iNegotiate: A Distributed SLA Negotiation System for Dynamic IoT Environments

by

Fan Li, M.Sc, B.Eng

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Doctor of Philosophy

University of Dublin, Trinity College

May 2020
Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

I, the undersigned, agree to deposit this thesis in the University’s open access institutional repository or allow the library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

______________________________

Fan Li

November 12, 2020
Acknowledgments

This thesis was made possible with the support from my supervisor, my colleagues, my friends and my families. First and the foremost, I would like to thank my supervisor, Professor Siobhán Clarke, for giving me this opportunity to do the Ph.D. at Trinity College Dublin, and for her guidance, expertise, encouragement, understanding and all the valuable advice during the research process. I would like to thank my colleagues in the SURF project: Christian Cabrera, Andrei Palade and Gary White. I learned a lot from you, and thank you for your help and support for the past four years. Thanks to Vivek Nallur and Nanxi Chen, for your help and advice in the incipient phase of this research. Thanks a lot to all my family and friends. In particular, I would like to thank my mother Xiaoling Jin and my father Xiaodong Li, for their support, love, and encouragement I receive everyday. I dedicate this thesis to them. I would like to thank my friend Qiang Ji, for his help, support, and understanding. Finally, I am grateful for the financial support I received from Science Foundation Ireland, which has offered me the opportunity to do the research in Ireland.

Fan Li

University of Dublin, Trinity College
May 2020
Abstract

The Internet of Things (IoT) envisions a large number of physical devices connecting over the Internet at an unparalleled rate, to enable ubiquitous and pervasive computing scenarios. By adopting the service-oriented architecture paradigm, the devices’ functional capabilities can be abstracted as IoT services and provided to application tasks on demand to create scalable, adaptable, and flexible IoT applications. For mission-critical IoT applications that have strict demands on the quality of service (QoS), a Service Level Agreement (SLA) can be used as a contract to specify the obligations and guarantees of involved parties for a particular service. By adopting an SLA monitoring and billing mechanism, service performance can be measured at runtime and compensation imposed when a contract is violated during the service execution stage.

An SLA is needed to enable demand-driven service provision within QoS constraints, while SLA negotiation is needed to create a valid SLA before actual service delivery. To date, SLA management for the IoT environment has focused on SLA modeling, while research on automatic run-time SLA negotiation is limited to a centralized cloud-based mediator platform. This is insufficient to accommodate IoT domain-specific characteristics such as distributed heterogeneous resources, scale and dynamics in the environment. In addition, when negotiation parties are mobile, timeout failures during the negotiation stage may increase, which further reduces the chance of finding a mutually acceptable solution.

This thesis presents a distributed negotiation system that dynamically negotiates with multiple service providers on behalf of service consumers, named iNegotiate. In an IoT environment with distributed, heterogeneous negotiation parties, semantic interoperability is
Abstract

required for these parties to understand each other. This work formalizes service information by proposing the WIoT-SLA ontology, which is an SLA ontology of IoT services built on two prominent web service SLA specifications: WS-Agreement and WSLA. In addition, in a large-scale environment where consumers are likely to have no prior experience with the surrounding service providers, negotiation with multiple service providers without a trust evaluation may be time-consuming and risky. This work proposes an experience-based trust model to identify trustworthy candidate service providers that have the potential to satisfy a consumer’s requirements, even before negotiating with them. Considering the scale and dynamics of the environment, this thesis further proposes a three-stage negotiation model for automatic SLA negotiation. The first is the pre-negotiation stage, at which a hierarchical overlay network is automatically created to organize service information and control message flows. The second is the negotiation stage, at which a negotiation strategy based on WS-Agreement Negotiation specification is designed to make decisions after receiving a negotiation offer from a service provider. To balance the success rate and negotiated utility, the strategy has two negotiation tactics targeting different negotiation scenarios: a context-based tactic that reflects a service consumer’s negotiation preference, and an artificial bee colony (ABC) algorithm-based tactic when the consumer’s negotiation preference is unknown. The third stage is the post-negotiation stage at which the best negotiated solution is selected and returned to consumers. The thesis measures system performance using a set of simulated experiments. Evaluations of the service match-making using the WIoT-SLA ontology measure matchmaking precision, recall, and accuracy by simulating various services that have different SLA terms. Evaluations of the negotiation strategy measure the success rate and negotiation utility by simulating different bilateral negotiation scenarios. Evaluations of the trust-based candidate selection measure the success rate, negotiation utility, and SLA compliance by simulating different types of service providers based on a real IoT dataset. Evaluations of the distributed negotiation model have been carried out on the Simonstrator simulator under different network densities and various dynamic conditions. The evaluation metrics are the success rate, message delivery rate and the number of messages. The results demonstrate the feasibility, efficiency, and limitations of iNegotiate.
## Contents

Acknowledgments iii

Abstract iv

List of Tables xi

List of Figures xiii

List of Abbreviations xvii

List of Publications xix

Chapter 1 Introduction 1

1.1 SLA and SLA Negotiation in IoT Environments 1

1.2 Problem Definition and Challenges 5

1.3 Existing Solutions 8

1.3.1 Service Level Agreement Modeling 11

1.3.2 SLA Negotiation Strategy 12

1.3.3 SLA Negotiation Protocol 14

1.3.4 Research Gaps and Observations 15

1.4 Research Questions 18

1.5 Thesis Approach 19

1.6 Thesis Contribution 22

1.7 Thesis Scope 27
# Contents

1.8 Thesis Structure ......................................................... 28  
1.9 Chapter Summary ....................................................... 29

## Chapter 2 State of the Art

2.1 Service-oriented Computing .......................................... 31  
  2.1.1 Capacity-aware SOA .............................................. 32  
  2.1.2 Service-oriented computing for the IoT ...................... 33  
  2.1.3 Assessment ....................................................... 34  
2.2 Service Level Agreement Development .............................. 36  
  2.2.1 SLA Lifecycle .................................................... 36  
  2.2.2 Service Level Agreement Specifications ..................... 39  
  2.2.3 Service Abstraction in the IoT environment .................. 44  
  2.2.4 Smart Contract .................................................. 48  
  2.2.5 Assessment ....................................................... 50  
2.3 Automated SLA Negotiation ........................................... 52  
  2.3.1 SLA Negotiation Protocol ..................................... 52  
  2.3.2 Negotiation Strategy ............................................ 56  
  2.3.3 Assessment ....................................................... 58  
2.4 Selecting Negotiation Candidates .................................... 60  
  2.4.1 Methods for Service Provider Selection ...................... 60  
  2.4.2 Assessment ....................................................... 62  
2.5 Chapter Summary ....................................................... 62

## Chapter 3 Design

3.1 Design Objectives and Required Features ........................ 67  
3.2 System Environment .................................................. 69  
3.3 Design Decisions ...................................................... 70  
  3.3.1 WSAG-based SLA Modeling for IoT services ................. 70  
  3.3.2 Distributed SLA Negotiation Model ......................... 72  
  3.3.3 Trust-based Negotiation Candidates Selection ............... 74
3.3.4 Deadline-aware negotiation strategy for negotiations with incomplete information ........................................... 75

3.4 iNegotiate ................................................................. 76

3.4.1 SLA Ontology Model for IoT Services .......................... 77

3.4.2 Distributed SLA Negotiation Model ......................... 88

3.4.3 Trust-based Evaluation of Negotiation Candidates .......... 105

3.4.4 Deadline-aware strategy for IoT service negotiation ....... 113

3.5 Chapter Summary ..................................................... 126

Chapter 4 Implementation .............................................. 128

4.1 iNegotiate Architecture ............................................. 128

4.2 Implementation of iNegotiate Model ............................. 132

4.2.1 iNegotiate Data Model ........................................... 133

4.2.2 iNegotiate Class Diagram and Operations ................. 137

4.3 Chapter Summary ..................................................... 146

Chapter 5 Evaluation .................................................. 147

5.1 Evaluation Approach ................................................ 147

5.2 Evaluation of WIoT-SLA Template Match-making ............. 151

5.2.1 Performance Metrics ............................................. 151

5.2.2 Experimental Setup ............................................. 152

5.2.3 Results ............................................................. 154

5.3 Evaluation of Trust-based Candidates Selection Model ....... 155

5.3.1 Performance Metrics ............................................. 156

5.3.2 Baseline Approaches ............................................ 157

5.3.3 Experimental Setup ............................................. 158

5.3.4 Statistical Analysis ............................................. 160

5.3.5 Result ............................................................. 162

5.4 Evaluation of iNegotiate's Bilateral Negotiation Strategy .... 167

5.4.1 Performance Metrics ............................................. 167
Chapter 5 Evaluation of Distributed Negotiation using HNON .......................... 190

5.5.1 Performance Metrics ............................................................................. 190
5.5.2 Baseline Approaches .......................................................................... 191
5.5.3 Experimental Setup ............................................................................. 193
5.5.4 Statistical Analysis .............................................................................. 196
5.5.5 Result of Experiment 1: Comparison with the Baseline ...................... 197
5.5.6 Result of Experiment 2: iNegotiate Performance Analysis ................. 201
5.5.7 Evaluation Summary ........................................................................... 207

Chapter 6 Conclusion ....................................................................................... 210

6.1 Thesis Summary ....................................................................................... 210
6.2 Discussion ................................................................................................ 212
6.2.1 Thesis Contributions ........................................................................... 213
6.3 Limitations and Future Work .................................................................. 215
6.3.1 SLA Template Matchmaking and Knowledge Ontology Maintenance .. 215
6.3.2 Negotiation Task Allocation ................................................................. 217
6.3.3 Mobile Entity Management ................................................................. 217
6.3.4 Network Efficiency ............................................................................. 218
6.3.5 Composite Service Negotiation ........................................................... 219
6.3.6 Computational Efficiency ................................................................. 220
6.3.7 Fault Tolerance ................................................................................... 222
6.3.8 Security and Privacy .......................................................................... 223
6.3.9 Summary ............................................................................................ 225

Appendix A Rough Set Theory ........................................................................ 227
Appendix B  Operation Code of Messages  230

Appendix C  Synonymous Words in Study 1  233

Bibliography  234
## List of Tables

2.1 Literature Summary ................................................. 63

3.1 An Example of a Decision Table ................................. 109

5.1 Research Questions and Corresponding Evaluation Studies .......... 150
5.2 Hardware Configurations ........................................... 153
5.3 Dataset Statistics .................................................. 158
5.4 Provider Configurations ............................................. 159
5.5 Hardware Configurations ............................................. 161
5.6 Performance Overview .............................................. 164
5.7 Spearman’s Rank Correlation ....................................... 165
5.8 Kendall’s Tau-b coefficient ......................................... 165
5.9 Average Execution Time ............................................. 165
5.10 Configuration of Negotiation Constraints for Each Request .......... 170
5.11 Configuration of Service Providers Based on Requests ............... 171
5.12 Simulation Parameters of Experiment 1: Evaluation of UMI-based Strategy . 171
5.13 Simulation Parameters of Experiment 2: Evaluation of ABC-Based Strategy . 172
5.14 Different operations of FDP and \textit{iNegotiate} ....................... 192
5.15 Simulation Parameters of Experiment 1: Comparison with the Baseline .. 194
5.16 Simulation Parameters of Experiment 2: \textit{iNegotiate} Performance Analysis .. 195
5.17 Statistical Analysis on \textit{iNegotiate}’s performance .................. 201
B.1 Configuration Message and the Associated Operations .......................... 230
B.2 Template Registration Message and the Associated Operations ............. 231
B.3 Negotiation Request Message and the Associated Operations ............... 231
B.4 Negotiation Customization Message and the Associated Operations ........ 232
B.5 Negotiation Message and the Associated Operations ......................... 232
B.6 Mobile Entity Locating Message and the Associated Operations ............ 232
C.1 Synonymous Words List ............................................................... 233
# List of Figures

1.1 Interaction in an SOA using web services [Erl, 2005] .......................... 2
1.2 SLA management lifecycle [Keller and Ludwig, 2003] .......................... 3
1.3 Game relationship of web service negotiation [Chen et al., 2016] ............ 9
1.4 Configuration plan of bargaining [Hung et al., 2004] .......................... 9
1.5 Important concepts of SLA negotiation system .................................. 10
1.6 SLA negotiation in the IoT environment ............................................ 17
1.7 Negotiation gateways (GW) in an IoT environment .............................. 19
1.8 Ontology of iNegotiate framework ................................................... 23

2.1 Structure of the literature review ....................................................... 31
2.2 Capacity-aware SOA driven by agreements [Ludwig, 2006] ................. 32
2.3 The collaborative SOA-based IoT system [Guinard et al., 2010] ............ 34
2.4 Four-layered capacity-aware SOA structure for the IoT [Da Xu et al., 2014] 35
2.5 IoT service ontology [De et al., 2011] .............................................. 46
2.6 Kiviat diagram ................................................................................. 64

3.1 System environment ........................................................................... 69
3.2 iNegotiate - Design decisions map .................................................... 71
3.3 Conceptual system model of iNegotiate .............................................. 77
3.4 Automated SLA management based on ontology concepts .................. 78
3.5 Upper ontology of capacity-aware SOA-based IoT ............................ 79
3.6 WIoT-SLA structure ........................................................................ 80
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.7</td>
<td>Service description terms</td>
<td>82</td>
</tr>
<tr>
<td>3.8</td>
<td>An example of configuration items (sensing service)</td>
<td>83</td>
</tr>
<tr>
<td>3.9</td>
<td>WiO-T-SLA template structure</td>
<td>84</td>
</tr>
<tr>
<td>3.10</td>
<td>An example of creation constraints</td>
<td>86</td>
</tr>
<tr>
<td>3.11</td>
<td>Ontology of negotiation offer</td>
<td>87</td>
</tr>
<tr>
<td>3.12</td>
<td>Three-stage iNegotiate model</td>
<td>91</td>
</tr>
<tr>
<td>3.13</td>
<td>Hierarchical negotiation overlay network</td>
<td>92</td>
</tr>
<tr>
<td>3.14</td>
<td>Cross-layer message flow in HNON</td>
<td>93</td>
</tr>
<tr>
<td>3.15</td>
<td>An example of a sub-area</td>
<td>98</td>
</tr>
<tr>
<td>3.16</td>
<td>Processing of template registration messages</td>
<td>99</td>
</tr>
<tr>
<td>3.17</td>
<td>Processing of negotiation request messages</td>
<td>101</td>
</tr>
<tr>
<td>3.18</td>
<td>Processing of negotiation messages</td>
<td>102</td>
</tr>
<tr>
<td>3.19</td>
<td>Processing of mobile entity locating messages</td>
<td>103</td>
</tr>
<tr>
<td>3.20</td>
<td>An example of distributed SLA negotiation using HNON</td>
<td>104</td>
</tr>
<tr>
<td>3.21</td>
<td>Trust model with SLA negotiation and monitoring</td>
<td>106</td>
</tr>
<tr>
<td>3.22</td>
<td>Impact fading</td>
<td>107</td>
</tr>
<tr>
<td>3.23</td>
<td>Decision-making model</td>
<td>117</td>
</tr>
<tr>
<td>3.24</td>
<td>Utility function</td>
<td>118</td>
</tr>
<tr>
<td>3.25</td>
<td>Concession and tradeoff tactic in UMI</td>
<td>121</td>
</tr>
<tr>
<td>3.26</td>
<td>iNegotiate’s features compared to other approaches</td>
<td>127</td>
</tr>
<tr>
<td>4.1</td>
<td>iNegotiate - component diagram</td>
<td>129</td>
</tr>
<tr>
<td>4.2</td>
<td>iNegotiate prototype implemented in Simonstrator platform</td>
<td>133</td>
</tr>
<tr>
<td>4.3</td>
<td>iNegotiate - activity diagram</td>
<td>134</td>
</tr>
<tr>
<td>4.4</td>
<td>iNegotiate - messages</td>
<td>135</td>
</tr>
<tr>
<td>4.5</td>
<td>iNegotiate - class diagram</td>
<td>138</td>
</tr>
<tr>
<td>4.6</td>
<td>iNegotiate - controller allocation operation</td>
<td>139</td>
</tr>
<tr>
<td>4.7</td>
<td>iNegotiate - coordinator allocation operation</td>
<td>140</td>
</tr>
<tr>
<td>4.8</td>
<td>iNegotiate - template distribution operation</td>
<td>141</td>
</tr>
</tbody>
</table>
List of Figures

4.9 iNegotiate - negotiation task allocation operation .................................. 143
4.10 iNegotiate - negotiation operation ............................................................... 144
4.11 iNegotiate - mobile entity locating operation .............................................. 145

5.1 Precision, recall and accuracy of test case 3 ................................................. 154
5.2 Average processing time ............................................................................... 155
5.3 QoS Changing trends ..................................................................................... 159
5.4 Evaluation of service providers ..................................................................... 162
5.5 Average Execution Time in Milliseconds ..................................................... 166
5.6 CPU and Memory Usage on Resource-constrained Devices ...................... 167
5.7 Negotiation performance in the resource-limited environment .................... 174
5.8 Utility-changing trend in the resource-limited environment ......................... 175
5.9 Negotiation performance in the resource-rich environment ......................... 176
5.10 Utility-changing trend in the resource-rich environment ............................. 177
5.11 Multiple comparison of ranked means (resource-limited environment) ...... 178
5.12 Multiple comparison of ranked means (resource-rich environment) .......... 179
5.13 Average execution time on different devices ............................................... 181
5.14 Negotiation performance in the resource-limited environment .................... 182
5.15 Utility-changing trend in the resource-limited environment ......................... 183
5.16 Negotiation performance in the resource-rich environment ......................... 184
5.17 Utility-changing trend in the resource-rich environment ............................. 185
5.18 Multiple comparison of ranked means (resource-limited environment) ...... 186
5.19 Multiple comparison of ranked means (resource-rich environment) .......... 187
5.20 Average execution time on different devices ............................................... 189
5.21 System performance with variable number of gateways/providers ............. 198
5.22 System performance with variable number of gateways ............................. 201
5.23 System performance with variable number of Internet-connected gateways .. 202
5.24 System performance under different mobility scenarios .......................... 203
5.25 An inaccurate template registration scenario .............................................. 204
5.26 An invalid template registration scenario ........................................ 205
5.27 $iNegotiate$ performance under different movement scenarios .............. 206
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>Artificial Bee Colony Optimisation.</td>
</tr>
<tr>
<td>APIs</td>
<td>Application Programming Interface.</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithms.</td>
</tr>
<tr>
<td>HetNets</td>
<td>Heterogeneous Networks.</td>
</tr>
<tr>
<td>HNON</td>
<td>Hierarchical Negotiation Overlay Network.</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure-as-a-Service.</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things.</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation.</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network.</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short-term Memory.</td>
</tr>
<tr>
<td>MANETs</td>
<td>Mobile Ad-hoc Networks.</td>
</tr>
<tr>
<td>MIMO</td>
<td>Multiple-Input–Multiple-Output.</td>
</tr>
<tr>
<td>OWL-S</td>
<td>Ontology Web Language for Services.</td>
</tr>
<tr>
<td>QoD</td>
<td>Quality of Data.</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service.</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement.</td>
</tr>
<tr>
<td>SLOs</td>
<td>service level objectives.</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture.</td>
</tr>
<tr>
<td>SoC</td>
<td>System on a Chip.</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td><strong>WLAN</strong></td>
<td>Wireless Local Area Network.</td>
</tr>
<tr>
<td><strong>WSAG</strong></td>
<td>WS-Agreement Specification.</td>
</tr>
<tr>
<td><strong>WSAN</strong></td>
<td>WS-Agreement Negotiation Specification.</td>
</tr>
<tr>
<td><strong>WSN</strong></td>
<td>Wireless Sensor Network.</td>
</tr>
</tbody>
</table>
List of Publications

This thesis is based on the following peer-reviewed publications:


Parts of this work have been submitted to the following publications:


Parts of this work have appeared in the following peer-reviewed publications:


Chapter 1

Introduction

1.1 SLA and SLA Negotiation in IoT Environments

The Internet of Things (IoT) envisions a large number of smart objects, wirelessly connecting over the Internet at an unparalleled rate to enable ubiquitous and pervasive computing scenarios. The possible massive number of interactions between the objects and the physical world enable the development of cyber-physical systems, from which new IoT application scenarios can emerge such as smart cities, health care, infrastructure monitoring, and automotive applications [Marino et al., 2019]. By adopting the Service Oriented Architecture (SOA) paradigm, the resources and computation capabilities provided by these smart objects can be abstracted as services [Karnouskos et al., 2010, Mingozzi et al., 2014], which is achieved by providing a well-defined interface for each service that focuses on business process and hides the complexity of heterogeneous devices [Shelby et al., 2014]. In a Service-oriented Computing environment, services represent the main blocks for building applications. Figure 1.1 shows the basic typical interactions between organizations following the SOA paradigm. The core functionalities specified in SOAs such as service registry, service discovery, service composition and service access enable the development of flexible demand-driven IoT applications irrespective of services’ geographic boundaries [Issarny et al., 2016].

For mission-critical IoT applications in transportation, health care, and emergency response domains, stakeholders require stringent Quality of Service (QoS) demands rather
Figure 1.1: Interaction in an SOA using web services [Erl, 2005]

than "best effort" services [Swiatek and Rucinski, 2013]. Traditionally, such applications have relied on a Service Level Agreement (SLA), which is a contract-like concept that formalizes the obligations and the guarantees of involved parties for a particular service, as well as penalties for the provider in case the SLA is violated during the service execution stage [Ludwig et al., 2003]. Once an SLA is created, it is monitored by a third-party that is trusted by both the consumer and provider to measure the run-time service performance against the criterion specified in the SLA. In an open market, service providers who support SLAs can have a comprehensive competitive edge in terms of QoS-aware service management [Kazmi et al., 2017], service customization [Erfattry and Layzell, 2004], optimized resource allocation [Singh and Viniotis, 2016], and trustworthiness [Qi et al., 2014]. In other words, the SLA management mechanism helps service providers reduce the risks of degradation in service quality. A good reputation in turn increases profit and benefits the provider’s business goals. From the perspective of a service customer, choosing SLA-supported services is helpful to guarantee expected service performance and enhance its level of satisfaction [Zheng et al., 2010]. As Figure 1.2 shows, the generic activities performed during the lifecycle of an SLA can be classified into five phases [Keller and Ludwig, 2003]:

- **SLA negotiation and establishment**: Trading parties negotiate on functional and non-functional service properties, and create an SLA for deployment based on the negotiated result.

- **SLA deployment**: System configuration (e.g., service negotiation and monitoring con-
Figure 1.2: SLA management lifecycle [Keller and Ludwig, 2003]

configuration) is generated according to the validated SLA.

- Service level measurement and reporting: The QoS parameters (e.g., responsiveness, availability, etc.) are measured by monitoring the corresponding metrics, and compared against the guaranteed quality levels. Once an SLA violation is detected, the management system is notified to take further actions.

- Corrective management actions: SLA violations may cause the termination of service provisioning. To achieve service continuity, corrective actions such as SLA renegotiation and provider notification can be executed to resolve detected violations.

- SLA termination: The SLA can be terminated when pre-defined events occur (e.g., SLA violation) or the SLA’s expiration date has been reached.

Compared to cloud services, service provisioning in the IoT environment needs to cope with the mobility and intermittent availability of devices, which may exhibit flexible quality levels and pricing options [Grubitzsch et al., 2017]. If we assume consumers have no prior knowledge about the available resources in the environment, a provider selection mechanism that supports finding suitable trustworthy candidates and negotiates with them to tailor service properties is necessary before SLA creation [Marino et al., 2019]. In the context of
a demand-driven service provisioning, the possible conflicts between trading parties can be resolved through the SLA negotiation process where both parties dynamically express their own demands and preferences to arrive at a consensus before the actual service delivery [Saravanan and Rajaram, 2015]. The negotiated result is specified in the SLA, which will be enforced and tracked for compliance during the service execution stage [Bianco et al., 2008].

