, E. Suh, and S. Kim. *Context-aware systems: A literature review and classification*. Expert Systems with Applications, 36(4):8509–8522, 2009. doi:10.1016/j.eswa.2008.10.071.

[4] M. Zuccalà. *A survey of structured P2P systems for RDF data storage and retrieval*. Transactions on Large-Scale Data- and Knowledge-Centered Systems, 6790:20–55, 2011. doi:10.1007/978-3-642-23074-5_2.

[5] J. Famaey, S. Latré, J. Strassner, and F. De Turck. *Semantic context dissemination and service matchmaking in future network management*. International Journal of Network and Systems Management, 2011. doi:10.1002/nem.805.

[6] N. Bricon-Souf and C. R. Newman. *Context awareness in health care: A review*. International Journal of Medical Informatics, 76(1):2–12, 2007. doi:10.1016/j.ijmedinf.2006.01.003.

[7] F. Paganelli and D. Giuli. *An ontology-based system for context-aware and configurable services to support home-based continuous care*. IEEE Transactions on Information Technology in Biomedicine, 15(2):324–333, 2011. doi:10.1109/TITB.2010.2091649.

[8] F. Ongenae, D. Myny, T. Dhaene, T. Defloor, D. Van Goubergen, P. Verhoeve, J. Decruyenaere, and F. De Turck. *An ontology-based nurse call management system (oNCS) with probabilistic priority assessment*. BMC Health Services Research, 11(26), 2011. doi:10.1186/1472-6963-11-26.

[9] E. Simperl, M. Mochol, and T. Bürger. *Achieving maturity: the state of practice in ontology engineering in 2009*. International Journal of Computer Science and Applications, 7(1):45–65, 2010. doi:10.1007/978-3-642-05151-7_17.

[10] L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. *Towards a complete OWL ontology benchmark*. In 3rd European Semantic Web Conference (ESWC), pages 125–139, 2006. doi:10.1007/11762256_12.