Research on service negotiation emerged from the requirements of market-based service provisioning [Holloway, 2017]. For example, cloud computing attempts to dynamically reconfigure virtualized resources in response to variable loads [Wieder et al., 2011], while grid computing aims to dynamically deploy computational resources in a distributed system as required to solve complex problems [Haberland et al., 2012]. As the technologies advanced, negotiation mechanisms were also discussed in Mobile Ad-hoc Networks (MANETs), where virtual cash rewarding can be used as a stimulation for autonomous and self-interested mobile nodes to cooperate in routing and packet forwarding [Janzadeh et al., 2009, Li and Shen, 2011]. Selfish nodes pursuing their own interests are motivated to dynamically adjust their bids according to environmental factors (i.e., remaining energy, computational resources, etc) for maximum profits, which makes negotiation an incentive mechanism that encourages nodes to announce individual information and coordinate with each other to achieve a global beneficial agreement [Yang et al., 2011, Zhao et al., 2011, Chen et al., 2011b]. With the popularity of smartphones, e-commerce environments have been diversified from the traditional wired networks to pervasive computing environments [Park and Yang, 2008], where buyers retrieve information about their interested products by interacting with servers deployed at different retail locations using mobile devices [Kurkovsky and Harihar, 2006]. To increase the joint profits of trading parties, automated multilateral negotiations can be carried out to cope with the profit conflicts with regards to multiple issues [Park and Yang, 2008]. With the development of System on a Chip (SoC) technologies, many embedded devices are qualified to run self-operation algorithms and make decisions dynamically depending on the context information [Sahni et al., 2017]. Automatic negotiation mechanisms have been adopted in the IoT environment for devices reaching an agreement on the sequence of actions when they have conflicting requirements on shared resources (e.g., smart traffic control) [Bui and Jung,
Although the importance of SLA management in realizing the full potential of the IoT domain has been identified [Papadopoulos et al., 2017, Mubeen et al., 2017, Perera, 2017, Marino et al., 2019], there has been little work to date on the related topics [Palade et al., 2018, Balint and Truong, 2017]. To our best knowledge, current research focuses on SLA modeling for Wireless Sensor Network (WSN) or IoT applications [Alqahtani et al., 2019, Gaillard et al., 2014b], while existing SLA negotiation approaches are limited to centralized cloud-based mediator platforms [Mišura and Žagar, 2017, Casola et al., 2013, Hayat et al., 2019], which may not work well when the corresponding network infrastructure is severely damaged by unforeseen events or disasters [Le and Kwon, 2017]. Also, the proposed solution is insufficient to accommodate the IoT domain-specific characteristics such as distributed heterogeneous resources, scale and dynamics in the environment. Generally, a negotiation process should have a good balance between the success ratio and negotiation efficiency under a short time constraint. However, this is challenging in the IoT domain due to the communication issues introduced by distributed mobile negotiation entities [Li and Clarke, 2018], the large number of third-party service providers available in the environment [Li et al., 2019a], the unknown negotiation constraints and preferences that the negotiation parties hold for business privacy reasons [Zheng et al., 2014], and the time-varying nature of IoT service qualities [Shao et al., 2019] that may affect the negotiation constraints of service providers at different times.

1.2 Problem Definition and Challenges

To illustrate the opportunities and challenges of SLA negotiation in a dynamic IoT environment, we firstly describe a simple use case. Consider a smart city environment where multiple third-party service providers deploy devices that measure the surrounding environment and provide real-time data through a wide range of services such as traffic flow monitoring, public transport services, noise detection, particle concentration monitoring, etc.
These services are developed using various approaches and technologies (e.g., web services, WSN services, and autonomous services\(^1\)), which are different in terms of service type, price, location, resource configurations, and QoS properties. These services can be best-effort services or SLA-supported services that are delivered on-demand by negotiating an SLA with the corresponding service provider. Since human intervention is infeasible to manage such a large number of services, a middleware can be deployed in different locations to provide the necessary functionalities such as service registry, service discovery and SLA negotiation. Service providers advertise their services and negotiation information to the middleware so that they can be contacted when their offerings match consumers’ requests. Service providers can join or leave the network at any time without advance notice after registering their services, or keep moving in the environment while waiting for negotiation requests.

Imagine Alice, who suffers from asthma, is on a business trip to another city. To decrease the likelihood of asthma relapse, instead of buying some expensive sensors herself and deploying them in her haunts, Alice prefers to rent a real-time particle concentration monitor service and a hazardous gas detection service that are provided with high accuracy, high availability as well as an acceptable budget. Since Alice has no clue about the available resources in the environment, she submits the request through the mobile application to the middleware. The middleware looks up suitable services according to the functional and non-functional requirements specified in the request, negotiates with corresponding service providers to find proposals that guarantee the most satisfying tradeoffs, and responds to Alice with the best option. If the solution is verified by Alice, an SLA which specifies the service properties with negotiated obligations and guarantees is created, and the service run-time state will be monitored once the service is delivered to Alice. During the service provisioning time, Alice can send a renegotiation request to the middleware if she wants to change the request. The middleware tries to contact the service provider to initiate a renegotiate process. If Alice’s new request is acceptable to the service provider, the current SLA changes to the “completed”

---

\(^1\)Autonomous services are defined as the services that providers can autonomously decide when to offer and withdraw without advance notice, and they are likely to be mobile. For instance, a traffic monitoring service provided by a smart dashcam on a taxi can be online or offline at any time according to the configuration set by the driver [Cabrera et al., 2017].
state, and a new SLA will be created based on the renegotiated results [Andrieux et al., 2007]. Otherwise, Alice can choose to terminate the current SLA by paying the penalty and submit a new negotiation request. From the above motivating scenario, the challenges of automatic SLA negotiation emerging from characteristics of IoT environments are generalized as follows:

- **Challenge 1: Large-scale environment and distributed negotiation entities.** Globally, the IoT will be an ultra large-scale network containing a billion or even trillion nodes distributed in different locations [Gavras et al., 2007]. The volume, variety, and velocity of data that the IoT generates make a cloud-based system inefficient in terms of minimizing latency, conserving network bandwidth and avoiding system failures [Computing et al., 2016]. Also, a potentially huge number of IoT devices are likely to engage in service provisioning, which implies the presence of multiple third-party service providers offering similar functionalities. To avoid unnecessary interactions and increase negotiation efficiency, the SLA negotiation process should efficiently identify the trustworthy service providers to negotiate with based on the user’s request [Li et al., 2019a].

- **Challenge 2: Highly dynamic.** The dynamic nature of the IoT environment derives from service providers’ mobility, unpredictable workload, unstable wireless network conditions, and device’s malfunction. This brings new challenges to the SLA negotiation process such as frequent disconnection between negotiating parties [Li and Clarke, 2018], insufficient awareness of local context (e.g., service location), and possible network congestion caused by multiple spontaneous interactions amongst devices [Li et al., 2019b]. To reduce the overhead and increase the success ratio, the SLA negotiation process should be time-constrained and conducted at places close to the candidate service providers.

- **Challenge 3: Deep heterogeneity of resources:** The IoT comprises a wide range of devices that are various in terms of device features, resource capabilities, communication technologies and service properties [Zanella et al., 2014]. Different from cloud services, IoT services are likely to have more negotiable attributes that are not limited
to QoS parameters. For example, to guarantee the functional integrity of the network, sensor nodes in WSNs with the same functionality are usually redundantly deployed, which implies that the service coverage can be affected by the sleep schedule of sensor nodes [Gupta et al., 2016]. Combined with the existence of mobile sensors, it is possible to tailor the service location based on a user’s expectation by adjusting a resource management mechanism. To achieve semantic interoperability and reduce the ambiguity in automating negotiation and monitoring activities, the SLA needs to be described with a machine-readable structure [Alqahtani et al., 2019]. Current standard SLA languages for cloud services and web services are insufficient to capture IoT domain-specific service properties. The SLA negotiator process should be able to draft SLAs with respect to describing IoT services from functional and non-functional perspective.

1.3 Existing Solutions

Due to the varying QoS demands and business objectives in an open market, IoT services can be provided in a demand-driven way and consumed based on the pay-as-you-go model [Kantarci and Mouftah, 2015]. Negotiation is a possible way to deliver the same service to different users with agreed service qualities and costs [Elfatatry and Layzell, 2004]. Normally, service consumers and providers have conflicting interests. For example, a consumer hopes to obtain a service with a lower price but higher availability, whereas the provider attempts to offer the service with a higher price but lower availability to increase business profit. The goal of SLA negotiation is to find the best possible solution that satisfies the minimum expectations of both parties [Chen et al., 2016]. Figure 1.3 shows the game relationships between a service provider and a consumer during SLA negotiation. The term game refers to the process in which negotiation participants have to make specific decisions that have mutual and possibly conflicting consequences [Felegyhazi and Hubaux, 2006]. During the negotiation process, both rational negotiation parties try to maximize their own revenue but in the meanwhile, adapt their preference and constraints through the bilateral bargaining process to reach an agreement within a reasonable period of time [Zheng et al., 2010]. Figure 1.4 shows a general
Once a negotiation process is initialized, the negotiation entity definition is triggered by a “session started” event, which generates at least one negotiation plan. Then the initial contact is performed to finalize negotiation context with the opponent. The successful handshake will trigger the sequential offer exchange between negotiation parties. In each round, the received offers are evaluated by participants using their pre-defined negotiation strategies. Based on the evaluation result, negotiation parties can choose to make concessions to propose a revised counteroffer, agree with the received offer, or quit the negotiation. The latter two events also trigger the termination of the whole bargaining process.

Under the loosely coupled computing environment of SOA, consumers may bind to certain services provided by different service providers according to their requests. To help service
consumers identify the candidates who are likely to match their business goals under the QoS constraints and budget limitation among the vast number of market participants, an extra step before the bargaining process is required in the SLA negotiation, which is publishing a document that specifies the expected service properties of the service provider to a unified registration system, such as UDDI [Ding and Zhao, 2012]. To make the document reciprocally understandable for both service providers and consumers, a standard specification can be used to formalize the document structure. WS-Agreement is an example specification regulating that services can be advertised in the form of agreement templates, which guide consumers in the process of creating valid offers and the final agreement [Ludwig et al., 2003, Hasselmeyer et al., 2007]. With the agreement template registration mechanism, an automatic match-making process can be triggered when a request comes, which iterates through all published agreement templates to check the similarity between each template and the request [Redl et al., 2012]. The template with the highest matching degree is chosen as the optimal candidate service for SLA negotiation.

As Figure 1.5 shows, the automated SLA negotiation mechanism described above implies three important concepts: negotiation objects, negotiation protocol and negotiation strategy [Jennings et al., 2001]. The negotiation object contains the set of issues over which agreement must be reached, representing various parameters along with their respective domain values.
(e.g., price range) [Venticinque et al., 2010]. In the SLA negotiation scenario, the negotiation objects can be modeled by SLA and SLA template ontologies [Li et al., 2019a]. The negotiation protocol defines the type and format of information exchanged during a negotiation process, as well as the message interaction rules [Yao and Ma, 2008]. The negotiation strategy is a mathematical model consisting of a decision-making algorithm that evaluates the received offers [Zheng et al., 2014] and a set of negotiation tactics that participants employ to generate counteroffers according to their business objectives [Faratin et al., 1998]. Considering the possible large latency and massive interactions during a bilateral negotiation process, negotiation with a service provider that has very little chance to reach an agreement not only wastes time but also introduces extra traffic that makes no contribution to the negotiation utility. A strategy that selects candidates before attempting to negotiate with them is required to improve system performance in an SOA-based environment where there are a large number of third-party service providers [Silva et al., 2012, Tserpes et al., 2012].

1.3.1 Service Level Agreement Modeling

To generate coherent and adaptable SLA documents and overcome the semantic heterogeneity between unfamiliar negotiation entities, SLA ontology definition and mapping are commonly used solutions for automatic (re-)negotiation. For example, SLA negotiation can be regarded as an alignment of ontologies that semantically represent the cloud service context, consumers’ requests and providers’ offers [Labidi et al., 2017]. By combining the ontology mapping algorithm and the inference rules to reason about the contextual changes of cloud services, a proactive SLA renegotiation can be conducted to adjust the established SLA without the suspension of the service provisioning and the sudden termination of the SLA [Labidi et al., 2018, Paputungan et al., 2018]. However, considering the scale of the IoT environment and the potential massive number of heterogeneous services, precise ontology mapping without bargaining may be inefficient and time-consuming [Li et al., 2019a]. Currently, WS-Agreement Specification (WSAG) [Ludwig et al., 2003] is a widely-used flexible SLA schema for web services, which provides the basis for SLA modeling in many cloud projects [Dimosthenis, 2013]. In addition to describing the service context and the intricate obligations
of involved parties, WSAG defines a decoupled negotiation layer on top of the agreement layer for bilateral multi-round SLA negotiation: WS-Agreement Negotiation Specification (WSAN) [Andrieux et al., 2007], to facilitate dynamic SLA negotiation with compatible service providers and ease the process of creating valid documents (i.e., SLA and negotiation offers). WSAN specifies the schema of negotiation offers and agreement templates conforming to the WSAG. However, it does not support on a constrictive ontology needed to define QoS metrics [Maarouf et al., 2015], and the negotiation process between the negotiation parties before committing and signing the final SLA is not specified [Sharaf and Djemame, 2015]. Recently, an SLA specification for end-to-end IoT application ecosystems has been proposed to accommodate IoT’s multilayered structure [Alqahtani et al., 2019]. This specification introduces the concept of “workflow activities”, which are the required activities to achieve the business goals of an IoT application (e.g., capture patients’ biological data, query analysis results, etc.). However, this approach assumes that a set of standard workflow activities have been pre-defined, and users need to manually select the services that correspond to the workflow activities. This is not a flexible solution since providers do not adapt to match a user’s needs due to the lack of SLA negotiation.

1.3.2 SLA Negotiation Strategy

Research on negotiation strategy is influenced by multiple disciplines, such as economics, mathematics, psychology, political science, etc [Bazerman et al., 2000]. Automatic negotiation among parties with conflicting preferences has been studied in electronic commerce and multi-agent systems [Jennings et al., 2001]. The focus of a negotiation strategy is to optimize the business utility of an agreement for involved parties through negotiation. Feature-rich SLA templates are composed of multiple negotiable attributes that may have various negotiation ranges (i.e., the range between the reserved value and preferred value of an attribute). Different strategies use different conceding tactics to adjust current expectations. For instance, decision functions and utility functions are the widely-used mathematical models that manipulate the negotiation participants’ conceding behaviours under different criteria (i.e., remaining negotiation time, available resources and opponents behaviour) and assess the ag-
Chapter 1. Introduction

Aggregated utility (i.e., level of satisfaction) of different proposals [Faratin et al., 1998, Yao and Ma, 2008, Zulkernine and Martin, 2011]. To accelerate the negotiation duration and maintain the highest possible utility, a tradeoff mechanism was introduced that makes a proposal that has the same utility as the previous proposal, but is more preferable to the negotiation opponent [Faratin et al., 2002, Son and Sim, 2015]. To address the conflict and cooperation problems for self-interested negotiation entities in making rational decisions, game-theoretic approaches can be applied to help negotiation participants contrive the best strategy, depending on the choices of opponents, to gain success [Bui and Jung, 2019]. For example, a Nash equilibrium is determined as a fair solution to both parties in a bilateral bargaining game [Zheng et al., 2010]. To provide some guidance in exploring the negotiation space and make a deal possible, some arguments such as threat, reward and appeal are added to the proposals to imply the possible consequences of acceptance or rejection, or the suggestions over different proposals [Sycara, 1990, Sierra et al., 1997]. These additional arguments are likely to influence the negotiation stances of participants, which may increase the success rate and negotiation speed. However, for automatic SLA negotiation without much human involvement, a unified argument ontology is required for argumentation-based approaches to achieve semantic interoperability, so that an adaptable decision-making model can be created to increase the global benefit [Monteserin and Amandi, 2011].

In the real world, negotiation usually happens with incomplete information as both parties hide their strategy and parts of their negotiation preferences from their opponents to avoid exploitation by competitors [Niemann and Lang, 2009]. A learning mechanism that gradually infers the opponent’s negotiation profile, strategy or deadline by analysing the sequence of offers received during the negotiation process is employed to provide a certain degree of adaptation on making reciprocal acceptable agreements [Baarslag et al., 2016]. Learning techniques including Bayesian learning [Zhang et al., 2008], non-linear regression [Yu et al., 2013], kernel density estimation [Coehoorn and Jennings, 2004], heuristic approaches [Abulkhair et al., 2017, Sim et al., 2009], Markov chains [Narayanan and Jennings, 2006] and neural networks [Carbonneau et al., 2008] have been applied in automated negotiation to minimize negotiation cost and reach a win-win solution. Compared to Kernel Density Es-
timation and Artificial Neural Networks, Bayesian inference is feasible for online learning since it can improve the estimates incrementally through a negotiation process without a training phase. However, many Bayesian-based models assume the negotiators have a prior belief about the opponent’s negotiation preferences, such as the probability distribution of negotiation constraints and the negotiation tactic adopted by the counterpart. This may not be practical in a dynamic environment where neither party knows the other, and both have a fluctuating business model that is adapted to the context information. Also, the number of hypotheses that are custom made or generated from the assumed counterparts’ negotiation strategies has a great impact on the computational complexity [Baarslag et al., 2016], which may introduce an intolerable latency when the model is deployed on resource-constrained devices. For neural network-based approaches, although the computationally intensive training phase can be applied offline depending on the large number of historical data, the possible changes of opponent’s behavior over time is ignored.

Since negotiation is a time-consuming and interaction-rich task, the necessity of candidate service provider selection according to pre-defined conditions has been introduced to reduce the complexity of automatic negotiation in an open market [Mišura and Žagar, 2017]. Apart from the match-making based on functional and non-functional service properties [Li et al., 2019a, Jin et al., 2014], trust and reputation systems represent another significant trend in decision support for internet-mediated service provision [Jøsang et al., 2007]. The trust-based or reputation-based optimal services selection mechanisms have been discussed in cloud computing and web service provisioning [Tang et al., 2017, Wang and Vassileva, 2007, Vu et al., 2005]. However, these approaches only consider the execution performance of services, while the support for efficient negotiation is neglected.

1.3.3 SLA Negotiation Protocol

To enable automatic negotiation, a protocol is needed to define messages and specify the interaction rules including negotiation states, event-changing states, and the action that needs to be taken in a particular state [Jennings et al., 2001, Yan et al., 2006]. The single-round “take it or leave it” protocol is the most simple one that is adopted by WSAG specifica-
Chapter 1. Introduction

1.3.4 Research Gaps and Observations

Based on the challenges described in section 1.2 and the limitations of the existing solutions, the following research gaps are observed. First, the existing SLA languages designed for cloud services and web services do not capture the characteristics of IoT services. However, providing a precise SLA specification for IoT services would enable a better QoS-aware service management [Kazmi et al., 2017]. For example, the varying syntax of different SLAs obstructs
automatic service matching and SLA negotiation in large-scale electronic markets. Consumers or third-party audit agents struggle to detect SLA violations unless they understand the SLA document. How to draft SLAs with respect to describing heterogeneous IoT services, as well as efficiently specifying the negotiation and monitoring information is still a problem.

Second, many existing negotiation strategies may be unsuitable for SLA negotiation in the IoT context because they do not support the evaluation of IoT service properties, or they are too heavyweight for resource-constrained IoT gateways. Compared to web services and cloud services, the negotiable issues of IoT services may be more diversified due to the existence of mobile devices and reconfigurable resources. For example, service coverage can be affected by the sleep schedule of sensor nodes [Gupta et al., 2016], and a provider may be willing to change devices’ status in line with the user’s demand to maximize profit [Mišura and Žagar, 2017] (e.g., reduced sleep mode for higher sampling rate). Under these assumptions, a service evaluation method, which considers IoT domain-specific properties (e.g., location, sampling schedule, data rate) is needed to detect suitable services. Also, for an IoT environment such as a smart city, the data transmissions between devices and cloud, and the spontaneous interactions amongst devices may produce an enormous number of messages. Considering the possible large connection density and the traffic capacity limit of wireless networks [Fujino et al., 2016], the negotiation process should avoid transmitting oversized packages and reduce the number of interactions with candidate service providers. More specifically, to reduce network congestion, the data model of negotiation information should be lightweight, and a negotiation strategy that can balance the success rate and negotiation efficiency with restricted interactions is encouraged for IoT SLA negotiation. This can be achieved by designing a negotiation tactic that can adapt to context information or the opponent’s behaviour to dynamically adjust concessions, and a provider selection mechanism when multiple candidates that have potential to satisfy a request are identified in the environment.

Third, considering the scale and the intermittent availability of geographically distributed service providers, the auction-based multilateral negotiation protocol may be unsuitable for IoT SLA negotiation due to the possible large latency and the absence of a centralized co-
ordinator. The existing bilateral negotiation protocols are focused on modeling the offer exchanging process, which does not address the negotiation problems that emerge from insufficient awareness of local context (e.g., service location, newly joined service providers) and frequent network disconnections introduced by mobile entities [Li and Clarke, 2018].

Figure 1.6 generalizes the relationships between IoT characteristics, negotiation problems and negotiation system requirements. In summary, automated SLA negotiation in a large and dynamic IoT environment should consider following:

- **SLA description using an IoT domain-specific ontology:** Service providers should express their offerings and negotiation information in a standardized way to achieve semantic interoperability.

- **Dynamic SLA template organization:** The SLA templates should be organized in a distributed manner to provide wide coverage and facilitate the run-time negotiation.

- **Distributed candidates selection and multi-bilateral negotiation:** The negotiation system should identify the trustworthy service providers based on SLA templates and historical data, and automatically perform time-constrained multi-bilateral negotiations without the management of a centralized controller. The negotiation strategy
should balance the trade-off between success rate and negotiation utility within limited negotiation rounds.

- Communication management: To adapt to changes to mobile entities’ communication path, the negotiation system should have an efficient message forwarding mechanism to avoid message flooding and reduce the negotiation timeout failures.

1.4 Research Questions

Based on the observed knowledge gaps, this thesis explores the question of how to identify the candidate service providers that are likely to satisfy a consumer’s negotiation constraints in a dynamic large-scale environment, and negotiate with them to resolve possible preference conflicts and reach to an agreement. This question can be decomposed as follows:

**RQ. 1. Negotiation object:** To what extent can the use of SLA templates to describe negotiable IoT services accommodate the requirements of finding the negotiation candidates that have the potential to match a consumer’s request?

**RQ. 2. Negotiation protocol:** To what extent can the communication problems during an SLA negotiation process be addressed in a large-scale environment where negotiating entities may be mobile and distributed in different locations?

**RQ. 3. Negotiation candidates selection:** Given a request, to what extent can the use of historical information to select negotiation candidates improve both negotiation efficiency and the consumer’s satisfaction level?

**RQ. 4. Negotiation strategy:** To what extent can a negotiation strategy balance the tradeoff between success rate and negotiation utility in a time-constrained negotiation scenario?
1.5 Thesis Approach

This thesis presents iNegotiate, a distributed negotiation system designed to be deployed in a dynamic IoT environment to negotiate SLA parameters with suitable service providers on behalf of users. The following assumptions are made about the negotiation environment and the general negotiation process:

Figure 1.7: Negotiation gateways (GW) in an IoT environment

Assumption 1. To enable an automatic SLA negotiation in a large-scale environment, we assume a negotiation system is deployed on a set of edge devices, which manages SLA negotiation, creation and monitoring in a distributed manner (Figure 1.7). These edge devices are referred to as negotiation gateways. Negotiation gateways can be a resource-constrained device such as a Raspberry Pi deployed in a road-side unit, or a dedicated device such as a desktop installed on supermarket’s information kiosk. Negotiation gateways may or may not have Internet connection. They either use Ethernet or WiFi interfaces to communicate with other negotiation entities (i.e., gateways, service providers and consumers). There is no central controller that manages the negotiation tasks in the environment.
Assumption 2. Consumers have no prior knowledge about the resources currently available in the environment. The service providers who are offering SLA-supported services outline service properties and negotiation information in SLA templates by following a unified ontology, and advertise the templates to the negotiation system. An SLA template can be regarded as a partially completed agreement with default term values capturing providers’ preferred offerings and the possible constraints to create a valid negotiation offer or the final SLA. For instance, to avoid unnecessary interactions during negotiation, a service provider may specify constraints relating to spatial features by listing all the available service locations in the template for gateways to select the most preferred one. However, providers may regard the negotiation constraints relating to non-functional service properties as business-sensitive information and may be unwilling to disclose them to the public [Holloway, 2017]. SLA templates are stored in the system within the period of validity so that the corresponding services can be discovered when requests are received.

Assumption 3. We assume some of the service providers are cloud/web service providers who expose negotiation services as RESTful APIs (i.e., static service providers); while others are autonomous service providers who negotiate with gateways through the mobile devices (i.e., mobile service providers). Considering the uncertainty in providers’ availability and mobility, SLA templates are distributed in the gateway network depending on the services’ spatial features. This is under the assumption that service providers are more likely to appear or move around in the areas that are close to the advertised service location. If a gateway detects a locally-stored template that has the potential to meet a consumer’s request, the gateway has a bigger chance to directly connect to the service provider to start a bilateral negotiation.

Assumption 4. Consumers submit requests to a surrounding negotiation gateway via WiFi. These requests are forwarded to different gateways that are close to the requested service locations. To detect the candidate services that possibly could solve the request, we assume gateways use a heuristic backward-planning algorithm to discover local registered templates that match consumers’ functional requirements according to the semantic
relations of service parameters (i.e., input and output) [Cabrera et al., 2018].

**Assumption 5.** To verify whether the prior negotiated SLAs are maintained by the respective service provider at runtime, monitoring solutions offered by service providers are insufficient since they may not be an independent and reliable evidence base for detecting SLA violations from a consumer’s perspective. This work envisions an intermediary-based monitoring approach, where a monitoring engine is deployed on the negotiation gateway to collect run-time QoS data according to the service level objectives (SLOs) and QoS metrics specified in the SLA, and to predict possible degradation in service quality to enhance consumer satisfaction and business continuity [Fatema et al., 2014].

Basically, this thesis explores the question of how to enable a timely and efficient SLA negotiation in a dynamic large-scale environment, so that a user’s request can be satisfied as much as possible within the anticipated negotiation time. As opposed to the existing cloud-based QoS negotiation approaches, this thesis investigates the questions outlined in Section 1.4. Four hypotheses are proposed according to these questions:

**Hypothesis 1.** A common SLA ontology adopted by both service consumer and provider is useful for describing supplies and demands in a uniform way, which facilitates automated SLA negotiation.

**Hypothesis 2.** A distributed negotiation system is feasible to address the communication issues of SLA negotiation in a dynamic large-scale environment when it is equipped with a template distribution and mobile entity management mechanism.

**Hypothesis 3.** Identifying trustworthy candidate service providers based on past experiences and prioritizing them before the negotiation is helpful to enhance the negotiation success rate and user’s satisfaction for the time-constrained negotiation scenario.

**Hypothesis 4.** A negotiation strategy that can dynamically adjust the concession rate based on context information or negotiation opponent’s behavior can help to balance the tradeoff between negotiation success rate and the optimality of negotiation results.
The following objectives are proposed based on the hypotheses and the state of the art analysis:

**Objective 1.** Formalizing SLA and SLA template in a standardized way is the precondition to automate the SLA negotiation and creation process. This work generalizes a common SLA ontology for IoT services based on the existing IoT reference model and SLA standards for web services.

**Objective 2.** A large number of service providers available in the IoT environment negatively impact negotiation performance. Previous centralized mediator platforms do not consider the scale of the environment and the possible wireless connections of negotiation parties. The objective of this work is to design a negotiation distribution model that forwards requests to different gateways to ensure a timely multi-bilateral negotiation without introducing heavy network traffic.

**Objective 3.** A mission-critical IoT application that has stringent QoS demands needs to make an agreement with a trustworthy service provider, who is likely to maintain the promised quality levels during the service provisioning stage. The objective of this work is to design a provider selection model that captures the capability of a service provider offering the requested service with satisfying performance.

**Objective 4.** Consumers may have various requirements on functional or non-functional service properties. The objective of this work is to design a decision-making algorithm that can quantitatively evaluate the combination of different negotiation issues, and make concessions based on the identified context such as time, negotiation preference, available resources, behaviour of the counterpart, etc.

### 1.6 Thesis Contribution

This thesis introduces the iNegotiate negotiation framework, which enables automatic SLA negotiation in a large-scale environment by clustering SLA templates based on location information, forwarding requests to different negotiation gateways where they are most likely to
Figure 1.8: Ontology of iNegotiate framework
be solved, and bargaining with suitable service providers to tailor the service properties based on users’ requests. Figure 1.8 shows the ontology of the iNegotiate framework, which outlines the required features of SLA negotiation in the IoT environment (i.e., scalability, adaptability, timeliness, mobility, heterogeneity) in terms of negotiation object, negotiation strategy, and negotiation protocol. The research contributes to the body of knowledge including:

- **WS-Agreement based SLA modeling for IoT services**

  To achieve semantic interoperability and reduce the ambiguity in automating SLA negotiation and monitoring activities, creating an SLA ontology is the common solution [Redl et al., 2012]. However, previous SLA languages are designed for web services or cloud services and do not capture the IoT domain-specific properties. iNegotiate proposes an ontology-based SLA management model for service providers expressing their offerings in a standardized way, called WIoT-SLA. This ontology generalizes the semantics of SLA and negotiation offers so that the dynamic SLA negotiation and SLA creation can be performed according to the negotiation context, creation constraints and validation rules specified in the SLA template. It can also be used to configure a monitoring instance that can detect SLA violations based on the semantics of SLOs and quality metrics specified in the final SLA. The WIoT-SLA ontology combines two commonly-used web service SLA specifications: WS-Agreement and WSLA, to take advantage of their complementary features. This knowledge model can also be extended by domain-specific experts to construct SLAs for various IoT applications. This contribution is focused on answering RQ 1.

- **The distributed SLA negotiation model in the IoT environment**

  Existing negotiation approaches for IoT assume a centralized cloud-based architecture, which may not be practical given the scale of localized sensors deployed in different IoT platforms and the presence of autonomy of service providers that have limited communication ranges. iNegotiate proposes a novel self-organized negotiation model for the IoT environment that enables distributed negotiation gateways communicating through a logistic overlay network to efficiently negotiate with candidate service
providers on behalf of users. This overlay network has a three-layered architecture that classifies gateways into different groups. Each group has its specific responsibilities, and different groups cooperate with each other to accomplish the negotiation task in a distributed fashion. By automatically creating a hierarchical negotiation overlay, iNegotiate performs location-aware template organization/clustering to facilitate negotiation task allocation, and uses a layer-based communication mechanism to control the message flows and avoid the risk of message flooding. iNegotiate formalizes the negotiation model using a distributed negotiation protocol, which defines the communication messages, sequenced behaviours during the negotiation process, the message interaction rules, and the action that needs to be taken when receiving a particular type of message. This contribution is focused on answering RQ 2.

- **Trust-based evaluation of negotiation candidates**

With a possible ever-increasing number of service providers in an IoT environment, multi-bilateral SLA negotiation is likely to be prohibitively time-consuming and inefficient without a priori process to select trusted candidate providers with whom to negotiate. Current service selection approaches use service match-making or a reputation-based system to identify optimal services, which do not consider the negotiation performance of corresponding service providers. iNegotiate presents a trust model to evaluate candidate service providers before attempting to negotiate an SLA, which provides the basis for a negotiation gateway to select the optimal candidate when it discovers multiple services. The trust model evaluates service providers from two aspects: negotiation competence and provider’s integrity. This identified information is derived from historical data relating to previous negotiations and monitored service execution performance. Negotiation competence models the possibility of a successful negotiation with a service provider, which is inferred from the weighted similarity between a request and the recent successful negotiation records. Indiscernibility analysis from Rough Set theory is used to analyze the weight of different negotiation issues in making decisions. A provider’s integrity models the possibility of a service provider keeping its promises, which is reflected from the SLA violation rate deduced from Bayesian inference. Since IoT services
exhibit a time-varying nature in terms of service qualities, to avoid choosing the services that are likely to have a severe degradation in service quality, the current service performance is also analyzed in the integrity assessment by detecting abnormality from recent QoS observations. This contribution is focused on answering RQ 3.

- **A deadline-aware strategy for IoT service negotiation**

  Although existing SLA negotiation specification defines the sequential offer exchange rules for a bilateral negotiation process, the negotiation strategy and corresponding decision-making model are not specified. Current research on negotiation strategy is focused on web services negotiation or cloud services negotiation, which does not consider the domain-specific properties of IoT services or the dynamicity of the environment.

  To address this problem, iNegotiate proposes a deadline-aware SLA negotiation strategy for multi-round bilateral negotiation. This strategy is composed of three parts: the mathematical model that evaluates received offers based on users’ requests, the decision-making model that specifies whether to accept/reject the offer or propose a counteroffer, and the negotiation tactic that generates counteroffers based on context information or the behaviour of negotiation opponent. The mathematical model contains three types of scoring functions that quantitatively evaluate different types of service properties (i.e., QoS parameters, data rate, temporality, service coverage). The decision-making model is built based on the offer state transition rule defined in WSAN specification. To keep the negotiation utility as high as possible, iNegotiate defines two negotiation tactics targeting different negotiation scenarios: a context-based tactic whose concession rate is controlled by context information (i.e., the number of candidates and the user’s negotiation preference), and an artificial bee colony (ABC) algorithm-based tactic when the user’s negotiation preference is unknown. This contribution is focused on answering RQ 4.
1.7 Thesis Scope

iNegotiate manages SLA templates that are formalized using the WIoT-SLA structure. The mechanisms that autonomously enrich the concepts of SLA terms in WIoT-SLA ontology are outside the scope of this thesis. Two alternatives could help iNegotiate to understand new negotiation concepts and enrich the global ontology: the semantic negotiation that allows entities within a community sharing, acquiring and validating unknown knowledge [Comi et al., 2015], and the semantic enrichment of service description models using unsupervised probabilistic machine learning techniques [Cassar et al., 2013].

iNegotiate evaluates candidate service providers from a negotiation perspective and assumes these candidates are already discovered based on the user’s functional requirements. The actual discovery process is outside the scope of this work. Current research on goal-driven service discovery in smart cities using semantic relations of service parameters (i.e., input and output) [Cabrera et al., 2018] could enable iNegotiate to identify the candidates that meet user’s functional requirements.

iNegotiate dynamically negotiates with service providers who advertise SLA templates to the system. The negotiation for SLA establishment for composite services is outside the scope of this work. The SLA negotiation of composite service involves compound multi-party negotiations in which the concurrent negotiations with multiple candidates for atomic services are required to satisfy the end-to-end QoS requirements. The negotiations with potential atomic service providers require the dynamic derivation of the individual negotiation boundaries from the global negotiation boundary as the multi-party negotiation proceeds. Existing research on the negotiation boundary decomposition and the surplus redistribution [Richter et al., 2012] can be used by iNegotiate to decide the negotiation boundaries for each atomic service in a composition. Based on the derived negotiation constraints, multilateral negotiation can be performed to tailor the service properties with corresponding atomic service providers, and the negotiated results can be aggregated to verify whether the user’s end-to-end QoS requirements have been satisfied [Shojaiemehr et al., 2019].

iNegotiate specifies the semantics of monitoring and accounting information, such as the
QoS metrics, SLOs, guarantee conditions, penalties, and assessment intervals. The actual SLA monitoring and billing mechanisms are outside the scope of this thesis. To verify if a service provider assures SLA compliance with the negotiated thresholds, the QoS metrics should be collected and analyzed at runtime. This can be achieved by either actively sending synthetic requests to gather information (i.e., probing) [Michlmayr et al., 2009], or passively collecting existing packages exchanged between providers and consumers [Molina-Jimenez et al., 2004]. Also, the QoS prediction approaches using collaborative filtering [White et al., 2018a] or Long Short-term Memory (LSTM) network [White et al., 2018b] can enable iNegotiate to forecast the possible degradation in service quality.

iNegotiate employs a location-based negotiation task allocation mechanism under the assumption that various sensors are deployed in different locations or IoT platforms. The load balancing during the request forwarding process is outside the scope of this thesis. The existing load balance approaches in the fog environment [Ningning et al., 2016] can be added to iNegotiate as a supplementary mechanism to achieve resource efficiency and avoid bottlenecks (e.g., network overload, gateway failure, etc.) in high service density areas.

1.8 Thesis Structure

The rest of this thesis is structured as follows:

State of the art Chapter 2 analyses the research related to SLA negotiation in different fields and the state of the art SLA negotiation approaches in the IoT environment. In particular, this chapter analyses the related works from four aspects: a) the principles of SLA management and different SLA languages targeting various application scenarios, b) how are modelled machine-readable SLAs to automate the SLA negotiation, creation and monitoring process, c) how automated negotiation is performed in different computing environments, and d) how the negotiation priorities of candidate service providers to enhance negotiation efficiency and the consumer’s satisfaction are identified.

Design Chapter 3 describes the design objectives, system model, and design decisions of this thesis based on challenges and research questions outlined in Chapter 1. It intro-
duces WIoT-SLA ontology to formalize SLA, SLA templates and negotiation offers, specifies a three-stage multi-bilateral negotiation model and the corresponding sequenced behaviors that are controlled by a distributed negotiation protocol, defines the trust model to identify candidates’ negotiation priority and describes the negotiation strategy for generating counteroffers and making decisions.

**Implementation** Chapter 4 describes the implementation of iNegotiate according to the design presented in Chapter 3. It presents the structure, behaviour, and component interactions of iNegotiate.

**Evaluation** Chapter 5 evaluates how well iNegotiate achieves its objectives outlined in Chapter 1. It describes the experimental setup, specifies the evaluation metrics, and presents the analyzed results showing the feasibility, efficiency, and limitation of iNegotiate.

**Conclusion** Chapter 6 concludes this thesis by generalizing the achievement and limitation of the proposed solution and highlights potential areas for future work.

### 1.9 Chapter Summary

This chapter introduced the background of this research, together with the general description and scope of the proposed iNegotiate solution. It first describes the service-oriented computing in the IoT environment, specifies why SLA management is required for reliable service provisioning and presents the incentives of applying negotiation mechanism in different computing environments. Then it outlines the challenges of this research, and analyze the limitations of existing solutions. Based on the identified knowledge gaps and required features of the SLA negotiation system, four research questions are proposed as guidance to introduce the objectives and contributions of iNegotiate.
Chapter 2

State of the Art

This chapter reviews current research related to SLA negotiation in different domains, including web services provisioning, cloud computing, MANET, and WSN. Based on the characteristics of the web services market, a negotiation system should consider a set of problems including service provider selection, service usage prediction, functional composition, management of service dependencies and uncertainty, negotiation convergence, contract enactment and trust management of negotiating parties [Elfatatry and Layzell, 2004]. More generally, many existing works suggest that the negotiation object, negotiation protocol, and negotiation strategy are the three important topics that should be considered when designing an automated negotiation system [Jennings et al., 2001, Yao and Ma, 2008, Zheng, 2014]. This chapter describes the related works in the context of these identified topics and specifies their limitations by analyzing the extent to which they meet the requirements of efficient SLA negotiation in large and dynamic IoT environments. This review follows the structure outlined in Figure 2.1, which corresponds to the research questions identified in Chapter 1.4. This chapter first reviews the work of SOA-based IoT systems and introduces the works related to the negotiation object (SLA) modeling. These works include SLA lifecycle that automates service management activities in service-oriented environments, existing SLA languages that describe services and obligations, existing ontologies that abstract things or their measurements as IoT services, and smart contract that can automate SLA monitoring and billing mechanisms without a trusted authority. Then it assesses how existing negotiation frame-
works solve negotiation tasks using different negotiation protocols and negotiation strategies. Finally, it generalizes different provider selection approaches that identify the potential service providers based on a consumer’s request.

2.1 Service-oriented Computing

Service-oriented computing is a conceptual paradigm to which software applications are built by using existing loosely-coupled services, which are self-contained software entities performing pre-defined tasks [Erl, 2005]. The tasks can be described using their functional operations and the messages used to trigger the operations, including inputs and outputs. A set of non-functional service properties like response time, throughput, reliability, availability, and scalability may be associated with the service describing its run-time performance. Traditional SOA involves three main actors that directly interact with each other: a service provider, a service consumer, and a service registry. Service consumers look for services depending on the service capabilities described in the registry and bind to the chosen service using a pre-defined transport protocol (e.g., sending SOAP messages over HTTP). However, this traditional SOA paradigm does not consider the service’s capacity, which is defined as a service being provided at a given quality level within a period of time for a specific consumer’s constraint [Ludwig, 2006].
2.1.1 Capacity-aware SOA

To provide a certain level of control to a consumer and enable a QoS-aware service provisioning, capacity-aware SOA was proposed, which regulates that service providers and consumers should agree, prior to service usage, on the conditions under which the requested services can be legally consumed (e.g., limited request rate, guaranteed performance) [Ludwig, 2006]. Figure 2.2 shows the interactions of capacity-aware SOA, driven by agreements.

The service invocations between a service consumer and a service provider are governed by a pre-negotiated agreement. To obtain the authorization to legally use a service, the consumer submits a request specifying the required capacity and non-functional properties to an agreement management component, which identifies suitable service providers and negotiates an SLA with them to tailor a service based on the request. According to the negotiated SLA, information describing the negotiated terms and how to access the service (e.g., a specific endpoint or a token added to the SOAP header when requesting the service) is returned to the client application for service invocation and monitoring. If the monitored service performance does not conform to what has been guaranteed, correction actions (i.e., re-configure resource allocation, SLA renegotiation, etc.) or financial recourse can be applied to ensure the rights and obligations of involved parties.

Compared to traditional SOA, some extra functionalities are required to enable capacity-
Chapter 2. State of the Art

aware SOA. Both service providers and consumers should support the SLA management life-
cycle including SLA creation, SLA negotiation, SLA monitoring, and possibly, SLA renewal
[Faniyi and Bahsoon, 2016]. For example, service providers would need to design a decision-
making function that derives the resource allocation for a particular request and assesses
whether reaching an agreement is feasible and economically viable.

2.1.2 Service-oriented computing for the IoT

To create IoT applications flexibly and efficiently, the cooperation of heterogeneous network-
connected devices are encouraged [Marron and Minder, 2009]. Service-oriented computing
is a promising paradigm envisioned for the IoT environment due to its inherent support for
 interoperability and composability [Guinard et al., 2010, Da Xu et al., 2014, Ahmed et al.,
2019]. A system can be referred to as an IoT system if the data transmitted across a network
are generated under the control of devices [Green, 2014]. As Figure 2.3 shows, in SOA-based
IoT systems, network-connected sensors can offer their functionalities via SOAP-based Web
Services or RESTful APIs [Guinard and Trifa, 2009, Gubbi et al., 2013, Zanella et al., 2014],
which can be used by other entities such as business applications or other devices to retrieve
real-time data about the physical world. Figure 2.4 shows the layered structure of capacity-
aware SOA in IoT environments, which consists of sensing layer, networking layer, service
layer and interface layer. The sensing layer is composed of heterogeneous devices including
RFID, sensors, and actuators that sense and control the physical world. The networking
layer connects these devices allowing them to transfer data and sharing information. The
service layer relies on the middleware technologies to seamlessly integrate services in the
IoT, which processes service-oriented functionalities such as service discovery [Atzori et al.,
2010, Marino et al., 2019], service composition [Atzori et al., 2010], service negotiation [Mišura
and Žagar, 2017, Marino et al., 2019], QoS monitoring[Duan et al., 2011] and trustworthiness
management [Atzori et al., 2010]. The interface layer describes the specifications or standards
that facilitate interactions between applications/users and services. In this layer, contract
management is an important component to enable demand-driven and QoS-aware service
provisioning, as well as ensuring the smooth running of IoT applications [Marino et al.,
Enabling the four-layered architecture faces the challenges introduced by IoT characteristics such as large-scale, deep heterogeneity and high dynamics [Issarny et al., 2016]. For example, the Application Programming Interface (APIs) and protocols implemented by different IoT stakeholders may differ significantly in terms of interaction styles and data formats. The lack of a unified standard regulating IoT service abstraction and access in the application layer brings new interconnection problems for SOA-based IoT ecosystems [Issarny et al., 2016]. Recent research on applying middleware technologies in IoT environments has resulted in a number of solutions in the service layer [Palade et al., 2018]. However, very little attention has been paid on SLA development and management in application layer, although its importance for reliable service provisioning had been identified in recent works [Papadopoulos et al., 2017, Mubeen et al., 2017, Marino et al., 2019, Ahmed et al., 2019].

2.1.3 Assessment

Integrating SOA into IoT-based systems provides an efficient and flexible way to create IoT applications by re-using loosely-coupled services abstracted from heterogeneous things or
Figure 2.4: Four-layered capacity-aware SOA structure for the IoT [Da Xu et al., 2014]

their measurements. Traditional SOA follows the “publish, find and use” pattern to provide homogeneous access to various resources without revealing the heterogeneous nature of the underlying infrastructures [Guinard et al., 2010, Da Xu et al., 2014]. However, this pattern implies the consumer’s high dependency on the service provider due to the lack of control on service performance. To support a legal assurance on QoS for consumers, capacity-aware SOA follows an agreement-driven pattern that provides services under the control of a pre-negotiated SLA, which specifies the service level guarantees that a provider should deliver and the consequences if the commitments are violated [Ludwig, 2006]. Enacting capacity-aware SOA in IoT systems requires middleware technologies to discover requested services and bond the service provider to a consumer through an SLA. However, existing IoT middlewares focuses on service discovery, service composition and data management, while the
SLA management mechanism is ignored.

2.2 Service Level Agreement Development

Generally, SLA is a business concept describing the contractual financial agreement between trading parties who engage in a business activity [Faniyi and Bahsoon, 2016]. Due to fluctuating market demands and various unpredicted users’ requirements, service provisioning has shifted from a statically pre-defined mode to a demand-driven orientation [Dimonsthenis, 2013]. Computing services can be delivered on-demand as other utilities like water or electricity. In such a utility computing system, SLA is a legal contract that specifies the party information, business terms, and expectations/obligations of trading parties, which enhances the consistency of service performance and user satisfaction [Wu et al., 2012]. An example is that SLAs are used as the basis for efficient resource management and scheduling in grid computing [Wieder et al., 2008]. Compared to the significant works that have been done on SLA development for web or cloud services, related research in the IoT domain is still in a preliminary stage [Papadopoulos et al., 2017]. This section firstly describes the SLA lifecycle and current SLA management solutions in the IoT environment. Then it introduces existing SLA specifications designed for different application scenarios and generalizes the ontologies describing devices or devices’ measurements in IoT environments. Finally, it discusses the potential usage of smart contracts in automating SLA management without the arbitration of a trusted authority.

2.2.1 SLA Lifecycle

According to the European Commission report on SLA management of recent cloud computing projects, the SLA lifecycle meta-model consists of six main phases [Kyriazis, 2013]. The goal and potential dependencies of each phase are as follows:

(a) **Service use**, which reflects information on service usage by a consumer. The high-level attributes related to the service should be described in an SLA.

(b) **Service modeling**, which deals with the service design, modeling, and analysis issues.
The parameters affecting the service execution, usage and delivery should be captured, such as the potential dependencies between services, elasticity rules, and the behaviour hints that are required to guarantee the service performance.

(c) **SLA template definition**, which creates and refines SLA templates based on the providers’ business objectives that optimizes the offerings.

(d) **SLA instantiation and management**, which covers various processes to create a signed SLA between trading parties, including attributes mapping or translation, service provider discovery, and most importantly, the dynamic SLA (re-)negotiation.

(e) **SLA enforcement**, which aims to verify the reliability of pre-negotiated QoS parameters during the service provisioning time by exploiting adaptable QoS monitoring and SLA violation detection mechanisms.

(f) **SLA conclusion**, which handles the termination of signed SLAs or violated SLAs according to the pre-defined accounting and billing mechanisms. To maintain service continuity, SLA renegotiation may be triggered as a corrective action when a violation happens or be predicted to happen [Hani et al., 2015].

The lifecycle beta-model implies that SLA development is crucial in SLA-supported service provisioning since it provides the basis for automated service matching, SLA negotiation, and QoS monitoring. Existing works had identified two general requirements for developing an SLA language: precision and flexibility [Sahai et al., 2002]. An SLA must be specified unambiguously so that both parties have a clear knowledge about the service offerings and their corresponding obligations or guarantees. The typical elements of an SLA specification include the roles of trading parties, the SLA validity period, the service operations covered in the SLA, the service level indicators with associated metrics, the guarantees that needs to be monitored at runtime, and the penalties or actions to be undertaken when those guarantees are not satisfied [Anderson et al., 2005]. On the other hand, the heterogeneous nature of services offered by various third-party service providers demands a flexible SLA language that can be extended with new elements to accommodate the requirements of a specific service
domain. In cloud computing, XML-based SLA languages are widely used due to the extensibility and popularity of XML\(^1\) in service-related specifications such as WSDL\(^2\) and SOAP\(^3\) [Kyriazis, 2013]. However, resource-constrained IoT devices may not be powerful enough to afford the computation cost of full-fledged XML processing, while JavaScript Object Notation (JSON)\(^4\) is a lightweight data representation syntax that is more suitable for storing and exchanging text information in IoT systems [Soliman et al., 2013].

As an open-source IoT platform composing and delivering IoT services that comprise data from multiple sensors [Soldatos et al., 2015], OpenIoT proposed a self-management framework based on a six-phase IoT service lifecycle [Calbimonte et al., 2013], which shares something in common with the SLA management lifecycle described in Chapter 1.1. The proposed IoT service lifecycle consists of a service creation phase, a service customization phase, a service management phase, a service operation phase, a service billing phase, and a customer support phase. Service creation is supported by an efficient scheduling optimization scheme including multi-query data management and caching techniques. Service customization is implemented by combining query languages with stream data processing components. Service management includes a set of operations such as service distribution, service maintenance, service invocation, service execution, and service assurance policies. Service operation is usually monitored by agents that can translate raw data into explicit semantics that suit the needs of different applications. Service billing uses a pre-defined accounting mechanism to charge customers according to the required resources. Customer support provides assistance with purchased IoT services. However, the drawback of the self-management framework is that, it only supports the best-effort services. There is no contract that explicitly specifies service properties, QoS guarantees, and the obligations of involved parties. This may obstruct the self-management process in complex situations such as customizing IoT services through bilateral negotiation and automatically enforcing the service monitoring and billing mechanisms that are fair to both trading parties. Similarly, Balint \textit{et. al.} proposed an extensible platform that automat-

---

\(^1\)https://www.w3.org/XML/
\(^2\)https://www.w3schools.com/xml/xml_wSDL.asp
\(^3\)https://www.w3schools.com/xml/xml_soap.asp
\(^4\)https://www.json.org/json-en.html
ically distributes data to relevant subscribers in accordance with negotiated data contracts [Balint and Truong, 2017]. During the process, the monitoring microservice instances are configured to verify QoD and QoS based on the corresponding metric. However, the model of the data contract and the concrete negotiation process by which the data contract is created are not specified.

### 2.2.2 Service Level Agreement Specifications

To automate the SLA management procedure through which different SLAs are flexibly negotiated and observed, SLA modeling is the first step allowing service providers and consumers to collect details about the service properties [Kyriazis, 2013, Faniyi and Bahsoon, 2016]. Works related to SLA modeling for web services and cloud computing have been on-going since 2003. IBM published **Web Service Level Agreement** (WSLA), which is an XML-based specification defining the SLAs and SLA management mechanism for web services [Ludwig et al., 2003, Keller and Ludwig, 2003]. WSLA supports the precise definition of how SLA parameters are supposed to be computed or aggregated from resource metrics. The SLA parameters are specified in the service description and each parameter has a reference to its metrics. The anticipated service quality with respect to business and technical level objectives are defined in SLOs, which are monitored during the service provisioning stage by either or both signatories. WSLA consists of three parts: *parties* (i.e., signatory and supporting parties), *service description* including SLA parameters and metrics, and the *obligations* of involved parties.

Although WSLA specifies a run-time architecture that interprets SLAs and handles SLA management tasks, it lacks support for automated SLA negotiation and creation. **WSA-Agreement Specification** (WSAG) is another popular web service SLA specification defined by Open Grid Forum (OGF) [Andrieux et al., 2007, Battré et al., 2010], which enables the automation of SLA creation process and allows service consumers to dynamically explore possible services with their appropriated quality levels. WSAG proposes a layered model that the delivered service performance is monitored on the service layer, and the measurements are evaluated in the agreement layer to verify SLA compliance. WSAG uses a XML-based
schema to formalize an SLA/template, which consists of an agreement identifier, agreement name, agreement context (i.e., parties, template reference, expiration date, etc), and a set of terms describing the service. WSAG proposes two important types of terms: service description terms and guarantee terms, which describe the functionalities to be delivered by the service and the QoS assurance with the conditions under which this guarantee applies respectively. For SLA templates, an optional creation constraint section can be defined to validate the agreement requests. In general, WSAG defines a high-level structure for SLAs/templates that must be complemented with expressions suitable for a particular domain [Ludwig et al., 2006], which makes it a flexible language to describe SLAs in diverse application scenarios. However, it lacks a constructive ontology to define QoS metrics, which obstructs the semantic interoperability for SLA monitoring supported by a third-party. To facilitate run-time SLA management and improve service continuity, Frankova et. al. analyzed WSAG using finite state automata, and enriched it with a number of extensions such as the initial negotiation of an agreement, issuing warnings before a possible violation, and the renegotiation of a running SLA [Frankova et al., 2006]. However, the negotiation mechanism defined in WSAG only supports a single interaction that the agreement responder can only accept or reject by unilateral decision, which is inadequate to cater for business-oriented aspects of a contract such as pricing and negotiation. Karaenke et. al. analyzes how SLAs can support service transactions in electronic markets using a three-phase transaction model, which consists of the information phase in which details about the services and parties are collected, the agreement phase in which the parties negotiate service terms and price, and the settlement phase in which an SLA is enforced and evaluated [Karaenke and Kirn, 2007].

Inspired by WSAG and the three-phase transactions model, a specification named Service-Level-Agreement for Clouds (SLAC) has been proposed to support the business aspects such as brokerage (e.g., matching, intermediation, aggregation or arbitrage) and pricing/billing models (e.g., flat, variable or auction) [Uriarte et al., 2014]. SLAC predefined a set of metrics based on the requirements of Infrastructure-as-a-Service (IaaS) and specified a denotational semantics (i.e., the elements of the syntax are symbolic realizations of abstract mathematical objects or functions) to check the conformance of SLAs. Since the flexible
on-demand service provisioning is a key characteristic of the cloud computing paradigm, a business extension is defined in SLAC to specify different pricing schemes, including fixed pricing, bilateral agreement, exchange, auction, posted pricing and tender [Uriarte, 2015]. Although SLAC considers the possibility of the bilateral agreement through bargaining, the semantics of negotiation message and the concrete negotiation process are not defined.

As a cloud project aiming at providing predictability, dependability and automation in all phases of the SLA management lifecycle, SLA@SOI proposed an abstract SLA syntax named SLA* [Kearney et al., 2010, Chronz and Wieder, 2010] to support services in general. SLA* consists of five parts: template attributes (e.g., template id, temporal availability and agreed time), agreement parties identified by the role (i.e., supplier, consumer), service description defined by interface declarations, variable declarations, and QoS guarantees formalized as action warranty and status. Based on SLA*, Kouessi et. al. introduced an SLA negotiation and monitoring framework using simulation data, and a set of SLA templates are presented as examples [Sagbo et al., 2016]. However, the SLA* specification lacks precise semantics due to its multi-domain support, which is not easy to understand by non-technical consumers. The domain experts need to understand the specification and develop domain-specific vocabularies for each use case.

Based on WSLA and WSAG, the Cloud Service Level Agreement (CSLA) has been proposed to finely express SLAs and address SLA violations in the cloud [Kouki et al., 2014] domain. The structure of CSLA is similar to WSAG, and consists of the parties (i.e., signatory parties and supporting parties), validity and template. Validity outlines the temporal features of an SLA such as the initial and expiration dates. The template describes the service definition, the associated parameters, the guarantees related to these parameters, the billing scheme, and the termination conditions. To support the cloud elasticity management such as the QoS or functionality degradation in an unpredictable and dynamic environment, CSLA introduces the concepts of fuzziness and confidence, which specify the error margin for a metric and the minimum ratio of the enforcements that the metric values do not exceed the guaranteed thresholds respectively [Serrano et al., 2013]. For example, if a service guarantees that the response time would be less than three seconds with 0.5 fuzziness and 90% confi-
dence, this means that at least 90% of the requests should respond in less than 3 seconds, while a maximum of 10% requests can respond within 3 to 3.5 seconds without breaking the guarantee. Also, to cater for the business demands in the cloud domain, CSLA defines the possibility of using services on a pay-as-you-go basis, in addition to the traditional fixed-price billing mechanism. As for drawbacks, the SLA parameters and metrics defined in CSLA are limited for cloud applications, and it lacks support for any type of brokerage or negotiation to dynamically create SLAs based on consumers' requirements.

With the emergence of the microservice paradigm, RESTful APIs are being established as an efficient way to integrate services and build applications [Harms et al., 2017]. Generating accurate documentation that describes the behavior and capabilities of APIs as well as the pricing plan is useful to promote services that fit into the potential user’s needs [Gámez Díaz et al., 2018]. iAgree and SLA4OAI are two JSON-based languages describing usage plans (e.g., flexible cost, functionality restrictions, and invocation rate limits) of microservices in a vendor-neutral way. iAgree aims to model SLAs in RESTful APIs (i.e., computational services) or business processes (i.e., human services), while SLA4OAI only focuses on modeling SLAs in RESTful APIs, which specify limitations such as quotas and rates using standard OpenAPI Specification (OAS). An OAS document can be extended with an optional “x-sla” attribute specifying a URI that points to the SLA4OAI document. The core elements in iAgree and SLA4OAI specifications are context and terms. Context holds the information related to parties, SLA validity, SLA definitions (e.g., log, schema, and scope) and SLA management infrastructures. Terms contain a set of SLA terms including pricing, metrics, guarantees, configurations, quotas, and rates. However, no studies have investigated whether the specifications are expressive enough to describe SLAs in different domains. Also, the proposed SLA-driven API development lifecycle does not consider the negotiation process that automatically creates an SLA.

In the IoT domain, the WSLA specification has been integrated with the WSN constraints to specify the network performance guarantees for WSN operators [Gaillard et al., 2014b]. The extended WSLA-based WSN Nodes Specification (WSN-SLA) defines each sensor node with an identifier, its 3D location and a list of features. Based on the locations and
features, these nodes are further grouped into sensor sets so that a specific collection of sensors can be identified with a given QoS. To support the automated QoS monitoring and analyzing, two basic metrics that express the performance of WSN are defined: message counters and message delays. This specification focuses on SLA modeling in the bottom layer (i.e., sensing layer) rather than the top layer (i.e., service layer or interface layer), while service consumers may be additionally interested in service-oriented aspects rather than just the concrete device information. The authors further proposed an SLA management framework to monitor the state of WSN in a centralized way, which helps to guarantee the QoS parameters such as delay, packet loss rate, traffic load, and residual energy [Gaillard et al., 2014a]. The new admission of SLA requests is controlled by the framework based on the current state, remaining capacity and physical topology of the WSN. However, this framework relies on human intervention to finish the negotiation process.

The Multi-Level Service Level Agreements (ML-SLA) is an SLA language designed for flexible IoT services, which addresses the demand for flexible changing of service quality and pricing at runtime without a renegotiation process [Grubitzsch et al., 2017]. ML-SLA defines an SLA with parties and SLOs. It can be automatically created by following the single round negotiation protocol specified in WSAG. Different from the semantics of SLOs specified in WSAG, it defines an extended notion of SLOs to express multiple objective levels within a single SLA. By using a multi-level pricing model and a pre-defined level switching policy, this extended SLO schema supports the dynamic change of service levels with a price adjustment, which guarantees the service continuity with less time and effort compared to SLA renegotiation mechanism.

Recently, a JSON-based SLA specification for end-to-end IoT application has been proposed to accommodate IoT’s multilayered structure [Alqahtani et al., 2019]. The proposed SLA-IoT specification is built based on a layered reference architecture for IoT, which is composed of IoT device layer, edge computing layer, cloud computing layer and the IoT application layer. The device layer contains devices that sense and reflect the physical world. The edge computing layer moves lightweight computation tasks (e.g., data collection and filtering) to edge devices to improve performance and reduce unnecessary data transfer to
cloud data centers. The cloud computing layer provides large and scalable hardware infrastructures for performing computation-intensive tasks such as stream/batch processing and big data analysis. According to the reference architecture, it identifies five entities in the conceptual SLA model, which are SLA (i.e., agreement name, id, validity, and application type), party (i.e., individual or group involved in the SLA, may include an end user and multiple service providers across different IoT layers), SLOs that quantifies the required value of a QoS metric, workflow activities that have to achieve the business goal of an IoT application (e.g. capturing patients’ biological data, analyzing real-time data, etc.), services that need to cooperate with each other to achieve SLOs at the application level\(^5\), and infrastructure resources on which the services are deployed or executed. This end-to-end SLA specification requires users to select the workflow activities based on their business goals, map the workflow activities with required services, and specify the QoS requirements for involved services and infrastructure resources. This SLA creation mechanism implies that a set of workflow activities needs to be pre-defined by domain experts using a standard ontology to achieve semantic interoperability. Also, the manual selection of services that correspond to the required workflow activities is not a flexible solution, since the services may not fully satisfy users’ requirements, and it lacks a negotiation mechanism to resolve the possible preference conflicts across different layers.

2.2.3 Service Abstraction in the IoT environment

Towards automating SLA management in the IoT environment, the first step is to model SLAs based on the abstraction of IoT services. Since service discovery, negotiation, monitoring, and resource allocation rely on the service information specified in the SLA [Alqahtani et al., 2019, Marino et al., 2019], abstracting things or their measurements as services is required for SLA modeling in IoT environments. There is limited research related to SLA specification in the IoT domain, and so IoT service abstraction from the large, distributed and heterogeneous resources is discussed as a basis to draft SLAs with respect to describing functional and non-

\(^5\)For example, a remote health monitoring application requires an ingestion service and a stream-processing service to transfer data from sensors to edge/cloud infrastructures, and analyze data on the fly
In WSN, the most common pieces of information available from sensor platforms are sensor type, sensor location, and the data they generate [Borges Neto et al., 2015]. Current research has developed several ontology models to describe sensors and their observations. Kim et al. presented a high-level ontology model to describe sensing data from a service-oriented perspective, which consists of service property (e.g., humidity, temperature, pressure), location property (e.g., latitude and longitude) and physical property (e.g., device information) [Kim et al., 2008]. These properties outline users’ expectations rather than just focusing on the physical sensor information described by a legacy sensor ontology. The SENSEI project [Tsiatsis et al., 2010, Villalonga et al., 2010] provides a framework for wireless sensor and actuator networks that enables sensor data, actuators and processors, which are called as Real World Entities, accessible through services. To support dynamic service composition and instantiation, the entities are modeled as resources whose functionalities and accessing information are stated in the resource descriptions published in a resource directory. The semantic ontology associated with the resource includes resource description, resource type, location, temporal availability, semantic operation description (e.g., input, output, preconditions, post-conditions), observation area, quality, and cost. The SSN ontology [Compton et al., 2012] is a high-level model that describes sensor devices, measurement capabilities and related attributes in sensor web applications, which is further extended by OpenIoT [Soldatos et al., 2015] and IoT-Lite [Bermudez-Edo et al., 2016] to semantically annotate sensor data. The SSN ontology can be seen from the sensor perspective, observation perspective, system perspective, and feature/property perspective. The sensor perspective focuses on the measuring capability of sensors under various conditions such as its stimulus, sensing method, and the observations it makes. The observation perspective interprets observed data and related meta-data. The system perspective focuses on systems of sensors, operating/survival restrictions, and deployments (e.g., platform, deployed location and temporal features). The feature/property perspective outlines the devices and observations related to a particular property. Based on the SENSEI project and SSN ontology, a data model that describes the entities, resources and service models was proposed to provide interoperability in service
levels [De et al., 2011]. In the data model, entity constitutes things in the IoT environment, which could be a human, an embedded device, a building or a closed/open environment. The software component that provides information on the entity or controls the attached devices is defined as a resource. A service exposes the functionality of a device by accessing its hosted resources, which provides a standardized interface to support the interactions with entities or related processes. According to the Ontology Web Language for Services (OWL-S), semantic web services are described by the service profile, service model, and service grounding [Martin et al., 2004]. The service profile can be used to facilitate service discovery and composition, which contains the service’s inputs, outputs, preconditions (i.e., the conditions to invoke an actuation service), effects (i.e., desired state after execution), observation area and observation schedule. The service model describes the service’s operation. The service grounding specifies the technical details to invoke a service, such as the service endpoints and message types. Figure 2.5 shows the relationships between these concepts, and how the properties described in the service profile link the service model to the entity model.

Usually, the non-functional service properties may depend on the requirements of domain-specific applications. For example, the typical QoS indicators of grid services are the availability, failure recovery times and response time [Waldrich, 2011], while for cloud services, availability, usability, reliability, responsiveness, security, elasticity, request rate are commonly used as key indicators [Zheng, 2014, Faniyi and Bahsoon, 2015]. To support the QoS-aware
service provisioning in the IoT domain, existing research also proposed a set of QoS metrics for IoT services. Marie-Aurélie et. al. built three service models (i.e., open/supple/complete service model) based on interactivity (i.e., supported or unsupported), delay (i.e., non-real-time, soft real-time or hard real-time), and criticality (i.e., yes or no) [Nef et al., 2012]. The open service model represents the services that are interactive, non-real-time and non-mission-critical (e.g., augmented maps, social networking, smart gym, etc.). The supple services model represents the mission-critical soft real-time services that may or may not be interactive (e.g., environment monitoring, industrial plants, authentication, etc.). The complete service model represents the non-interactive mission-critical services that are soft real-time or hard real-time depending on the application scenario (e.g., assisted driving, healthcare tracking, etc.). Similarly, Duan divides IoT applications into four types based on the quality levels they require [Duan et al., 2011]. This work proposed a layered QoS architecture for IoT applications, which consists of the perception layer, network layer and service/application layer from bottom to top. QoS in the perception layer reflects the quality of sensing, which is indicated by sampling parameters (e.g., sampling frequency, sampling precision, and data transmission rate), coverage/location, time synchronization, and mobility. QoS in the network layer reflects the quality of data transmitting, which is evaluated by indicators including bandwidth, delay, packet loss rate, and jitter. QoS of the service/application layer reflects the quality levels received by end users, which has attributes including service time, service delay, service accuracy and service priority (i.e., differentiated quality levels). This work highlights the necessity of translating QoS requirements from the upper layer to the lower layer to guarantee an end-to-end QoS. Due to the heterogeneous technologies and resources existing in the IoT environment, this work proposes the possibility of QoS negotiation across different layers. It envisions that QoS brokers would set up in the network layer and perception layer, which are responsible for QoS translation, QoS monitoring and possibly, conflicts resolution through negotiations if the requirements of the upper layer can not be satisfied after performing the resource allocation and scheduling mechanism in the local layer.

To address the QoS requirements of cloud-based IoT applications and resource limitations of mobile devices and sensors, the OpenIoT platform proposes a QoS manager whose
main objective is to identify the best data sources (i.e., sensor nodes) that fit the end user’s request, while achieving energy efficiency by deactivating currently redundant sensor nodes [Martina Marjanović, 2014]. The resource management and utility-driven optimization mechanisms make use of a set of utility metrics related to physical level (i.e., quality of sensors, energy consumption, bandwidth of a sensor, data volume, and trustworthiness), sensor network level (i.e., system lifetime, latency, delay, bandwidth, throughput, hop count, etc.), and application level (i.e., reliability, scalability, survivability, confidentiality, relevance) [Calbi-monte et al., 2014]. These utility metrics can serve as a basis for SLA accounting and management between the OpenIoT cloud service providers and end users.

To support the real-time analysis of massive data and sensory information produced by various data sources, the CityPulse project proposes an information model that represents the summarisation and reliability of streaming IoT and social media data for smart city applications [Tönjes et al., 2014]. The concepts that are used to assess the quality of stream data include cost, timeliness, communication, security, and accuracy. Each quality concept consists of a set of quality parameters that are defined by corresponding measurement units and value ranges. For example, accuracy can be measured in terms of precision, completeness, and correctness, which are assessed by the data resolution and deviation, the probability that datasets contain valid information and be updated in pre-defined frequency, and the degree of conformity to the specified precision and completeness.

2.2.4 Smart Contract

Since the service management mechanism controlled by providers is highly non-transparent for the consumers in service-oriented computing environments, enabling automatic SLA monitoring and billing mechanisms that verify SLA compliance and compensate for a customer’s loss if an SLA violation occurs, are required for distrusted trading parties. Recent research has envisioned the possibility of leveraging smart contracts to automate the SLA management without the arbitration of a trusted authority.

One example is Blockchain, which is an append-only ledger based on a distributed database and harsh chain working on a P2P network. Blockchain removes trusted intermediaries by
adopting a set of security mechanisms such as decentralizing and replicating the data through-
out all nodes in the network, using cryptography and a consensus mechanism to prevent the
data from being maliciously changed or deleted [Hari and Lakshman, 2016]. The smart
contract is an important element of blockchain technology, and is a self-executing script run-
ing on a blockchain to automate agreement enforcement in a trustless environment [Rode,
2017, Alharby and Van Moorsel, 2017]. Rafael Brundo et al. outlines the advantage of man-
aging SLAs of cloud services using smart contract technology [Uriarte et al., 2018]. First,
it supports the environment where trading parties do not need to trust each other. Second,
by transforming SLAs into public smart contracts and deploying them in the blockchain, the
service can be monitored by random entities in the network and assessed under a proof-of-
concept consensus protocol, which reduces the chance of cheating. Zhou et al. proposes a
similar idea that smart contracts can be used to detect SLO violations in a trustworthy way.
A witness is designed as a new role in the supporting party, who earn rewards by behaving as
an anonymous service monitor. The witness can be any user of the blockchain, who registers
its wallet address in the witness pool. This work assumes a service provider first negotiates
service properties with a consumer off-chain, then creates a smart contract specifying the
service detail and randomly selected witness committee. To solve the trust issue of the wit-
nesses, a game theory-based payoff function for different actions was designed to guarantee
that only honest behaviours can get the maximum profit. Once a violation event is approved
by the witness committee, the smart contract automatically issues the compensation fee for
the consumer.

In the IoT domain, a large number of connected devices spread sensitive personal data
and reveal behaviours and preferences of device owners, which puts users’ privacy at risk if
these data are managed by centralized companies [Zyskind et al., 2015]. Blockchain tech-
nologies can help to detect data abuses and define access policies without entrusting user
data to centralized companies [Conoscenti et al., 2016]. Also, a blockchain network where
cryptocurrency is exchanged provides a convenient billing layer. Integrating blockchain into
the IoT environment facilitates the sharing of services and resources, which leads to the cre-
ation of a service marketplace between devices [Christidis and Devetsikiotis, 2016] and secure
share economy applications [Huckle et al., 2016]. To support a data transaction in the IoT environment without the help of any other third party, an E-business model based on smart contracts and encrypted coins was proposed [Zhang and Wen, 2017]. This model assumes sellers create smart contracts in advance, specifying their offerings and publish them on the Blockchain. Anyone in the network can accept this contract within the time. If both parties sign the smart contract with their private key, the contract becomes undeniable, and the transaction is issued according to the pre-defined conditions.

To our best knowledge, research on managing SLAs using smart contract technologies is still limited, especially for the IoT domain. Despite the benefits suggested in existing works, the challenges related to online service matching, automated service negotiation, data privacy [Uriarte et al., 2018] and semantic interoperability [Zhang and Wen, 2017] still remain. Also, the storage capacity and scalability of blockchain are still under debate, as the large amounts of data produced in IoT environments is a big challenge for IoT-blockchain integration.

2.2.5 Assessment

Service modeling and SLA definition are two necessary phases in the SLA lifecycle. The WSAG and WSLA are two prominent web service SLA specifications that provide the basis for SLA modeling in different computing environments. Although WSLA specification precisely describes how SLA parameters are aggregated from measurable metrics, WSAG specification is more comprehensive than WSLA in terms of defining services and guarantees. For example, WSAG contains scopes for which the guarantee holds, conditions under which the guarantee is valid, and a business value list including penalty and rewards. This implies that service providers do not only state guarantees regarding capabilities, but also have a chance to express their own requirements such as the constraints on requesting rates or dependent services. Considering the layered QoS architecture of IoT applications, the semantics of guarantees defined in WSAG is more suitable to model the dependency of SLOs in different IoT layers.

Based on these two specifications, several SLA languages have been proposed to address domain-specific requirements of different application scenarios. However, few of them cap-
ture the characteristics of IoT services, and the support for automated SLA negotiation is very limited. With regard to the SLA specifications designed for the IoT domain, WSN-SLA extends the WSLA specification to capture sensor nodes, and focuses on SLA modeling on the sensing layer; the SLA-IoT specification targets SLA creation for end-to-end IoT applications (i.e., SLA modeling on the application layer), but needs a set of pre-defined workflow activities describing business goals of IoT applications; while the ML-SLA language captures information about the service delivery with flexible quality levels that complies with a user’s dynamic requirements at runtime. However, none of these IoT SLA specifications provide a flexible SLA creation mechanism. The ML-SLA uses the simple unilateral negotiation model specified in WSAG to create an SLA, but the multi-round bilateral negotiation is not supported. Neither SLA-IoT nor WSN-SLA supports any kinds of SLA negotiations. SLA-IoT requires users to manually select the workflow activities and map them to required services and infrastructure resources to create an end-to-end SLA. WSN-SLA assumes SLAs are created from manually performed negotiations.

To abstract IoT devices and their measurements from a service-oriented perspective, current research has developed a set of ontology models or utility metrics that describe the functional and non-functional features of different IoT layers, which can be used to model SLAs for IoT services by extending the existing SLA specifications with domain-specific concepts derived from them. To address the trust issues during an SLA management process, recent research envisions the possibility of transforming SLAs to smart contracts that are automatically executed on blockchains. Using a consensus protocol, distributed ledger and cryptography technologies implemented in blockchains, it is difficult to maliciously modify or delete smart contracts, and the verification of SLA compliance can be performed without a centralized arbitral authority. However, implementing smart contract-based SLA management in the IoT environment is challenging for heterogeneity and scalability reasons. For example, transforming service management rules specified in SLAs to smart contracts requires a standard ontology describing the format and semantics of SLAs, and an efficient lightweight consensus protocol is needed to decrease the latency of transactions in IoT environments.
2.3 Automated SLA Negotiation

As described in Chapter 1.1, negotiation is an important SLA creation mechanism in capacity-aware SOA that encourages self-interested entities to announce individual information and coordinate with each other to achieve a global beneficial agreement as well as establishing the most favorable business relationship. This section describes the existing negotiation frameworks along with the corresponding protocols or strategies they use.

2.3.1 SLA Negotiation Protocol

To automate the SLA negotiation and creation process, many agreement-based SOA middlewares use the WSAG specification because it is extensible. For example, Mobach et. al. presented a negotiation infrastructure for mobile agent applications deployed on Internet-sale distributed systems, which uses a WSAG-based protocol to negotiate a time-limited contract with autonomous hosts that provide resources to agents under specific usage and access policies [Mobach et al., 2001]. Valentina et. al. proposed a centralized Cloud Sensing Broker Platform that models sensor networks as providers who offer their sensing infrastructure to external cloud applications under a signed SLA [Casola et al., 2013]. Providers and users register to the platform by advertising their demands and supplies using the structure specified by WSAG. The platform acts as a broker that returns to the user a list of candidate providers based on registered information and prepares an SLA for the user and the candidate provider selected by the user according to their specified constraints. Once the SLA is signed by both parties, the service is monitored at runtime by the platform to verify SLA compliance. Since the requirements of the business participants may change over time in a dynamic environment, to avoid the suspension of service provisioning and the unexpected SLA termination during the service operation time, Di Modica et. al. extended the WSAG specification to capture run-time re-negotiation that can adjust the QoS guarantees stipulated in the current SLAs [Di Modica et al., 2007]. Similarly, Michael et. al. proposed a domain-independent SLA re-negotiation protocol based on the WSAG and the principles of contract law to support multi-round re-negotiations in a network environment, where messages may be lost, delayed,
duplicated and re-ordered [Parkin et al., 2008].

Since the basic "take-it-or-leave-it" protocol defined in WSAG only allows consumers to accept or reject the conditions proposed by service providers without any further negotiation step, the WS-Agreement Negotiation specification (WSAN) was proposed as the extended negotiation layer on top of WSAG [Waeldrich et al., 2011]. The WSAN consists of three parts: the offer exchange model, the layered architecture model, and the schema of negotiation offers/templates conforming to the WSAG. The offer exchange model defines the dynamic interactions between a negotiation initiator and a negotiation responder, which can be generalized as follows: the negotiation initiator (e.g., consumer) starts the bilateral negotiation by initiating a new negotiation process with the responder (e.g., service provider). Then the initiator retrieves negotiation templates from the negotiation responder, creates an initial negotiation offer based on the preferred template, and sent the initial offer to the responder. The responder creates counter offers based on the constraints and states specified in the previously received offer and replies them back to the initiator. Negotiation offers are exchanged between negotiation parties until an agreement can be created. The layered architecture model describes the decoupled negotiation layer\(^6\) on top of the agreement layer and the service layer. Once an agreement is reached, the negotiation layer relies on the lower layers to deal with SLA creation and monitoring. Also, WSAN supports the renegotiation of an existing agreement. Technically, SLA renegotiation is still a bilateral bargaining between an initiator and a responder. Nevertheless, the renegotiation process must specify an endpoint reference of the original agreement that is going to be renegotiated. When a renegotiated SLA is successfully created, the state of the original SLA changes to complete.

Based on the WSAN specification, the BETaaS platform includes a QoS negotiation component for M2M applications [Mingozzi et al., 2014]. The BETaaS platform is deployed on top of a distributed architecture made of a local cloud of gateways. Physical objects are connected to local gateways, which can be accessed by applications through a service-oriented interface exposed by the platform. The BETaaS platform has a three-layer architecture in

\(^6\)The decoupled negotiation layer means the offers and counteroffers are only used as instruments for negotiation parties finding a mutually acceptable agreement, which does not imply a promise of the corresponding party.
which things provide their functionalities by *thing services*, each *service* relies on one or more *thing services*, and an *application* is the final consumer of *services*. Based on the QoS requirements, services are classified into three types: *realtime services*, *assured services* and *best effort services*. To ensure the fulfillment of the service level requested by applications, a cross-layer negotiation mechanism is designed, which stipulates that the *service* layer must negotiate with the underlying *thing services* before negotiating QoS parameters with *applications*. Although the BETaaS platform relies on WSAN to formalize the process of SLA creation, it only supports unilateral interaction, where responders can not propose counteroffers, sending only confirmation or rejection messages.

Apart from WSAN, the *Alternate offers protocol* (AOP) proposed by Rubinstein is another bilateral negotiation protocol, which can be used for multi-round bargaining where both parties alternately exchange offers and counteroffers until they come to an agreement or terminate the negotiation process [Rubinstein, 1982]. This protocol has been extended for different negotiation scenarios such as argumentation-based negotiation [Hadidi et al., 2010], multilateral negotiation [Aydoğan et al., 2017], or semantic-based negotiation [Ragone et al., 2007]. However, it is a simple proposal exchange protocol that does not place a limit on the negotiation deadline.

Recently, the *Contract Net Protocol* (CNP) that was initially defined for distributed task allocation has been used for automated contract negotiation in IoT environments. The CNP is proposed through a use case that in a distributed sensing system, sensing nodes and task manager nodes with extensive processing capabilities express their own disparate viewpoints and seek for a solution through negotiation [Smith, 1980]. The task manager node specifies a task using CNP, which includes task abstraction, eligibility specification, bid specification, and an expiry date. Each eligible potential contractor node submits a bid message and waits for a contract award message if its proposal is the most optimal one for the manager node. Misura *et. al.* integrated the CNP into a cloud-based mediator platform to provide sensing data to IoT applications on-demand [Mišura and Žagar, 2017]. In this platform, devices are registered by posting an XML-based device description that specifies the device ID, measurement type, location, and negotiation strategy (i.e., MinPrice or TwoTariffs). The
applications are registered by posting an XML-based application description specifying the expected schedule, requested area, measurement type, and negotiation strategy (i.e., Fixed-Budget or ByPriority). The platform selects candidate devices based on the requirements and creates corresponding negotiation agents. The agents perform multi-bilateral negotiations using CNP to find the conditions of data provisioning that are acceptable for both parties. Once an agreement is reached, a contract is created accordingly. However, disclosing negotiation profiles to public may be impractical in a competitive market.

In the agent community, FIPA Iterated Contract Net Interaction Protocol is a simple negotiation protocol supporting one-to-many negotiation scenarios [Fip, 2001], which have been used for distributed contract negotiation and composite service negotiation. Marsá et al. assume that a multi-agent system is deployed on wirelessly connected personal devices, which act on behalf of users [Marsá Maestre et al., 2005]. The authors developed an interaction protocol based on FIPA that allows a group of agents to pursue a set of public global goals through negotiation in a fully distributed manner. The idea is similar to the distributed ledger in the blockchain. The initiator sends a call for proposals to all other agents, and each agent is allowed to respond with zero, one or more proposals. The judgment of each issued proposal is shared by all agents so that it is possible to reach to an agreement without a privileged entity that makes the final decision. A drawback of this negotiation model is the high risk of network overload since each message needs to be broadcasted to other entities. The SLA-supported composite service provisioning requires service consumers to achieve a set of interrelated agreements with various services providers that collectively fulfill the end-to-end QoS requirements. The ASAPM project [Yan et al., 2007] presented an SLA negotiation framework in which the service consumer is represented by a set of agents who negotiate QoS constraints with the service providers for various services in the composition. The agents exchange negotiation messages with service providers in a coordinated way to ensure the end-to-end QoS. This coordinated negotiation process is controlled by an extended FIPA protocol. During the negotiation process, a decision-making model based on a utility function is proposed for agents to evaluate the received offers and take corresponding actions. Based on a fuzzy similarity between the proposal and the current offer that indicates
the chance of reaching to an agreement, agents can either make concessions or trade-offs on
the previous proposals to encourage the service providers to accept their counter proposal.

Pawel et. al. proposed another approach that enables the QoS-aware composite service
delivery in a distributed IoT system [Swiatek and Rucinski, 2013]. The end-to-end QoS re-
quirement is secured by the negotiation procedures between two arbitrary atomic services on
the execution path, which is started by the service that wants to send its data to another
service. This negotiation procedure is controlled by a lightweight protocol, which defines the
message types and corresponding actions including request, accept, reject, acknowledge, rene-
gotiate and ready. However, this approach requires a centralized server to manage negotiation
tasks and confirm negotiation results.

2.3.2 Negotiation Strategy

Negotiation can be regarded as a distributed search in the n-dimensional agreement space
\(n\) is the number of terms over which an agreement must be achieved) [Jennings et al.,
2001]. For each involved party, only a sub-region of the agreement space corresponds to an
advantageous agreement. The negotiation process aims to find a mutually acceptable solution
that lies within the overlapped sub-regions. To achieve this, different negotiation strategies
have been proposed.

To dynamically make concessions during the negotiation process, Faratin et. al. pro-
posed three types of negotiation tactics: time-dependent tactic, resource-dependent tactic,
and behavior-dependent tactic [Faratin et al., 1998]. They concluded that there is a trade-
off between the number of deals and the utility gained. Similarly, Yao et. al. proposed
a negotiation strategy that reacts to the changing environment [Yao and Ma, 2008]. This
strategy makes concessions using a utility function, which considers the effect of remaining
time, available resources and proposals from the opponents. The acceptability of each offer
is determined by satisfaction degree, restriction obedience, and obtained utility. Although
this strategy employs fuzzy truth propositions to specify the constraints of negotiation par-
ties, it is not clear how the constraints are modeled, and the truth function is not specified.
Fharna et. al. proposed a policy-based negotiation strategy consisting of a policy-mapping
model, an adaptive algorithm, and a strategy selection algorithm [Zulkernine and Martin, 2011]. They specified three types of decision functions based on the time-dependent tactic. According to the performance observations of different time-based decision functions, the negotiation agents dynamically adapt the decision functions during the negotiation process to comply with an opponent’s preferences. However, this adaptation may be inefficient when the strategy adopted by the counterpart is unknown. To balance the success rate and negotiation utility when negotiating participants conceal their complete negotiation information (i.e., negotiation with incomplete information), Zheng et al. proposed a game-theory based strategy that combines the concession and tradeoff tactics to resolve possible conflicts, which demonstrates a good balance between utility and success rate under Monte Carlo simulations [Zheng et al., 2014]. However, this approach may fail to find a solution when one exists. To support composite service negotiation, Jan et al. presented an SLA negotiation strategy that decomposes the global utility boundary into atomic service utility boundaries and redistributes the surplus from successful negotiation outcomes among the remaining negotiations [Richter et al., 2012]. The proposed algorithm coordinates concurrent service negotiations within complex workflows by enabling the iterative and interactive adjustment of the negotiation boundaries for each atomic service in a composition.

To optimize utility for negotiations with incomplete information, some approaches aiming at learning opponents’ preferences or reserved values have been proposed. For instance, Faratin et al. used the fuzzy similarity to approximate an opponent’s preference, and the hill-climbing algorithm is integrated to detect a tradeoff offer that might be acceptable by the counterpart [Faratin et al., 2002]. However, this approach highly depends on the availability of prior knowledge, which is unrealistic in a dynamic environment such as the IoT. Coehoorn et al. adopted kernel density estimation to estimate an opponent’s preference on negotiated issues [Coehoorn and Jennings, 2004], but it assumes that the opponent only employs a time-dependent tactic. Zhang et al. introduced a bilateral negotiation strategy for coal tradings, which uses Bayesian learning to reduce negotiation time and improve bid efficiency. However, this model only considers a single negotiation issue (i.e., price) and assumes negotiation parties have prior knowledge about the opponent’s learning objective, which will be gradually
modified through the negotiation process. Lin et al. proposed a Bayesian learning-based strategy that uses a reasoning model to learn the likelihood of an opponent’s profile [Lin et al., 2008], which also assumes that a set of possible opponent’s profiles are known a priori. Similarly, [Yu et al., 2013] presented a Bayesian learning model for the single-issue negotiation by assuming negotiators have a prior belief about the probability distribution of their opponents’ negotiation constraints (i.e., the deadline and reserved value). By comparing an opponent’s offers with the offers derived from regression analysis, negotiators revise their beliefs using the Bayesian inference and correspondingly adapt their concession strategies for more benefit. Narayanan et al. use a Markov chain to model bilateral negotiations among agents, and Bayesian learning is adopted by agents to learn an optimal strategy [Narayanan and Jennings, 2006]. Carbonneau et al. created a three-layer neural network that exploits information from past counteroffers to predict an opponent’s future proposals [Carbonneau et al., 2008]. However, this approach requires a large amount of historical data, and possible changes in the opponent’s strategy over time is ignored. Apart from decision functions and machine learning techniques, metaheuristic algorithms are also applied to making concessions. Sim et al. combined Bayesian learning with Genetic Algorithms (GA) to search for the optimal strategy when negotiating with incomplete information [Sim et al., 2009]. But this approach is not suitable for multi-issue negotiation, and the success rate is unsatisfactory when the deadline range is lower than 30 negotiation rounds. Alkayal et al. presented a bilateral negotiation model that adjusts proposals by using particle swarm optimization. The counteroffer proposed by the opponent is modeled as wind, which controls the particles’ movement in each round [Abulkhair et al., 2017]. Also, this model requires multiple negotiation rounds to approach to the optimal solution.

2.3.3 Assessment

As a de-facto standard for web services SLA negotiation, WSAN specification defined an offer exchange model for bilateral negotiations, which can be performed asymmetrically (i.e., simple client-server mode) or symmetrically (i.e., peer-to-peer mode). This process indicates that consumers know whom they are going to negotiate with and how to contact them when
requesting SLA templates. We refer to this situation as *known participants*, which is not practical in large-scale IoT environments. The current existing IoT mediator platform [Mišura and Žagar, 2017] addresses this problem by providing a registration component, which uses the registered device/requirement descriptions to identify candidate service providers before negotiation starts. However, this mechanism does not consider the presence of autonomous service providers who may withdraw their offerings at any time. We refer to this situation as *unknown static/mobile participants*. Since the computation complexity increases linearly with the number of negotiation sessions, the centralized architecture may bring scalability problems in IoT environments considering the scale of localized sensors deployed in different IoT platforms and the possibly large number of devices providing the same or similar functionalities.

Existing negotiation frameworks have proposed a centralized negotiation protocol (i.e., client-server mode), partially distributed negotiation protocol (i.e., negotiation tasks are controlled by a single entity) and fully distributed negotiation protocol (i.e., no centralized controller, which can be referred to as decentralized negotiation). Decentralized negotiation outperforms the other two in terms of scalability and fault-tolerance, but increases network consumption [Marsá Maestre et al., 2005]. Existing distributed negotiation models either rely on a server to manage the negotiation tasks [Swiatek and Rucinski, 2013] or broadcast messages to all participants to reach to an agreement [Marsá Maestre et al., 2005]. They are insufficient to address the IoT-specific communication issues introduced by mobile entities and the limited-range of many wireless networks.

Current negotiation strategies have defined a set of mathematical models to make decisions during negotiations, which is usually composed of utility functions and decision functions. The utility function quantitatively measures the satisfaction level of each offer, while the decision functions control the concession rate in each negotiation round. However, existing utility functions do not capture the important features of IoT services, such as spatial and temporal properties. For negotiation with incomplete information, some approaches use machine learning techniques to predict the opponent’s negotiation profile, negotiation strategy, or negotiation deadline. However, they are either computationally expensive for resource-
constrained IoT devices (e.g., using Bayesian learning for multi-issue negotiation), or ignored the dynamicity of the opponent’s behaviour (e.g., proposal prediction using a pre-trained neural network). For negotiations using metaheuristic algorithms, the common disadvantage is the multiple negotiation rounds that are required to find the optimal solution. Also, GA needs a coding mechanism to transform each possible offer to a real number, which increases the computation consumption. Considering the unstable wireless connection and the massive interactions between devices and devices/clouds in the IoT environment, a deadline-aware negotiation strategy should be developed to reduce the number of timeout failures.

2.4 Selecting Negotiation Candidates

A bilateral negotiation process has the potential to cause significant latency and a massive number of interactions. For this reason, negotiation with a service provider that has very little chance to reach an agreement is not only unnecessary and time-wasting but also introduces extra traffic that does not contribute to the negotiation utility at all. To increase negotiation efficiency in a time-constrained negotiation scenario, a service provider selection is a necessary phase in the SLA negotiation model to reduce the number of negotiation candidates [Elfatatry and Layzell, 2004]. Also, making a trade with a misbehaving service provider may end up requiring expensive provider switching and a service migration process [Itani et al., 2014].

2.4.1 Methods for Service Provider Selection

Service provider selection usually depends on service match-making [Wang et al., 2006, Chatterjee and Misra, 2016] or reputation assessment of service providers [Tang et al., 2017, Wang and Vassileva, 2007, Vu et al., 2005]. Service match-making compares requests with services’ properties to select the set of candidates that satisfy consumers’ needs, while reputation assessment provides a complementary mechanism that reduces the risks of choosing dishonest or incompetent providers [Billhardt et al., 2007]. Considering the possible large search space in the cloud environment, Sunderes et. al. firstly indexed cloud services based on their common set of properties, then used the K-Nearest Neighbor algorithm to select the services that
satisfy the user’s requirements [Sundareswaran et al., 2012]. Chatterjee et. al. proposed the QASeC algorithm to automatically select a cloud service provider for naive users in the IoT environment, based on the maximum achievable QoS [Chatterjee and Misra, 2016]. This algorithm built a utility metric based on the mathematical quantification of different QoS types and performed Lagrangian Multiplier optimization to allocate the cloud gateway that maximizes the QoS utility metric with respect to a particular user. Jin et. al. defined four QoS attributes that reflect features of physical services, and proposed a service selection algorithm to select optimal service based on aggregated QoS ratings and user’s preference [Jin et al., 2014].

For reputation-based provider selection, the reputation of a provider is usually calculated based on the aggregation of validated feedback from users [Már mol and Pérez, 2009] or evaluation results provided by third parties that are trusted by trading parties [Itani et al., 2014]. For example, Chen et. al. proposed a trust management model based on fuzzy reputation for wireless sensors. This model measures the trust of sensors by evaluating the QoS trust metrics including packet forwarding/delivery ratio and energy consumption [Chen et al., 2011a]. Saied et. al. proposed a trust management system for wireless sensor networks, which assigns dynamic trust scores to cooperating nodes according to different contexts and propagates the trust data using centralized servers [Saied et al., 2013]. Sonja et. al. proposed a distributed reputation system for misbehavior detection in mobile ad-hoc networks using both direct observations of each node and the compatible indirect recommendations provided by other nodes [Buchegger and Le Boudec, 2003]. Similarly, Nitti et. al. proposed a trust model in the social IoT environment by considering the social relationships of IoT device owners. In this model, each node computes a subjective trustworthiness of its friends by combining the past experience and the recommendations, and an objective trust deriving from P2P networks [Nitti et al., 2013]. The trust-based selection mechanism is also used in the existing cloud-based IoT mediator platform [Mišura and Žagar, 2017]. Credibility reflecting the trustworthiness of devices is proposed as a criterion for pre-selecting candidate devices. This platform keeps credibility on a per-owner basis using a rating mechanism that is similar to the one implemented on eBay. However, the authors do not specify how to check the
consistency of delivering data under the negotiated conditions, and how the QoS parameters that are useful to measure trustworthiness (e.g., correctness, reliability, safeness, effectiveness) is specified in a contract is unclear.

2.4.2 Assessment

In an open market where multiple third-party providers offer similar services, the possible existence of misbehaving services that perform malicious behaviours (e.g., overstating QoS guarantees during the negotiation stage to increase the chance of coming to an agreement) for their own benefit at the expense of other similar services should be considered. To improve negotiation efficiency and enhance the consumer’s satisfaction, selecting candidate service providers before starting a negotiation is necessary. Existing service selection approaches use service match-making or reputation systems to identify optimal service providers, which only considers services’ functionalities and execution performance. However, candidates selection for negotiation purposes requires choosing the providers that are more likely to make an agreement and conform to the negotiated results. In other words, the service match-making mechanism should be able to measure the similarity between requests and services when service properties are negotiable, even if the provider’s negotiation preference and constraints are unknown and varying as time. Also, to avoid choosing the dishonest service providers, an automated contract monitoring mechanism is needed to check the compliance of negotiated contracts, which provides the basis for reputation and trustworthiness assessment.

2.5 Chapter Summary

Adopting capacity-aware SOA in the IoT environment enables a reliable service delivery for mission-critical IoT applications. This agreement-driven service provisioning requires middleware technologies to identify the optimal candidate service provider and create an SLA by negotiating with the candidate on behalf of the consumer. This chapter analyses the works related to automatic SLA negotiations from two perspectives: SLA development and SLA negotiation frameworks. Due to the limited research on SLA modeling in the IoT domain,
existing IoT ontologies and QoS models that are helpful to define SLAs for IoT services are also discussed. Existing negotiation frameworks are analyzed from the negotiation strategy, negotiation protocol, and provider selection perspective. The representative papers and the issues they mainly addressed are summarized in Table 2.1.

Figure 2.6 presents the most relevant researches and to what extent they satisfy a set of criteria. These criteria are derived from the challenges identified in Chapter 1.2. The existing approaches are analyzed in terms of SLA specifications they are using, the negotiation model they follow, the communication environment they assumed, and the proposed negotiation
As illustrated in Figure 2.6, the current SLA negotiation standard WSAN is inadequate to address the negotiation problems in complex IoT environments since it only supports bilateral negotiations under the assumption that negotiation participants have prior knowledge about each other [Waeldrich et al., 2011], which is impractical given the scale and dynamicity of IoT environments. Also, its corresponding SLA specification does not capture IoT domain-specific properties. Approaches that model SLAs in IoT domains [Alqahtani et al., 2019, Grubitzsch et al., 2017] provide formal syntax to describe IoT service properties, but do not exploit the possibility of creating an SLA through bilateral or multilateral negotiations. One of the approaches uses unilateral negotiations to create offers, which only allows the agreement responder to accept or reject SLA creation requests. Existing negotiation frameworks in IoT environments [Mišura and Žagar, 2017, Marsá Maestre et al., 2005, Swiatek and Rucinski, 2013] have proposed several negotiation protocols/strategies and provider selection algorithms.
for different negotiation models, but lack a standard SLA description language to achieve semantic interoperability. Considering the large number of third-party service providers and the limited wireless network connection ranges, cloud-based centralized architecture increases the risks of single point of failures, timeout failures, and low scalability. However, using purely decentralized architecture may introduce a lot of unnecessary message transmissions since each gateway only has a partial knowledge about the environment. A hybrid architecture that uses multiple distributed infrastructures to manage negotiation tasks may help to balance the tradeoff between efficiency and network consumption, which are likely to be more suitable for IoT SLA negotiation. Another common disadvantage of existing negotiation approaches is that they do not address the communication problems introduced by autonomous service providers and mobile consumers (i.e., unknown mobile participants).

Figure 2.6 also demonstrates a shortage of negotiation strategies that consider services’ location information and concealed negotiation preference of opponents. A location-aware strategy is necessary for offer evaluations since location is an important feature of IoT services [Bermudez-Edo et al., 2016]. Although some strategies designed for cloud/web service negotiations employ machine learning technologies or meta-heuristic algorithms to adapt their concession rates when the opponent’s negotiation profile is unclear, these approaches are either too heavyweight for IoT SLA negotiation, or not flexible enough to adapt to dynamic changes in IoT environments. A lightweight location-aware negotiation strategy is required for negotiations with incomplete information. With regard to the provider selection mechanism, solutions that use a service match-making mechanism to identify candidate service providers only consider services’ functionalities and execution performance. This may not be practical in agreement-based computing environments where service properties can be dynamically tailored based on users’ different requirements. The reputation-based selection relies on third-party monitors or users’ feedback to evaluate services’ actual performance, which requires a dedicated monitoring mechanism instructed by an underlying agreement or a trust verification of the user to guarantee a fair judgment.

In summary, open gaps with the current solutions are:

- SLA creation in the IoT domain either requires high human intervention or a sim-
ple unilateral negotiation, which obstructs demand-driven service provisioning in IoT environments.

- Approaches do not consider the dynamic nature of IoT environments such as mobile entities, provider’s fluctuating negotiation preference and changeable network topology, which increases the risk of negotiation failures. Also, a pure centralized or decentralized architecture can not control the trade-off between negotiation efficiency and network consumption.

- Negotiation strategies lack a mathematical model that can quantitatively evaluate IoT service properties such as spatial and temporal features. Also, an adaptable concession tactic is required to balance the trade-off between negotiation utility and success rate.

- Provider selection mechanisms focus on evaluating services’ functional or non-functional features, which is difficult to capture when these features are negotiable.
Chapter 3

Design

The literature review in Chapter 2 identified a number of limitations on applying current SLA negotiation approaches in large and dynamic IoT environments. Based on the research gaps identified in Chapter 2.5, this chapter introduces iNegotiate, a distributed negotiation system designed to be deployed on edge devices, which creates SLAs that are mutually acceptable for both trading parties through automated multi-bilateral negotiations. This chapter first presents the design objectives of iNegotiate, from which a list of requirements is presented. Then it introduces the system model and the interactions between different components. It continues with a detailed description of iNegotiate, which is composed of the WIoT-SLA ontology, a distributed SLA negotiation model, a trust-based candidate selection mechanism and a deadline-aware strategy for IoT service negotiations with incomplete information. Finally, this chapter generalizes the contributions of iNegotiate.

3.1 Design Objectives and Required Features

According to the research questions identified in Chapter 1.4 and the state of the art analysis, a global design objective can be generalized as: designing a distributed SLA negotiation system for a large-scale dynamic IoT environment, which automatically tailors service properties with suitable service providers based on users’ requirements. This global design objective can be decomposed into the following features that correspond to the identified research questions:
Feature 1. **IoT context-based SLA modeling**
Automated SLA negotiation requires services to be specified in a uniform way, especially for the IoT services that are likely to have more negotiable domain-specific attributes (Challenge 3). An SLA data model should describe IoT services from functional and non-functional perspectives, and specify the information that is necessary for automated SLA negotiation and monitoring (RQ.1).

Feature 2. **Distributed self-adaptable negotiation framework**
IoT is a dynamic large-scale environment where the network topology, a service’s availability, and a provider’s negotiation preference may constantly change over time (Challenge 1 and Challenge 2). A distributed negotiation framework allows gateways to autonomously allocate/perform negotiation tasks without the management of a centralized controller. This framework should have a negotiation protocol that can address the scalability and communication problems introduced by the limited wireless network connection ranges and autonomous mobile negotiation participants, without consuming too much network resources (RQ.2).

Feature 3. **Negotiation-oriented candidates selection**
There will be a huge number of IoT devices that have the potential to engage in service provisioning. Negotiating with multiple service providers offering similar services is not only time-consuming but also increases the risk of network congestion (Challenge 1). The negotiation framework should have a provider selection mechanism to avoid unnecessary interactions in a bandwidth-limited wireless network, without reducing the consumer’s satisfaction (RQ.3).

Feature 4. **Time-constrained negotiation strategy for IoT services**
Business negotiations in an open market are likely to have incomplete information about negotiation opponents’ profiles. Due to the dynamic nature of the IoT environment, a negotiation session that takes a long time or multiple interactions increases the number of timeout failures and network overhead (Challenge 2). The negotiation framework should have a deadline-aware negotiation strategy that can balance a trade-off between negotiation
success rate and negotiation utility within the limited negotiation rounds according to different negotiation scenarios \((RQ.4)\).

3.2 System Environment

In a large-scale IoT environment, IoT entities can expose their devices’ functionalities and data as services, and register them in an open market to increase their potential for business. These services are provided to interested users on demand. To provide a certain level of control to consumers and increase the chance of successful trading, providers deliver services based on negotiated SLAs. To ensure the SLA conforms to the interests of both sides, a negotiation process is conducted between trading parties allowing them to resolve possible conflicts by dynamically adjusting their expectations. Since manual negotiation is time-consuming in the presence of a large number of consumers and providers, the automation of SLA negotiation is required in an IoT environment. To achieve this, providers advertise their SLA templates to nearby gateways. These templates are forwarded within the gateway network and registered in the gateways that close to the advertised service location. An SLA template can be regarded as a blueprint to create a valid SLA and SLA negotiation offer, which contains service description and negotiation information. These registered templates are stored in different gateways according to the advertised service locations. As Figure 3.1
shows, service consumers\(^1\) submit requests to the gateway network to retrieve negotiated pending SLAs\(^2\). To create such an SLA that can satisfy a consumer’s functional and non-functional requirements, negotiation gateways forwards the request to the gateways close to the requested service location. Gateways match the request with registered templates to identify candidate service providers and start multi-bilateral negotiations with the Top-K candidates to seek mutually acceptable solutions. The pending SLA is created based on the optimal negotiation result that gateways can achieve.

### 3.3 Design Decisions

To support these required features, this section generalizes a set of decisions that are encompassed in \textit{iNegotiate}. These decisions are classified in terms of SLA modeling, distributed negotiation model, trust-based candidates’ selection and deadline-aware negotiation strategy. The SLA modeling specifies the semantic annotations of an SLA, an SLA template and a negotiation offer. The negotiation model defines how the templates are organized in the gateway network, how the negotiation tasks are allocated to different gateways, and how the gateways perform negotiation tasks based on the pre-defined negotiation protocol. The process of selecting candidates improves negotiation efficiency by applying a trust-based assessment to candidates before negotiating with them. The negotiation strategy uses a deadline-aware decision-making model and an adaptable negotiation tactic to maintain the highest possible utility during bilateral negotiations. Figure 3.2 shows the design decisions that map to required features and \textit{iNegotiate} contributions.

#### 3.3.1 WSAG-based SLA Modeling for IoT services

Existing SLA specifications are designed for web services or cloud services, which do not capture IoT domain-specific properties. To facilitate automated SLA negotiation and monitoring, the following decisions were made to enable the definition of the \textit{WIoT-SLA} ontology that

---

\(^1\)A service consumer could be an end-user, a service-based IoT application, or a service that has dependencies on other services.

\(^2\)The pending SLA refers to the negotiated SLA that has been neither accepted or rejected by the consumer.
Chapter 3. Design

Figure 3.2: iNegotiate - Design decisions map

<table>
<thead>
<tr>
<th>Required Features</th>
<th>iNegotiate Design Decisions</th>
<th>iNegotiate Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. IoT context-based SLA modeling</td>
<td>1. Structuring SLAs based on existing standards</td>
<td>1. WSAG-based SLA modeling for IoT services (Section 3.4.1)</td>
</tr>
<tr>
<td>2. Distributed self-adaptable negotiation framework</td>
<td>2. Simplified negotiation offer schema based on WSAN</td>
<td></td>
</tr>
<tr>
<td>3. Negotiation-oriented candidates selection</td>
<td>3. Hybrid Distributed Architecture</td>
<td>2. Distributed SLA negotiation model in the IoT environment (Section 3.4.2)</td>
</tr>
<tr>
<td>4. Time-constrained negotiation strategy for IoT services</td>
<td>4. Location-based templates organization</td>
<td>3. Trust-based evaluation of negotiation candidates (Section 3.4.3)</td>
</tr>
<tr>
<td>5. Layer-based communication</td>
<td>5. Layer-based communication</td>
<td></td>
</tr>
<tr>
<td>6. Experience-based candidates assessment</td>
<td>6. Experience-based candidates assessment</td>
<td>4. Deadline-aware strategy for negotiations with incomplete information (Section 3.4.4)</td>
</tr>
<tr>
<td>7. WSAN-based decision-making model</td>
<td>8. Adaptable negotiation tactic</td>
<td></td>
</tr>
</tbody>
</table>

describes IoT services and QoS guarantees in iNegotiate. The concrete WIoT-SLA ontology is described in Section 3.4.1.

**Design Decision 1. Structuring SLAs based on existing standards**

The heterogeneous nature of IoT services offered by various third-party service providers demands a flexible SLA language that can be extended with new elements to accommodate the requirements of a specific service domain. WSAG is a high-level structure for SLAs and SLA templates that must be extended with expressions suitable for an application domain [Maarouf et al., 2015]. Considering the popularity of WSAG [Kyriazis, 2013] and, in particular, its support for multi-round negotiations, WIoT-SLA models SLAs by extending WSAG with important concepts derived from existing IoT ontologies. Although WSAG specification is more comprehensive than WSLA in terms of defining services and dependencies of QoS guarantees [Maarouf et al., 2015], the semantics of QoS parameters
are not specified in WSAG. WSLA precisely describes how QoS parameters are computed or aggregated from resource metrics, and so WIoT-SLA combines WSAG and WSLA to take advantage of their complementary features in describing QoS and QoS guarantees. This decision supports Feature 1 and 3.

Design Decision 2. Simplified negotiation offer schema based on WSAN
As a negotiation layer on top of WSAG, WSAN [Waeldrich et al., 2011] defines a schema for negotiation offers that conforms to the WSAG SLA schema, which also contains service information that is unrelated to negotiation issues. To reduce network overhead, WIoT-SLA defines a lightweight negotiation offer schema which only describes the negotiable terms and offer context. Each negotiation offer must have a template reference in the offer context for validation. After negotiation, an SLA is created by following constraints specified in both the SLA template and the acceptable negotiation offer. This decision supports Feature 1 and 4.

3.3.2 Distributed SLA Negotiation Model
As stated in Chapter 2.3, a negotiation framework that uses a centralized infrastructure to manage negotiation tasks has limitations such as a high risk of a single point of failure, low scalability, and large latency, while a purely decentralized negotiation framework has the potential to be inefficient. Considering the scale and limited bandwidth of IoT environments, a hybrid architecture that combines centralized and decentralized solutions can help to balance the trade-off between efficiency and network consumption. iNegotiate is a distributed negotiation framework that defines and implements a hybrid centralized-decentralized architecture using a logical Hierarchical Negotiation Overlay Network (HNON). The HNON allows gateways to coordinate negotiation tasks without a centralized infrastructure.

Design Decision 3. Hybrid Distributed Architecture
To balance the trade-off between efficiency and network consumption, a hybrid centralized-decentralized architecture was designed for iNegotiate to coordinate negotiation tasks in a distributed manner. The architecture is implemented by the HNON, which is created
by gateways autonomously assigning different roles to themselves during the system initialization stage. The HNON has a three-layered structure consisting of a follower layer, controller layer and coordinator layer. The follower layer takes charge of candidates’ prioritizing and bilateral negotiations. The controller layer clusters registered templates in the local area and identifies candidates by performing business task-based searching (i.e., matchmaking based on the semantic relations of service parameters [Chen and Clarke, 2014]). The coordinator layer propagates messages and allocates negotiation tasks based on location information. A detailed description of HNON is presented in Section 3.4.2. This decision supports Feature 2.

**Design Decision 4. Location-based templates organization**

SLA templates are distributed in the gateway network depending on services’ spatial features. This is under the assumption that service providers are more likely to appear or move around in the areas that are close to the advertised service location. The gateways deployed at places close to the service location have a bigger chance of directly connecting to the service provider, which increases the chance of a timely bilateral negotiation without introducing extra network traffic and reduce timeout failures. In other words, the gateway which performs the bilateral negotiation is the one that stores the template that potentially accords with a consumer’s request. The location-based templates organization facilitates negotiation task allocation and avoids gateways negotiating with providers that cannot satisfy a consumer’s spatial requirement. This decision supports Feature 2.

**Design Decision 5. Layer-based communication**

Negotiation gateways have different physical network connections (i.e., Internet, Local Area Network (LAN), Wireless Local Area Network (WLAN)) and communication interfaces (i.e., WiFi or Ethernet). The differences in communication ranges and network capacity can cause communication problems during the negotiation process. To increase message propagation efficiency, iNegotiate defines a layer-based communication mechanism where requests (i.e., template registration requests and negotiation requests) are first forwarded to the coordinator layer to search for appropriate controllers, then forwarded
to the follower layer to process the request. This layer-based communication mechanism
avoids message flooding and supports the optimal propagation of messages since each
gateway only has partial knowledge about other nearby gateways. The detail of message
exchange rules during the SLA negotiation process is described in Section 3.4.2. This
decision supports Feature 2.

3.3.3 Trust-based Negotiation Candidates Selection

Existing service provider selection approaches use service matchmaking or reputation systems
to identify optimal service providers, which do not measure the similarity between requests
and services when service properties are negotiable and the corresponding negotiation con-
straints are unknown. iNegotiate addresses these issues by making the following decision:

**Design Decision 6. Experience-based candidates assessment**

A trust model based on historical information relating to a provider’s previous negotiations
and SLA compliance was designed to select a service provider that has a higher chance
of making an agreement as well as providing the requested service with negotiated quality
level. However, IoT service providers may dynamically adjust their negotiation constraints
based on different factors (e.g., device status, time, current workload, etc.), which causes
inconsistency in the negotiation result. To predict the possibility of successful negotiation
from imprecise historical data, Rough set theory [Pawlak and Skowron, 2007] is used to
analyze the importance of different negotiation terms in making final decisions. To keep
track of the SLA fulfillment (i.e., reputation) of a particular service provider, Bayesian
inference based on historical monitoring data is employed to estimate the probability that
the provider keeps its promises. Since unforeseen events in the environment may damage
the devices related to an IoT service, which further influences service performance signif-
ically, a fading mechanism was designed to allocate more weight to recent observations.
This decision supports Feature 3.
3.3.4 Deadline-aware negotiation strategy for negotiations with incomplete information

Section 2.3.2 discusses negotiation strategies that use utility functions, decision functions and machine learning techniques to make concessions during bilateral negotiations. These approaches are not suitable for IoT service negotiations because they do not support IoT service properties, require massive interactions and have a high requirement for hardware computation capability. iNegotiate addresses these issues by making the following decisions:

Design Decision 7. WSAN-based decision-making model

Since negotiating participants are potentially mobile, and there is limited network bandwidth in the IoT environment, SLA negotiation requires a deadline-aware strategy to reduce network consumption and the number of timeout failures. iNegotiate uses a deadline-aware decision-making model to take action when receiving a new offer. This model is designed based on the offer state transition model specified in WSAN. The state of each offer indicates a subsequent interaction mode. A negotiating gateway needs to make a decision that conforms to the state of the received offer. Negotiation stops when an offer is in an acceptable state or the only received offer is in the rejected state. This decision supports Feature 4.

Design Decision 8. Adaptable negotiation tactic

Negotiation in a business environment is likely to have incomplete information, which makes it hard to find a win-win solution. To balance the trade-off between success rate and negotiated utility, iNegotiate uses two adaptable negotiation tactics to generate new proposals in different situations. The utility function-based tactic dynamically adjusts concession in each round based on context information and the consumer’s negotiation preference, while the metaheuristic tactic seeks the optimal solution based on the consumer’s negotiation constraints and the counteroffers proposed by the service provider. These adaptable negotiation tactics create irregular concessions in each round, which makes them hard to predict by the negotiation opponent. This decision supports Feature 2 and 4.
3.4 *iNegotiate*

*iNegotiate* is a middleware solution for distributed SLA negotiations in the IoT environment, designed to be deployed on a set of edge devices used as negotiation gateways. Figure 3.3 shows the core components of the proposed system and the sequential activities of an SLA negotiation process where service providers publish their SLA templates (marked in green arrows), requests are forwarded to different gateways based on location information (marked in blue arrows), gateways select candidate service providers based on consumers’ requests, and start the bilateral negotiations with trusted candidates (marked in yellow arrows).

The *iNegotiate* is composed of five main components as follows:

- **SLA manager**: stores the SLAs/templates in local repositories and initializes SLA management tasks (i.e., template verification and SLA monitoring) based on related concepts defined in WIoT-SLA ontology. A QoS monitoring engine is assumed to be deployed in the SLA manager, which instantiates monitoring instances when an SLA comes into effect. The monitoring instance collects run-time QoS data and reports the SLA violation rate to the trust evaluator. The details of SLA management and WIoT-SLA ontology are described in Section 3.4.1.

- **Interaction handler**: This component is responsible for exchanging messages between service consumers/providers and forwarding messages to other gateways. The interaction handler receives requests from providers/consumers and forwards them to the gateway that is closest to the advertised/requested service location for further processing. The details of negotiation task allocation are described in Section 3.4.2.

- **Template matchmaker**: This component detects the candidate templates that have the potential to satisfy a consumer’s request by performing semantic template matchmaking, which calculates the WuPalmer relatedness [Jurafsky and Martin, 2014] between the requested service terms and the terms specified in SLA templates using an auxiliary source *WordNet* [Pedersen et al., 2004]. The semantic similarity checking is described in Section 3.4.1.
• **Trust evaluator:** This component is used to prioritize negotiation candidates if multiple candidate templates are detected by the template matchmaker. It calculates the trust credit that reflects the applicability of each candidate based on its historical negotiation records and SLA compliance records. Details of the trust-based assessment is described in Section 3.4.3.

• **Negotiator:** This component performs the sequential bilateral negotiations with Top-K candidates and returns the optimal solution to the service consumer through the interaction handler. Also, the negotiation results are passed to trust evaluator to update the negotiation records in preparation for new requests. Details of the negotiation strategy adopted by negotiators are described in Section 3.4.4.

### 3.4.1 SLA Ontology Model for IoT Services

To automate the SLA management in capacity-aware SOA, common global knowledge of SLAs is needed to make SLAs reciprocally understandable. This uniform SLA ontology not only allows providers to express their offerings in a standardized way but also provides necessary information for SLA management tasks such as negotiation and monitoring.
Automated SLA Management

Figure 3.4 presents the relationship between the SLA ontology model and SLA management activities including SLA negotiation, SLA creation, SLA monitoring, and service billing. The SLA ontology generalizes the semantics of SLA specification and negotiation specification that formalizes SLAs/templates and negotiation offers respectively. During the negotiation, negotiation participants express their expected values of negotiable terms in negotiation offers, which are created by following the negotiation information, creation constraints and validation rules specified in the SLA template. These offers are adjusted and exchanged between trading parties until an agreement is reached. Based on the mutually acceptable offer and the corresponding template, a final SLA is created. During the service provisioning time, a monitoring instance is instantiated according to QoS assessment information (e.g. negotiated guarantees, measurement metrics, and assessment schedulers) specified in the SLA, which automatically collects run-time QoS data and verifies if the actual QoS conforms to the guaranteed quality level. Once the negotiated guarantee is violated, the pre-defined adaptation actions (e.g., SLA renegotiation) and automatic billing are triggered to protect consumers’ benefits.

The SLA specification defines the standard format of an agreement, which contains the
SLAs in the IoT domain require the non-ambiguous description of IoT services, but in the meanwhile, should remain as simple as possible [Papadopoulos et al., 2017]. As described in Chapter 2.2.3, existing IoT ontologies specified three well-accepted concepts, which are the entity, resource, and service [Bermudez-Edo et al., 2016]. The semantic models of these concepts are associated with each other by attributes such as location, domain information, physical concept, and observations. In service-oriented IoT environments, consumers may focus more on the service/application layer rather than the sensor layer described by a physical device ontology. By merging the important attributes of the entity/resource model to the service model, a uniform SLA ontology for IoT services can be created. Figure 3.5 shows the upper ontology that describes the relations between different core concepts of capacity-aware SOA-based IoT.

**SLA Contextual Ontology**

To enable a flexible and reliable service provisioning in a capacity-aware SOA-based IoT environment, this thesis proposes WIoT-SLA as an SLA ontology to describe IoT services along
Figure 3.6: WIoT-SLA structure

with the negotiation and monitoring information. In general, WIoT-SLA specifies the formal syntax of SLA, SLA templates and negotiation offers. Considering the extendability and unambiguity of SLAs and SLA templates, WIoT-SLA combines two of the most commonly-used extendable web service SLA specifications: WSAG [Andrieux et al., 2007] and WSLA [Ludwig et al., 2003], to take advantage of their complementary features. The WSAG schema is used to structure SLAs and formalize service descriptions, while WSLA is used to define measurement metrics of QoS parameters. Considering the readability, device’s capability and message payload, WIoT-SLA formulates the SLA, SLA template and negotiation offer using JSON format. Figure 3.6 and Figure 3.9 show the structures of SLA and SLA template respectively. The blocks marked in green are the concepts derived from WSAG, the blocks marked in red are the concepts derived from WSLA, while the blocks marked in yellow are the extensions made by WIoT-SLA.

As Figure 3.6 shows, the structure of the WIoT-SLA consists of agreement context (i.e.,
party information, expiry date, agreement template identifier, and dependent SLAs\(^3\) and \textit{terms}. Services are described by \textit{terms}, which are classified into \textit{service description term} (SDT), \textit{service property} (SP), \textit{service reference} (SR), and \textit{guarantee term} (GT). SDT is the fundamental component of an SLA, which describes the functionality that will be delivered by the service. SP defines the measurable and exposed QoS properties associated with the service. SR (optional) lists the references point to the service (e.g., a WSDL document or a restful web service interface). GT defines the assurance of an SP variable in the form of SLO, which specifies a customized quality level that is guaranteed by the obligated party. A GT may have an optional \textit{importance} indicator that expresses the importance of meeting the SLO, which would be useful for making trade-offs during SLA negotiation. Each SLO is associated with a compensation definition specifying the penalty when the assurance is not satisfied. To guarantee a correct billing mechanism, compensation is associated with an assessment interval over which the penalty is assessed. The assessment can be issued regularly according to the monitoring scheduler or triggered by a pre-defined event.

Figure 3.7 shows the ontology of service description terms proposed by WIoT-SLA, which consists of service type, time constraints, service coverage, and price. The blocks marked in purple are the concepts derived from IoT-A ontology [De et al., 2012]. \textbf{Time constraints} specify the actual service provisioning time. \textbf{Service coverage} specifies the spatial features of the service (e.g., the observation area of sensors). In WSN, modeling service coverage is complex because devices’ energy consumption, network topology, and deployment of sensor nodes are unpredictable [Gupta et al., 2016]. To simplify the problem, a regional coverage that models the service coverage as a circular region is pre-defined. The centre location of the circular region can be specified using the geographic coordinate. \textbf{Service type} generalizes a service’s functionality with domain information, service parameters (i.e., input and output), resources and their configurable features. In IoT, the service type can be clustered as sensing services (e.g., temperature sensing), sensing and actuation services (e.g., trigger the alarm when detected hazard gas concentration greater than a threshold), edge services (e.g., a data dispatch service that collects data from sensors and publish verified data to subscribed

\(^3\)The dependent SLAs (optional) specify the references of other SLAs that the current service depends on.
services) and cloud services (e.g., data storage/processing service that analyzing historical data and predict abnormalities) based on their accessed resources. The difference between a sensing service and a sensing/actuation service is that the latter has an extra “effect” attribute specifying an event or an action when the pre-defined condition is met. For example, for temperature monitoring, a sensing service only specifies “temperature” as its quantity kind, while a sensing and actuation service also specifies the effect “trigger the alarm” when the condition “the ambient temperature rises above the threshold” is met. To achieve semantic interoperability, WIoT-SLA suggests to use an international standard (e.g., ISO-80000 [ISO Central Secretary, 2009]) or existing ontology (e.g., Qudt ontology [Chalk et al., 2017]) to define service parameter types and units. The resources exposed by a service are associated with one or more configuration items, which specifies the service’s functional features, such as sampling parameters for sensing services, data reporting rate for edge services, or memory capacity for cloud services. The configuration item is defined with a name, a value, and data type (e.g., boolean, numeric and string) of the value. Figure 3.8 presents a set of possible
configuration items for sensing services, such as sensitivity, security, sample interval, data aggregation, and data transmission [Project, 2013].

In WIoT-SLA, SP contains a set of variables specifying a service’s dynamic QoS features (i.e., values are affected by devices’ status and run-time environment) that can be monitored by a measurement party. Each SP variable is described by a name and its customized measurement metrics. As described in Chapter 2.2.3, the QoS model in the IoT context needs to consider the complexity introduced by the layered architecture of IoT applications [Duan et al., 2011], which is composed of the perception layer, network layer, service layer, and application layer from bottom to top. Quality parameters of the perception layer include data correctness, data completeness, transmission speed, energy consumption, price, etc [Tönjes et al., 2014, Kolozali et al., 2018]. Quality parameters of the network layer focus on the performance of data transmissions, such as network delay, bandwidth, packet loss, and network jitter [Chen and Varshney, 2004]. Quality parameters of a service layer, especially for cloud services, contain usability, availability, reliability, responsiveness, security, and elastic-
ity [Zheng, 2014]. The end-to-end QoS of the application layer depends on the aggregated or nested QoS metrics across the lower layers. Achieving SLOs at the application layer requires the satisfaction of the SLOs of lower layers, the guarantee states\(^4\) of SLOs in lower layers can be used as a precondition under which the application SLOs take effect. The precondition can be specified in the Qualifying Condition associated with each SLO. To precisely define the semantics of QoS parameters, the concepts of measurement directives and composite metrics derived from WSLA [Ludwig et al., 2003] is merged into WIoT-SLA. The measurement directive is the QoS metric that can be directly retrieved from managed resources (e.g., a request URI exposed by the QoS monitor). The composite metric is an aggregated QoS metric that is built based on measurement directives or other composite metrics according to a pre-defined function. For example, availability is a composite metric aggregated from two measurement directives (i.e., service uptime and service execution time) using a function (i.e., the ratio of the service uptime to the service execution time). WIoT-SLA uses KPI name and custom guarantee level to define the guarantee of each service property (i.e., SLO). The KPI name is the name of the service property that the guarantee holds (e.g., the service property “responsiveness” defined as the probability of a service responding within 3 seconds) and the custom guarantee level specifies the guaranteed thresholds (e.g., the responsiveness should be greater than 90%).

\(^4\)The guarantee state model specified in WSAG represents a fulfillment state for each GT of an SLA.
As Figure 3.9 shows, SLA templates share the same structure as SLAs except for some additional segments, which are temporality (mandatory), negotiation information (optional), and creation constraint (optional). The temporality segment is designed for template management, while the remaining segments are designed for dynamic SLA negotiation and creation. An SLA template’s temporality is defined by creation timestamp and expiry date, which allows negotiation gateways to update the registered templates to the latest version and remove the expired ones. The negotiation information specifies the negotiation interface (i.e., a restful negotiation service EPR or an instant message address) and the negotiation protocol (e.g., CNP or WSAN) if the service is negotiable. If a template does not specify this segment, the service is non-negotiable, and users have to accept all the default values specified in the template. For negotiable templates, the creation constraint is used to specify the terms that must be presented in the initial negotiation offer and the final SLA with the values satisfying the constraints. For instance, a service provider may list all the available time slots in constraints for gateways to select the most preferred one. WIoT-SLA defines three types of creation constraints, which are Range (i.e., specify the minimum and maximum value), Enumeration (i.e., lists all the possible values) and Function (i.e., the values are defined as a function). The creation constraint is composed of an name attribute and a targetLocation attribute. The name attribute uniquely identifies different constraints (i.e., the duplicated name is not allowed). The targetLocation specifies where to put the constraint using the JSONPath\textsuperscript{5} querying language. The terms that are not specified in creation constraints are regarded as non-negotiable terms, which must hold the same value presented in the template. Figure 3.10 shows a set of example creation constraints that correspond to pre-defined SDTs and GTs. As stated in Chapter 2.3.2, a service provider may not disclose all the negotiation constraints in SLA templates for business competition reasons. In such a case, for example, the price constraint may be unspecified.

WSAN formalizes negotiation information as negotiation offers [Waeldrich et al., 2011], which are generated based on an SLA template. The offer’s structure specified in WSAN conforms to the SLA schema specified in WSAG, which is too heavyweight for SLA negotiation.

\textsuperscript{5}https://goessner.net/articles/JsonPath/index.html#e2 - Accessed 15 Jan 2019
in IoT environments. To reduce message payload during the negotiation process, WIoT-SLA regulates that the SLA template must be referred to in each negotiation offer, and only the negotiable terms specified in creation constraints can be presented in negotiation offers. Other terms are omitted but regarded as holding the same values presented in the template. Any inconsistency will cause an offer validation failure. Figure 3.11 shows the structure of WIoT-SLA negotiation offers. Each offer is composed of the offer context (mandatory), negotiable terms (mandatory), and negotiation constraints (optional). The offer context contains the reference of the previous offer (i.e., counterOfferTo), the offer creator, the offer state derived from the state transition model of WSAN, and the referred template identifier. The negotiation constraints contain the possible constraints on negotiable terms when creating a valid counteroffer, which has a similar format to a creation constraint, except that an additional constraint type FixedValues is designed to indicate the value can not be changed in subsequent offers or the final SLA. For the rejected offers, to avoid futile interactions, the offer context may be extended with domain-specific information to indicate why the offer is rejected. The set of predefined rejected reasons are: \{UnsupportedTerm, SLOConflict, Timeout, InvalidOffer, UnderPayment\}.

**SLA Template Matchmaking**

The WIoT-SLA template matchmaking process identifies the SLA templates that are compatible with a request based on the similarity between requests and services. A request is defined as \( R = \langle I, O, T, L, F, Q, C \rangle \), consisting of inputs, outputs, time, location, functional
terms (e.g., sample interval), QoS terms, and negotiation constraints (e.g., the expected price range). To balance the match-making efficiency and response time, a multi-phase match-making mechanism gradually narrows down the searching space (Algorithm 1). First, a business task-based search is performed by a service discovery engine based on the semantic relations of service parameters [Chen and Clarke, 2014] (Line 2). Second, the discovered candidate templates are filtered based on time and spatial features (Line 3-8). Third, a deeper filtering mechanism is performed by iteratively checking the semantic similarities between requested features (i.e., functional/non-functional requirements) and service features (i.e., configuration items and service properties). The semantic similarity lower than the predefined threshold indicates an unmatched feature, and the templates that have unmatched features are filtered (Line 10-20). Fourth, the correspondence between the request and the remaining candidate templates is calculated to identify the optimal template that is the most similar to the request (Line 23-31). The correspondence is defined as the weighted sum of semantic similarities between the requested features and service features (Equation 3.1).

\[
\text{correspondence} = \sum_{t \in T} w_t \cdot \text{SemanticSim}(R_t, st_t) \tag{3.1}
\]

where \( R_t \) and \( st_t \) are the matched features in the request and candidate template, and \( w_t \) is the weight of term \( t \). The weight of each matched term is calculated based on the data type, default value and creation constraints presented in the template. If the term is negotiable but the constraint is not clearly specified, the weight \( w_t \) is set to 1. If the feature is negotiable
and the value is restricted in the constraints, the weight of a string type is measured by the minimum Levenshtein distance [Gusfield, 1997], and the weight of a numeric type is calculated by Equation 3.2 for cost-type features (e.g., price or sample interval from a consumer’s perspective), and by Equation 3.3 for benefit-type features (e.g., availability or reliability from a consumer’s perspective).

\[
w = \begin{cases} 
1, & \text{if } S_{\text{max}} \leq R_{\text{min}} \\
\frac{|R \cap S|}{|R|}, & \text{otherwise.} 
\end{cases}
\] (3.2)

\[
w = \begin{cases} 
1, & \text{if } S_{\text{min}} \geq R_{\text{max}} \\
\frac{|R \cap S|}{|R|}, & \text{otherwise.} 
\end{cases}
\] (3.3)

where \(S_{\text{max}}, S_{\text{min}}, R_{\text{max}}, R_{\text{max}}\) are the maximum and minimum values of the offered feature and the requested feature respectively. \(R \cap S\) is the intersection between the request values and the offered values.

### 3.4.2 Distributed SLA Negotiation Model

Figure 3.12 shows the negotiation model of *iNegotiate* that has three stages: pre-negotiation, negotiation, and post-negotiation. The different phases in each stage are controlled by a negotiation protocol, which specifies a set of messages, message interaction rules, and processing operations when receiving a particular type of messages. To support the dynamic negotiation in a distributed environment, there are seven types of messages, which are generalized as follows:

**Definition 1. Ping message**, which is defined as: \(\text{Ping} = S_{\text{id}}, R_{\text{id}}, \text{"hello"}\), consisting of the sender identifier \(S_{\text{id}}\), receiver identifier \(R_{\text{id}}\) and a “hello” string.

**Definition 2. Configuration message**, which is defined as: \(\text{Cfmsg} = O_{p}, S_{\text{id}}, R_{\text{id}}, m, \text{ttl, Route}\), consisting of an operation code \(O_{p}\), sender identifier \(S_{\text{id}}\), receiver identifier \(R_{\text{id}}\), message content \(m\), the maximum number of hops \(\text{ttl}\), and a routing table \(\text{Route}\).
Algorithm 1  WiIoT-SLA Template Matchmaking

**Input:** Request \( R = (I, O, T, L, F, Q, C) \), candidate templates \( ST \)

**Output:** candidate templates \( ST \) ranked by template’s correspondence \( st_\sigma \)

1: while receive a request \( R \) do
2: /* \( ST \leftarrow \) serviceDiscovery(\( R_I, R_O \)) */
3: \( d \leftarrow R_C.getLocationConstraint \)
4: for all \( st \in ST \) do
5: \( st_L \leftarrow ST.getServiceLocations \)
6: \( st_T \leftarrow ST.getServiceTemporalities \)
7: if minDistance(\( R_L, st_L \)) > \( d \) or \( R_T \cap st_T = \emptyset \) then
8: \( ST.remove(st) \)
9: else
10: \( st_F \leftarrow st.getConfigurationItems \)
11: for all \( st_f \in st_F \) do
12: \( st_{fss} \leftarrow getMaxSemanticSimilarity(R_f.name[], st_f.name) \)
13: if \( st_{fss} < \) threshold then
14: \( ST.remove(st); \) break
15: end if
16: end for
17: \( st_Q \leftarrow st.getServiceProperties \)
18: /*Semantic matching for QoS features similar to Line 10-16*/
19: end if
20: end for
21: for all \( st \in ST \) do
22: \( st_{cc} \leftarrow st.getCreationConstraints \)
23: for all \( r_f \in R_F \) do
24: \( s_f \leftarrow getSemanticMatchedPair(r_f) \)
25: \( w_f \leftarrow calculateWeight(st_f.value, r_f.value, R_C, st_{cc}) \)
26: \( st_\sigma \leftarrow st_\sigma + st_{fss} \times w_f \)
27: end for
28: /*Calculate correspondence for QoS features similar to Line 23-27*/
29: end for
30: \( ST \leftarrow ST.sort(st_\sigma) \)
31: end while

**Definition 3. Template registration message**, which is defined as: \( Tr_{msg}=<O_p, S_id, R_id, S_t, Route, \_m> \), consisting of an operation code \( O_p \), sender identifier \( S_id \), receiver identifier \( R_id \), a template \( S_t \), a routing table \( Route \) and a customized message content \( m \) (optional) that facilitates the template distribution process.

**Definition 4. Negotiation request message**, which is defined as: \( N_r_{msg}=<S_id, R_id, ...
\(R_{eq}, O_p, SC_{id}, Route, \ast m\rangle\), consisting of the sender identifier \(S_{id}\), receiver identifier \(R_{id}\), negotiation request \(R_{eq}\), operation code \(O_p\), consumer identifier \(SC_{id}\), a routing table \(Route\) and a customized message content \(m\) (optional) that facilitates the request forwarding process.

**Definition 5. Negotiation customize message**, which is defined as: \(Nc_{msg}=\langle Ni_{id}, Nr_{id}, m, S_t, O_p, \rangle\), consisting of the negotiation initiator identifier \(Ni_{id}\), negotiation responder identifier \(Nr_{id}\), message content \(m\) that specifies the negotiation context (e.g., negotiation protocol, SLA schema, deadline and communication interface), the referred template \(S_t\), and operation code \(O_p\).

**Definition 6. Negotiate message**, which is defined as: \(Ng_{msg}=\langle S_{id}, R_{id}, O, O_p, N_{id}, \ast Route\rangle\), consisting of the sender identifier \(S_{id}\), receiver identifier \(R_{id}\), negotiation offers \(O\), operation code \(O_p\), a negotiation instance identifier \(N_{id}\) and a routing table \(Route\) (optional) that facilitates the negotiation result aggregation process.

**Definition 7. Signing request message**, which is defined as: \(Sr_{msg}=\langle G_{id}, SC_{id}, O_a, a_{pl}\rangle\), consisting of the gateway identifier \(G_{id}\), the consumer identifier \(SC_{id}\), an acceptable offer \(O_a\), and the message content \(m\) containing an approval indicator.

**Definition 8. Mobile entity locating message**, which is defined as: \(Ml_{msg}=\langle S_{id}, R_{id}, E_{id}, m, O_p, Route\rangle\), consisting of the sender identifier \(S_{id}\), the receiver identifier \(R_{id}\), the entity identifier \(E_{id}\), message content \(m\) (i.e., handover requests and messages), a operation code \(O_p\), and a routing table \(Route\).

The pre-negotiation stage comprises a system initialization phase and a template distribution phase. In the system initialization phase, gateways exchange ping and configuration messages to create a logic hierarchical negotiation overlay network (HNON), which manages SLA templates and controls the message flow during the negotiation process (Phase 1.1). During the template distribution phase, service providers publish their templates by sending template registration messages. HNON propagates the registration message to a gateway.
close to the advertised service location to store the template (Phase 1.2). This location-based template distribution aims to increase the chance of a successful communication with candidate service providers.

The **negotiation stage** begins when a consumer submits the request through a negotiation request message. The negotiation task is allocated to the service providers that have the potential to satisfy the requirements and the gateways that can communicate with these providers. First, the message is forwarded to the gateways close to the requested location over HNON. The gateways compare the request with locally stored templates to search for candidate templates and prioritize the discovered candidates based on the providers’ historical information (Phase 2.1). Then, gateways select the high priority candidates and send negotiation customization messages as the handshake to test network connections (Phase 2.2). If the provider is mobile and the changed network connection has not been identified by gateways yet, an entity locating message is generated and propagated over HNON to locate the provider (Phase 2.3). If the handshake is successful, a negotiation instances is initialized before starting bilateral negotiations (Phase 2.4).

In the **post-negotiation stage**, the optimal solution is selected from the acceptable negotiation offers achieved by different gateways and sent back to the consumer through a
signing request message (Phase 3.1). If the consumer is mobile and the network connection has changed, gateways forward the entity locating message over HNON to locate the consumer (Phase 3.2). Once the solution is received and approved by the consumer (i.e., the message is digitally signed), an SLA can be created based on the template referred by the optimal solution (Phase 3.3).

System Initialization Phase

To enhance the negotiation efficiency without introducing an excessive number of messages, iNegotiate uses a hybrid centralized-decentralized architecture to cluster registered SLA templates and allocate negotiation tasks, which is implemented by a hierarchical negotiation overlay network (HNON). HNON is a logistic overlay network built upon the actual network topology. Figure 3.13 shows the three-layered HNON where each gateway is assigned to at least one of the following roles: follower, controller or coordinator. Each follower has a controller and each controller is associated with a coordinator. All gateways have the follower role by default, while controllers and coordinators are autonomously assigned by gateways according to their properties in system initialization phase.

The bottom layer of HNON consists of followers that work as brokers. The middle layer consists of controllers that divide the environment into a set of sub-areas. Each
controller can be regarded as a small data centre in a sub-area, which replicates the templates registered to its followers. The sub-area is referred to as the controller’s range, which is roughly estimated by the maximum distance between the controller and its followers:

$$\text{Range} = \max_{f \in F} \{\text{distance}(C_{loc}, f_{loc})\}$$  \hspace{1cm} (3.4)

where $F$ is the collection of followers of controller $C$.

The top layer consists of coordinators that are directly connected with each other through the internet. In HNON, the follower layer guarantees a timely bilateral negotiation with service providers, the controller layer discovers candidate services and allocates negotiation tasks to appropriate followers. The coordinator layer propagates messages over different sub-areas and allocates negotiation tasks to appropriate controllers. Figure 3.14 shows the cross-layer message flow in the HNON. Followers forward messages to controllers to submit negotiation tasks or replicate local registered templates. Controllers forward messages to coordinators to enable efficient message transmissions over different sub-areas.

To get as much local template information as possible, in each sub-area, the gateway that can access the highest number of other gateways through WiFi is assigned as the local controller. To maximize communication efficiency, an Internet-connected gateway that can access the highest number of controllers through WiFi is assigned as a coordinator. Gateways
Algorithm 2 Autonomous Role Assignment (a) - Message Sender

1: Start timer T1; Broadcast Ping;
2: while T1 is not due do
3:   if receives a reply then
4:     Record neighbour information;
5:   end if
6: end while
7: Cache itself as controller;
8: Create $Cf_{msg}$ ($Cf_{msg}.m \leftarrow$ cached controller info, $Cf_{msg}.O_p \leftarrow$ CIM);
9: Set Timer T2; Send $Cf_{msg}$ to neighbours;
10: if T2 is due then
11:   if cached controller is itself then
12:     Mark itself as controller;
13:   else
14:     Mark itself as follower and save cached controller’s information;
15:     Create $Cf_{msg}$ ($Cf_{msg}.m \leftarrow$ self info, $Cf_{msg}.O_p \leftarrow$ CVM);
16:     Send $Cf_{msg}$ to the controller;
17:   end if
18:   /*wait one minute for message synchronizing*/;
19: if current gateway is marked as controller then
20:   Create $Cf_{msg}$ ($Cf_{msg}.m \leftarrow$ self info, $Cf_{msg}.O_p \leftarrow$ RIM);
21:   Set Timer T3; Send $Cf_{msg}$ to neighbours;
22: end if
23: if T3 is due then
24:   Create $Cf_{msg}$ ($Cf_{msg}.m \leftarrow$ self identifier, $Cf_{msg}.O_p \leftarrow$ RVM);
25:   Save cached coordinator information and send $Cf_{msg}$ to the coordinator;
26: end if
27: if current gateway is marked as a new coordinator then
28:   Create $Cf_{msg}$ ($Cf_{msg}.m \leftarrow$ self info, $Cf_{msg}.O_p \leftarrow$ FRM);
29:   Broadcast $Cf_{msg}$;
30:   if receives a reply: Add the responder to coordinator list;
31: end if
32: end if

communicate through the WiFi interface at this stage because the limited connection range of the WiFi network can help to cluster gateways according to their locations.

Algorithms 2, 3, and 4 show the interaction rules of configuration messages when assigning different roles to gateways from the sender and receiver’s perspectives. Gateways first broadcast ping messages through WiFi to identify their neighbouring gateways (Algorithm 2, Line 1-5). Based on the number of replies, gateways acquire their connection information
Algorithm 3 Autonomous Role Assignment (b.1) - Message Receiver

1: /* Listening configuration messages */
2: if \( Cf_{msg}.Op = \text{CIM} \) then
3:  resvCon ← \( Cf_{msg}.getMessageContent().getControllerConnections() \);
4:  resvRoute ← \( Cf_{msg}.getRoute() \);
5:  if resvCon > cached controller’s connections then
6:     Update cache with received controller’s information;
7:  Set state into active;
8:  else if \( Cf_{msg} \) specifies a shorter route for a same controller then
9:     Update cache with resvRoute;
10:    Set state into active;
11:  else
12:     Set state into inactive;
13: end if
14: if state is active and TTL is not reached then
15:     Add self identifier to resvRoute;
16:     Create \( Cf_{msg} (Cf_{msg}.m \leftarrow \text{cached controller info}, Cf_{msg}.Op \leftarrow \text{CIM}, resvRoute) \);
17:     Send \( Cf_{msg} \) to other neighbours;
18:     Set state into inactive;
19: end if
20: else if \( Cf_{msg}.Op = \text{CVM} \) then
21:     Mark itself as a controller, update range;
22:     Mark sender as a follower, save follower’s identifier, location and route;
23: end if

and exchange this information with neighbours using configuration messages. Each gateway initially caches itself as the controller and sends the controller’s information (i.e., controller identifier and the number of connections) to its neighbours through a configuration message (\( O_p \) is set to CIM, Line 1-6 in Algorithm 2). The gateways which receive the message check if it is necessary to update the cached controller’s information. If the controller specified in the received message has more connections than the cached one, or a shorter route has been detected for the same controller, the gateway caches the new controller’s information and forwards the message to other neighbours (Algorithm 3, Line 2-18). To avoid propagating useless information, gateways only send their cached controller’s information when the information is updated (i.e, in the active state) and the maximum hop (i.e., TTL) has not been reached. The message’s TTL controls the message propagation range, which also affects the range of each sub-area. Since there is no centralized server that controls the information
Algorithm 4 Autonomous Role Assignment (b.2) - Message Receiver

1: /* Listening configuration messages */
2: if $C_f_{msg}.O_p = \text{RIM}$ then
3:   if ttl is not reached then
4:     add the self identifier to message’s routing table;
5:     forward the message to other neighbours;
6:   end if
7:   if current gateway has Internet access then
8:     temporarily collect message sender’s information
9:   /*wait one minute for message synchronizing*/;
10:  route ← $C_f_{msg}.getRoute();
11:  Create $C_f_{msg}$ ($C_f_{msg}.m ←$ self info, $C_f_{msg}.O_p ← \text{RIMB}$, route)
12:  Send $C_f_{msg}$ to message sender
13: end if
14: else if $C_f_{msg}.O_p = \text{RIMB}$ then
15:   if current gateway is the destination then
16:     $C_f_{msg}.get MES_{Content().getCoordinatorConnections();}
17:     recsRoute ← $C_f_{msg}.getRoute();
18:   if Cached coordinator is $\emptyset$ then
19:     Create cache with received coordinator’s information;
20:   else if recsCon > cached coordinator’s connections then
21:     Update cache with received coordinator’s information;
22:   else if $C_f_{msg}$ specifies a shorter route for a same coordinator then
23:     Update cache with recsRoute;
24:   end if
25: else
26:   forward the message according to routing information
27: end if
28: else if $C_f_{msg}.O_p = \text{RVM}$ then
29:   Mark itself as a coordinator, add sender to controller list;
30: else if $C_f_{msg}.O_p = \text{FRM}$ then
31:   if current gateway is a coordinator then
32:     Save sender identifier to coordinator list;
33:     Reply with self identifier;
34:   end if
35: end if

exchange process, a pre-defined timer T2 is set for gateways collecting the controller’s information. When the timer is due, a verification is sent to the cached controller using a $C_f_{msg}$ whose $O_p$ is set to $\text{CVM}$ (Algorithm 2, Line 13-16). The controller which receives the verification saves the follower’s information (i.e., identifier and routing information) and updates its
range (Algorithm 3, Line 21-22). After a short message synchronizing time, controllers start to query coordinator information by multicasting $Cf_{msg}$ ($O_p$ is set to $RIM$) to neighbours (Algorithm 2, Line 21-22). The receivers forward the message to other neighbours to search for an internet-connected gateway within the message propagation range. If a receiver has an Internet connection, it temporarily records the sender’s information (i.e., identifier and routing information) and waits for a short time to possibly collect more information about controllers in neighbouring sub-area. Then it sends a response to the message sender with the collected information using a $Cf_{msg}$ whose $O_p$ is set to $RIMB$ (Algorithm 4, Line 2-13). The gateway that receives the $RIMB$ $Cf_{msg}$ caches the sender’s information and updates the cached information when a better coordinator is detected (Algorithm 4, Line 14-27). When the pre-defined message collection time $T3$ is due, a verification is sent to the cached coordinator ($O_p$ is set to $RVM$). The receiver marks itself as a coordinator and saves the controller’s identifier, location, and range (Algorithm 2, Line 23-26; Algorithm 4, Line 28-29). A new coordinator collects other coordinators’ information by broadcasting a $Cf_{msg}$($O_p$ is set to $FRM$) through the Internet (Algorithm 2, Line 27-31; Algorithm 4, Line 30-34).

Figure 3.15 shows an example of a sub-area after performing the automatic role assignment algorithm. Gateway $c$, with the maximum number of connections, is assigned as the controller. The internet-connected gateway $e$ that has access to $c$ and other controllers in neighbouring areas is assigned as the coordinator, The gateways marked in blue remain as followers. The range of the sub-area marked in yellow is estimated by the maximum distance between the controller and its followers. By creating such an overlay network, each coordinator knows the location and range information of nearby controllers, and each controller knows the locations of the followers within its managed sub-area and the templates registered in these followers. Gateways can utilize their partial knowledge to facilitate information propagation. For example, in the template distribution process, coordinator $e$ only needs to ask its controllers about the minimum distance between the service and the followers within each sub-area, and exchange this information with other coordinators to identify the closest gateway to save the template. In the request forwarding process, coordinator $e$ decides whether or not to propagate a request to controller $c$ based on $c$’s location and range. Controller $c$
decides whether or not to forward the request to a follower by matching the request with local registered templates. This mechanism allows messages to be propagated and processed only in the gateways that are likely to solve the request rather than all of the gateways in the network. The following subsections will detail the behaviours and message interaction rules in the main phases.

**Location-based Template Distribution Phase**

Service providers advertise their services by sending SLA templates to a nearby gateway through a $Tr_{msg}$ ($Op$ is set to $ADV$). A template is forwarded over the HNON and stored in the gateway closest to the service location. The template distribution process can be generalized as follows: coordinators first detect the subareas that cover or are closest to the service location, then controllers of these subareas detect the follower that is closest to the service location. Figure 3.16 shows the interactions of followers, controllers, and coordinators during the template distribution process. The message is first forwarded to a coordinator. The coordinator computes the distance between the service location and its controllers’ locations and forwards the message to controllers whose range is within the service coverage ($Op$ is set to $CREG$). Controllers compute the distance between service location and their followers’ locations, cache the follower that has minimum distance and informs its coordinator of this
The coordinator caches the controller that reports the minimum distance and propagates the distance $d_f$ to other coordinators ($O_p$ is set to TRG). If the coordinator does not have any controllers whose range is within the service coverage, the $d_f$ is set as the minimum distance between the service location and its controllers’ locations. Other coordinators perform the same process and reply with the minimum distance if it is not greater than the $d_f$ specified in the received message. When the message synchronizing time is due, the originating coordinator forwards the template to the cached controller if there is no reply ($O_p$ is set to FREG), or forwards the template to the coordinator that replies with the minimum $d_f$ ($O_p$ is set to RREG). Then, the template is further forwarded to the cached follower so that it can be saved in the gateway that is closest to the service location ($O_p$ is set to REG). All the registered templates are also replicated in corresponding controllers.

If a service provider is mobile and specifies a flexible service location, the SLA template is stored in the gateway that receives the request. The mobile provider re-submits the request
when moving more than a pre-defined distance (e.g. about 200 meters if the provider connects to the gateway network by WiFi). All the registered templates are periodically checked by gateways to remove the ones that are out of date or the providers are unreachable. Also, a provider can change their offerings after registration by submitting a new template with the same identifier but different creation timestamp. The updated template is submitted to a nearby gateway through a $\text{Tr}_{\text{msg}}(O_p \text{ is set to } \text{UPD})$. Similarly, this message is forwarded to the coordinator layer. Each coordinator forwards the message to its controllers to detect if the originating template is stored in their local areas and update it if so. If the service location is changed in the updated template, the same template distribution process will be performed to search for the closest gateway, and the old template will be deleted from the original gateway.

**Negotiation Task Allocation Phase**

The SLA negotiation process is mainly composed of three activities: (a) request forwarding and provider selection (i.e., negotiation task allocation); (b) negotiation customization with candidate service providers; (c) bilateral negotiations conducted by different followers. The request forwarding can be generalized as follows: coordinators first detect the subareas that cover the service location, then controllers of these subareas search for candidate templates and forward the request to the followers that store these templates. Figure 3.17 shows the location-based request forwarding mechanism. A user submits a request through a $\text{Nr}_{\text{msg}}$ ($O_p \text{ is set to } \text{SUB}$). The message is forwarded to the coordinator\(^6\) and propagated in the coordinator layer ($O_p \text{ is set to } \text{TREQ}$). Coordinators forward the message to the controllers whose range covers the requested location ($O_p \text{ is set to } \text{CREQ}$). Controllers match the request with backup templates to search for the candidates and forward the message and discovered templates’ identifiers to the followers that store the candidate templates ($O_p \text{ is set to } \text{INS}$). Considering the possibility that a controller may backup a large number of SLA templates, performing the whole WIoT-SLA template matchmaking process described in Algorithm 1 may be time-consuming and inefficient. In $i\text{Negotiate}$, controllers only perform business\(^6\)This coordinator is referred to as the initiating coordinator.
Figure 3.17: Processing of negotiation request messages

task-based searching to discover the candidate templates based on the semantic relations of services’ input and output signatures [Chen and Clarke, 2014]. Followers further filter the discovered templates and keep the semantically matched candidates. If a follower has multiple candidates to negotiate with, it evaluates these candidates based on historical information and selects the optimal one (Section 3.4.3).

Negotiation Customization and Bilateral Negotiation Phase

Based on the negotiation information provided in the candidate template, the follower sends a $N_{cmsg}$ to the provider’s negotiation interface as a handshake. Once the provider replies OK, the follower initializes the negotiation instance, which generates an initial offer according to the constraints specified in the request and the template. The bilateral negotiation phase starts by the follower sending the initial offer to the service provider using a $N_{gmsg}$ ($O_p$ is set to $NEG$). Figure 3.18 shows the message interaction rules during this phase. The follower
negotiates with the service provider by exchanging offers. Each time a new offer is proposed by the service provider, the follower makes decisions (i.e., accept/reject the received offer or propose a new offer) according to a pre-defined negotiation strategy (Section 3.4.4). Once an offer is accepted, the negotiated solution is sent back to the initiating coordinator through the controller layer and coordinator layer. When the maximum processing time specified by the consumer is reached, the initiating coordinator selects the optimal solution from acceptable offers and returns it to the consumer for approval using a $Sr_{msg}$.

**Mobile Entity Locating**

During the negotiation customization or consumer approval phase, the WiFi-connected mobile entities (i.e., service providers or consumers) may move to another place and lose their original network connections. Gateways locate mobile entities by propagating a $Ml_{msg}$ over different sub-areas. The entity locating process can be either triggered by a follower to locate a mobile service provider, or triggered by a coordinator to locate a mobile consumer. Figure 3.19 shows
the mobile entity locating mechanism. Similarly, if the message is created by a follower, it is first forwarded to the controller (\(O_p\) is set to FTC), then forwarded to the coordinator (\(O_p\) is set to FTR) for further processing. The coordinator propagates the message to its controllers to detect if the entity is still in the managed sub-areas (\(O_p\) is set to CINQ). Each controller that receives the message multicasts it to followers (\(O_p\) is set to FINQ) to detect whether it is possible to connect the entity in the local sub-area. The follower that can connect to the entity extracts the \(S_{r_{msg}}\) from \(M_{l_{msg}}\) (or creates a new negotiation customize message \(N_{c_{msg}}\)), and sends it to the consumer (or mobile provider). If the entity can not be located in any sub-area managed by the coordinator, the \(M_{l_{msg}}\) is multicasted to other coordinators (\(O_p\) is set to RINQ) to start an exhaustive searching over the HNON. This entity locating mechanism guarantees that the searching process is firstly performed in nearby sub-areas, and then in the whole gateway network.

**iNegotiate: an example scenario**

Figure 3.20 shows the distributed negotiation process under HNON in a small scale: 10 gateways deployed in the environment are connected through WiFi (dashed line) or Ethernet (solid line). The gateways \(G_7\), \(G_2\) and \(G_3\) are the controllers of three sub-areas that marked
Figure 3.20: An example of distributed SLA negotiation using HNON

in yellow, blue and pink. The Internet-connected gateway $G_2$ is also the coordinator of the three controllers. A consumer submits a request to gateway $G_1$ through WiFi at time $t_1$. $G_2$ finds the requested service location within the range of controller $G_7$ and $G_3$ and forwards the message to them. $G_7$ and $G_3$ search candidates in their backup template repositories and discover the templates provided by CSP1 and CSP2 have the potential to satisfy the request. Then the request is forwarded to the followers that actually stores these templates ($G_4$ and $G_8$). The followers instantiate negotiation instances and start bilateral negotiation with corresponding providers, and return the negotiated solution to the controllers. Controllers $G_7$ and $G_3$ forward the message to coordinator $G_2$, which is also the initiating coordinator. $G_2$ selects the optimal solution and sends it to the consumer through the mobile entity locating process.
3.4.3 Trust-based Evaluation of Negotiation Candidates

In a large-scale IoT environment where massive numbers of service providers are likely to offer similar services, a provider selection mechanism is needed when a follower gateway has multiple negotiation candidates. *iNegotiate* uses a trust model that reflects a provider’s competence and integrity, to prioritize the candidates before negotiating with them. In this model, competence measures the ability of a service provider to offer a service with the requested quality levels within the constraints specified by a consumer, while integrity measures the dependability of a service provider as to whether or not previously offered services were compliant with the promised quality levels outlined in the SLA, and whether the currently running service instances are operating in a normal state. A service provider’s trust credit is defined as the weighted sum of competence and integrity:

$$\text{credit} = \lambda \ast \text{competence} + (1 - \lambda) \ast \text{integrity} \quad (3.5)$$

where $\lambda \in (0, 1)$ represents a consumer’s preference on these two aspects. The candidate service provider who gets a high trust credit can have a higher chance of taking part in negotiations.

Figure 3.21 shows the events and activities of the trust model with regard to SLA negotiation. The loop starts when a negotiation request and discovered candidate templates are received by a follower gateway. If multiple candidates are received, the follower gateway first removes the ones that do not cover all the requested service terms (e.g., a consumer requests a service with high responsiveness and availability, a template that only guarantees availability should be removed from the candidate list). Then the gateway calculates the trust credit of each candidate service provider by assessing its competence and integrity based on historical records. The trust credit determines the negotiation priority of each candidate. Within the pre-defined negotiation time, the gateway performs a multi-bilateral negotiation with Top-K candidates until an agreement is reached or the negotiation time is due. The negotiation result is recorded for future assessment. Once the acceptable offer is approved by the consumer, an SLA is created and the corresponding monitoring instance is instantiated.
Figure 3.21: Trust model with SLA negotiation and monitoring

according to the measurement information specified in the SLA, which collects the run-time QoS data and predicts service performance in the near future. When the SLA is due, the SLA violation rate is recorded for future assessment.

**Competence Assessment**

Negotiations in business environments are likely to take place with incomplete information. To predict whether a provider can offer the service with requested quality levels, a negotiation record is created once an accepted/rejected offer is received. The negotiation record is defined as \( nr_i = (S_i, t_i, S_{Qi}, S_{Oi}) \), consisting of a service template identifier \( S_i \), creation timestamp \( t_i \), negotiated service terms \( S_{Qi} \) and offer status \( S_{Oi} \) (i.e., accepted/rejected). Since outdated records may provide a less accurate reference for negotiation result prediction and the storage of edge devices may be restricted, negotiation records are maintained in a sequence-based queue of size \( m \) containing the \( m \) most recent records. Each record is associated with a
discount factor denoted as $\delta_i$, which simulates the effect of time, using Equation 3.6:

$$\delta_i = \frac{e^{\eta(\tau - 1)} - 1}{e^{-\eta} - 1}$$ (3.6)

where $\eta$ is a positive integer reflecting the fading of impact as time goes (Figure 3.22), and $\tau$ is the relative time calculated by Equation 3.7.

$$\tau = \frac{t_{\text{now}} - t_i}{t_{\text{now}} - t_0}$$ (3.7)

where $t_{\text{now}}$ is the current timestamp, $t_i$ is the timestamp of record $nr_i$, and $t_0$ is the timestamp of the earliest record in the queue.

![Figure 3.22: Impact fading](image)

The probability of a successful negotiation with a service provider (i.e., competence) is measured by the similarity between the request and the recent negotiation offers accepted by the provider$^7$:

$$\text{competence} = \begin{cases} \sum_{i=1}^{k} \delta_i \ast \text{Sim}(R_Q, S_{Q_i})/k, & k \neq 0 \\ 0, & k = 0 \end{cases}$$ (3.8)

where $k$ ($k \leq m$) is the number of acceptable offers among the $m$ records, $\delta_i$ is the discount factor, $\text{Sim}(R_Q, S_{Q_i})$ is the similarities between request service terms $R_Q$ and previously negotiated terms $S_{Q_i}$ in record $i$. The similarity is measured by the modified Eular distance

$^7$If there are no historical records about the service, the probability is set to 0.5.
that considers the importance of each negotiated term in making a successful negotiation:

$$Sim(R_Q, S_{Qi}) = \frac{1}{\sqrt{\sum_{q \in Q} w_q (\tilde{r}_q - \tilde{s}_{qi})^2}}$$  \hspace{1cm} (3.9)

where \( w_q (\sum_{q \in Q} w_q = 1) \) is the importance of each term computed by analysing the negotiation records using Rough set theory, \( \tilde{r}_q \) and \( \tilde{s}_q \) are the normalized requested terms and previously negotiated terms using Equation 3.10.

$$\tilde{r}_q = \frac{r_q}{Max_{i \in [1,k]}(r_q, s_{qi})} \quad \tilde{s}_q = \frac{s_q}{Max_{i \in [1,k]}(r_q, s_{qi})}$$  \hspace{1cm} (3.10)

To calculate the weight of a negotiated term \( w_q \), a \( m \) row decision table is temporarily created by comparing the consumer’s requested terms with the negotiated terms in each record. The decision table is a concept defined in Rough Set Theory representing an information system, which is denoted as \( IS = (U, A, f) \), where \( U \) is the universe domain of discourse with \( m \) objects (i.e., observations in different rows), \( A \) is a finite set of attributes consisting of \( n - 1 \) conditional attributes \( C \) and a decision attribute \( D \) (i.e., attributes in different columns). Each attribute \( a \in A \) has a set of values \( V_a \), \( f \) is the function that denotes the map of \( U \times A \rightarrow V \). The conditional attributes in the decision table correspond to negotiated terms, while the decision attribute corresponds to the negotiation result (i.e., accepted or rejected). The value of each conditional attribute (i.e., negotiated term) is calculated using Equation 3.11 and Equation 3.12 for cost-type terms (i.e., the term where a lower value is preferred, such as price from the consumer’s point of view) and benefit-type terms (i.e., the term where a higher value is preferred, such as availability from the consumer’s point of view) respectively.

$$V_q = \begin{cases} 
0, & \text{if } S_{qi} > Max(R_q) \\
1, & \text{if } S_{qi} \leq Max(R_q)
\end{cases}$$  \hspace{1cm} (3.11)

$$V_q = \begin{cases} 
0, & \text{if } S_{qi} < Min(R_q) \\
1, & \text{if } S_{qi} \geq Min(R_q)
\end{cases}$$  \hspace{1cm} (3.12)
where $Sq_i$ is the value of negotiated term $q$ in each record, $\text{Min}(R_q)$ and $\text{Max}(R_q)$ is the reserved value of the consumer about the term (i.e., bottom line). The value of the decision attribute depends on the status of the associated offer:

$$V_d = \begin{cases} 
0, & \text{if } SO_i = \text{Rejected} \\
1, & \text{if } SO_i = \text{Acceptable}
\end{cases}$$  \hfill (3.13)

Table 3.1 shows a decision table created from 8 recent negotiation records of a particular service, which consists of three conditional attributes (i.e., negotiated terms including availability, latency and price, denoted as $C_i$) and a decision attribute (i.e., negotiation result, denoted by $D$). The value of 0 of a conditional attribute represents that the negotiated term conflicts with the consumer’s negotiation constraint. To identify whether these conditional attributes are important for making decisions, the significance of each conditional attribute is calculated using Equation 3.14:

$$\text{Sig}(c) = \frac{\text{card}(\text{Pos}_{C}(D))}{\text{card}(U)} - \frac{\text{card}(\text{Pos}_{C-\{c\}}(D))}{\text{card}(U)}$$ \hfill (3.14)

where $\text{card}(\cdot)$ denotes the cardinality of a set, $\text{Pos}_{C}(D)$ represents the positive region of the partition $U/D$ where objects are indiscernible by decision attribute $D$ with respect to condition attributes $C$, and $\text{Pos}_{C-\{c\}}(D)$ represents the positive region of the partition $U/D$ when the condition attribute $c$ is removed from $C$. After analyzing the significance of each
negotiated term, Equation 3.15 is used to transform the significance to a weight value whose sum is equal to 1. This weight value is used in Equation 3.9 to reflect the importance of a negotiating term in success rate prediction.

\[
w_c = \begin{cases} 
1/J, & \text{if } \forall c \in C, \text{Sig}(c) = 0 \\
\text{Sig}(c)/\sum_{c \in C} \text{Sig}(c), & \text{otherwise.}
\end{cases} \tag{3.15}
\]

where \( J \) is the number of conditional attributes.

The Appendix A briefly describes the rough set theory, the mathematical expressions of related concepts, and an example of calculating the significance values of different conditional attributes shown in Table 3.1.

**Integrity Assessment**

Once an SLA is negotiated and created, the monitoring engine starts to collect the runtime QoS data according to the assessment information specified in the SLA. The collected observations are analyzed against the guaranteed quality levels to verify SLA compliance of a service provider. Since quality degradation of an IoT service is likely to happen due to fluctuating workloads and unpredictable events such as temporal device malfunction, even though some service providers may have had a good SLA compliance record in the past, a current QoS degradation may lower their integrity level since the service is likely to be abnormal at such a moment. In the trust model, the integrity of a service provider is assessed by summarizing the SLA fulfillment of past executed services (i.e., reputation calculated by Equation 3.23) and the performance of currently running services (i.e., utility calculated by Equation 3.17):

\[
\text{integrity} = \text{utility} \times \text{reputation} \tag{3.16}
\]

These two values are set to 0.5 if there is no historical data or the associated template has no currently running service instances because attaching suspicion to a new provider is inefficient in a market where only a limited number of providers are malicious [Friedman* and Resnick, 2001]. An exponential function similar to Equation 3.6 is used to model the
non-linear relationship between the utility and degradation in service quality as follows:

\[
utility = \frac{0.9e^{5(ρ−1)} - 0.9}{e^{−5} − 1} + 0.1
\]  

(3.17)

where \( ρ \) is the average quality degradation of associated services. A service abnormality is assumed to be identified if the predicted QoS has been more than three standard deviations from its mean value. The degradation in terms of a specific QoS parameter is measured by Equation 3.19 and Equation 3.20 for cost-type parameters and benefit-type parameters respectively:

\[
ρ = \sum_{q_i \in Q} w_{q_i} ρ_{q_i}
\]  

(3.18)

\[
ρ_{q_i} = \begin{cases} 
0, & \text{if } p_{q_i} < μ - 3σ \\
1, & \text{if } p_{q_i} > μ + 3σ \\
\frac{p_{q_i} - μ + 3σ}{6σ}, & \text{otherwise}
\end{cases}
\]  

(3.19)

\[
ρ_{q_i} = \begin{cases} 
0, & \text{if } p_{q_i} > μ + 3σ \\
1, & \text{if } p_{q_i} < μ - 3σ \\
\frac{μ + 3σ - p_{q_i}}{6σ}, & \text{otherwise}
\end{cases}
\]  

(3.20)

where \( p_{q_i} \) is the predicted QoS value, \( μ \) and \( σ \) are the mean and standard deviation of recent QoS values observed by the monitoring engine respectively.

To calculate the reputation value, the parameter \( θ \) is used to represent the possibility of SLA violations of each service provider. By using Bayesian inference, the unknown parameter \( θ \) can be modeled according to a prior distribution and a likelihood function derived from the new observations \( X \):

\[
P(θ|X) = \frac{P(X|θ)P(θ)}{P(X)}
\]  

(3.21)

Since beta distribution is usually used for the conjugate prior distribution of binomial distribution parameters [Fang et al., 2016, Buchegger and Le Boudec, 2003], the beta distribution
is assigned to $\theta$ as the prior that reflects the uncertainty about the SLA violations of a service provider. The probability density functions (PDF) of beta distribution $\text{beta}(\theta|\alpha, \beta)$ is expressed using the gamma function $\Gamma$ with two parameters $\alpha$ and $\beta$:

$$\text{beta}(\theta|\alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1}(1 - \theta)^{\beta-1}$$

(3.22)

where parameters $\alpha, \beta > 0$ represent to what extent the promised QoS has been satisfied during service provisioning time. The reputation score is estimated by the probability expectation value of the beta distribution:

$$\text{reputation} = 1 - E(\theta) = 1 - \frac{\alpha}{\alpha + \beta} = \frac{\beta}{\alpha + \beta}$$

(3.23)

Let $\text{vio}_{\text{rate}}$ denote the observed SLA violation rate reported by the monitoring engine, which is defined by Equation 3.24:

$$\text{vio}_{\text{rate}} = \frac{\text{Number of violated QoS parameters}}{\text{Total number of QoS parameters}} \times 100\%$$

(3.24)

The posterior distribution can be calculated according to Equation 3.21 and the characteristics of beta distribution as follows:

$$\text{beta}(\theta|\alpha', \beta') = \frac{\Gamma(\alpha' + \beta')}{\Gamma(\alpha')\Gamma(\beta')} \theta^{\alpha'-1}(1 - \theta)^{\beta'-1}$$

(3.25)

where $\alpha' = \alpha + \text{vio}_{\text{rate}}, \beta' = \beta + 1 - \text{vio}_{\text{rate}}$. Since the Bayesian inference considers each observation equally regardless of the influence decay of old observations, a moving weighted average is introduced to update the two parameters in beta distribution when a new observation is available [Buchegger and Le Boudec, 2003]:

$$\alpha' = \mu\alpha + \text{vio}_{\text{rate}} \quad \beta' = \mu\beta + 1 - \text{vio}_{\text{rate}}$$

(3.26)
where $\mu$ is the discount factor of past experience calculated using Equation 3.27:

$$\mu = \frac{n - 1}{n} \quad (3.27)$$

where $n$ denotes the number of observations over which the parameter $\theta_i$ can be assumed stationary.

3.4.4 Deadline-aware strategy for IoT service negotiation

Bilateral negotiation is triggered by a follower gateway when the negotiation context is successfully customized with the corresponding service provider. The purpose of the negotiation is to reach an agreement that has the best possible utility through a bargaining process. In each round, negotiating parties perform their own negotiation strategies to evaluate a received offer and make decisions about whether to accept/reject the offer or propose a counteroffer. The sequential offer exchange between negotiation parties creates a negotiation session.

Definition 9. A **Negotiation Session** between a gateway $g$ and a service provider $p$ is defined as a finite sequence of length $n$ on the form $(X_{g\rightarrow p}^{t_1}, X_{p\rightarrow g}^{t_2}, X_{g\rightarrow p}^{t_3}, \ldots)$ with $t_1 < t_2 < \ldots < t_n$, $t_i \in$ negotiation time. $X_{g\rightarrow p}^{t_1}$ is a vector of negotiation offers that proposed by $g$ to $p$ at time $t_1$.

Definition 10. The **Negotiation Offer** proposed by $p$ to $g$ at time $t$ is defined as $x_{p\rightarrow g}^{t}$, the value of negotiable term $j$ offered in $x_{p\rightarrow g}^{t}$ is noted by $x_{p\rightarrow g}^{t}[j]$. Each negotiable term $j$ ($j \in 1, \ldots, k$) has a **negotiation space** noted by $\Omega_j$, which is the collection of possible values of term $j$. In a competitive market, providers may regard some negotiation spaces as business sensitive data and may not be willing to specify them in the constraints, which means the negotiation may occur under an assumption of incomplete information.

Quantitative Evaluation of Negotiation Offers

Many existing negotiation strategies are designed for cloud service negotiations or web service negotiations, which only consider QoS properties. However, apart from the QoS properties,
IoT services may have more negotiable terms such as time and spatial features. To measure the level of satisfaction of each offer, the negotiable terms $j$ in the offer need to be normalized and evaluated by gateways using a scoring function $V^g_j$ ($V^g_j \in [0, 1]$). The higher score represents a higher rate of the term. The definition of negotiable terms and their corresponding scoring functions are defined as follows:

- **Temporality**, which specifies the service provisioning time for a request. To resolve the possible conflicts on temporality, a service provider may specify its negotiation space of temporality ($\Omega^p_t$) in the SLA template or negotiation offers in the form of negotiation constraints. $\Omega^p_t$ can be specified in two ways. (a) Providers may restrict the service start time and service end time in the SLA template (i.e., $\Omega^p_t = [T_s, T_e]$ where $T_s$ represents the range of service start time ($[t_s, t_e]$) and $T_e$ represents the range of service end time ($[t_s, t_e]$). (b) If a consumer’s requirement cannot be fully satisfied, the provider may list all the available time ranges around the requested time slot in the initial offer for gateways to compromise and select the most preferred one (i.e., $\Omega^p_t = [T_1, T_2, ..., T_n]$ ($n \geq 1$) where $T_i$ represents one of the expected time ranges $[t_{i\text{start}}, t_{i\text{end}}]$). The scoring function measures a service’s temporality by calculating the maximum matching degree between the requested time range and the available time of the service, which is defined as [Jin et al., 2014]:

$$V^g_t(x^t_{p \rightarrow g}[T]) = \frac{[t_s, t_e] \cap R_T}{R_T}$$

$$V^g_t(x^t_{p \rightarrow g}[T_i]) = \frac{T_i \cap R_T}{R_T}$$

(3.28)

where $R_T$ is the requested time slot.

- **Service coverage**, which describes the service spatial feature. As described in Section 3.4.1, service coverage is modeled as a circular region by default, which is described by a centre location $l_i$ and cover range $lr_i$. A consumer may provide some negotiation space for spatial requirement, which is defined as $\Omega^l_g = \{loc_r, d\}$ where $loc_r$ represents the requested location, and $d$ represents the acceptable distance. The scoring function measures a service’s spatial feature by calculating the distance between the requested
Chapter 3. Design

area and the service area, which is defined as [Jin et al., 2014]:

$$V_g^i(x_p \rightarrow y[L]) = \begin{cases} 1, & \text{dist}(loc_r, l_i) < l_r_i \cr 1 - \frac{\text{dist}(loc_r, l_i)}{d + r}, & l_r_i \leq \text{dist}(loc_r, l_i) < d + l_r_i \cr 0, & \text{otherwise} \end{cases} \tag{3.29}$$

where \(\text{dist}(loc_r, l_i)\) represents the distance between the requested location and the centre location of service coverage.

- **QoS Properties**, which specifies the service’s non-functional properties such as availability, latency, reliability, data transmission rate, sampling rate, etc. In general, consumers and providers may have conflicting QoS interests, e.g., a consumer hopes to obtain a service at a lower price but higher availability, whereas the provider attempts to offer the service at a higher price but lower availability. The negotiation space of a QoS property is defined as \(\Omega^q = [\text{min}_q, \text{max}_q]\), presenting the range of preferred value and reserved value (i.e., bottom line). A QoS property is evaluated by the scoring function defined in Equation 3.30 for benefit type properties and Equation 3.31 for cost type properties respectively [Faratin et al., 1998]:

$$V_q^g(x_p \rightarrow y[j]) = \begin{cases} 1, & \text{if } x_p \rightarrow y[j] > \text{max}_j^q \cr 0, & \text{if } x_p \rightarrow y[j] < \text{min}_j^q \cr \frac{x_p \rightarrow y[j] - \text{min}_j^q}{\text{max}_j^q - \text{min}_j^q}, & \text{otherwise} \end{cases} \tag{3.30}$$

$$V_q^g(x_p \rightarrow y[j]) = \begin{cases} 1, & \text{if } x_p \rightarrow y[j] < \text{min}_j^q \cr 0, & \text{if } x_p \rightarrow y[j] > \text{max}_j^q \cr 1 - \frac{x_p \rightarrow y[j] - \text{min}_j^q}{\text{max}_j^q - \text{min}_j^q}, & \text{otherwise} \end{cases} \tag{3.31}$$

Based on these scoring functions, each received offer can be quantitatively evaluated by
the weighted sum of the score values:

\[ V^g(x^t_{p \to g}) = \sum_{j=1}^{k} w_i \cdot V^g(x^t_{p \to g}[j]) \]  

(3.32)

where \( k \) is the total number of negotiated terms and \( w_i \) is the weight of term \( j \) \( (\sum_{j=1}^{k} w_i = 1) \). If the value is not specified by the consumer, \( w_i \) can be estimated by \( 1/k \).

Apart from the weighted sum of scores, an offer can also be evaluated using the cost performance index (CPI), which is defined as:

\[ CPI(x^t_{p \to g}) = \frac{V^g(x^t_{p \to g})}{p_{\text{rice}}} \]  

(3.33)

where \( p_{\text{rice}} \) is the price of the service.

**Deadline-aware Decision-making Model**

Bilateral negotiation starts when a follower gateway sends the initial offer to the service provider. Each time a gateway \( g \) receives one or more counteroffers \( X^t_{p \to g} \) from provider \( p \), it first validates them based on constraints specified in the originating template or the previous offer. Then it evaluates the offers using Equation 3.32 and selects the offer with the highest score \( x^t_{p \to g} \) to make decisions (i.e., reject/accept the received offer or propose a counter offer). The decision-making process is controlled by the offer state transition model specified in WSAN [Waeldrich et al., 2011]. WSAN defines four states for each negotiation offer, which indicates the subsequent interaction mode after receiving the offer. The *Advisory* state indicates multiple back-and-forth interactions. The *Solicited* state indicates the single request-reply interaction is expected and the opponent wants to converge the negotiation process. The *Acceptable* state means all the negotiated terms specified in the previous offer are acceptable. The *Rejected* state means the previous offer is rejected, no further negotiation is necessary. Based on the state transition rule, a deadline-aware decision-making model on gateways takes actions at each round. As Figure 3.23 shows, the negotiation session is terminated when the provider only proposes a rejected offer or an acceptable offer is
achieved. If an advisory offer is received, a negotiation tactic is performed to adjust the current expectations of negotiated terms. The adjusted expectations are further compared with the previous counteroffer proposed by the opponent to select the solution that has a higher score. If there is no space and time for further negotiation, a solicited offer is created. If the negotiation session is approaching the deadline, to increase the success rate, the current proposal is modified by replacing the values of conflicting terms with reserved values. This guarantees a solution is found when negotiating parties have intersected negotiation space.
Context-based QoS Negotiation Tactic

A gateway’s negotiation goal is to find an acceptable deal that satisfies the negotiation constraints of a consumer but which, on the other hand, maximizes the utility. As described in Section 2.3.2, a combination of concession and tradeoff tactics can help to balance negotiation utility and success rate [Zheng et al., 2014]. The concession tactic means a negotiation party gradually reduces its utility until all conflicts are resolved, which is likely to reach an agreement since the new proposals are always more preferable for its negotiation opponent. The tradeoff tactic means a party yields on its less important negotiable terms and demands more on the important ones, which maintains the utility but lowers the success rate since the new proposal may be less beneficial to the opponent when negotiation participants have incomplete information about each other. iNegotiate uses a context-based negotiation tactic to make concessions and tradeoffs during the bilateral negotiation process. This tactic employs a utility function to control the concession rate according to context information such as time, the number of candidates, and a consumer’s preference. Based on Weber-Fechner’s law that describes the relationship between the response of an organism and stimulus caused by environments [Dehaene, 2003], an exponential utility function is defined to determine the
level of satisfaction in each negotiation round:

\[ U^g_j(x^t_{p ightarrow g}[j]) = \frac{e^{\beta V^g_j(x^t_{p ightarrow g}[j])} - 1}{e^\beta - 1} \]  

(3.34)

where \( V^g_j \) is scoring value of term \( j \) computed by the scoring function \( (V^g_j \in [0, 1]) \), \( \beta \) (\( \beta \in \mathbb{R} \) and \( \beta \neq 0 \)) determines the convexity degree of the curve, which controls the concession rate. Figure 3.24 shows how utility varies with the score value under different \( \beta \). When negotiation starts, utility equals to 1. By making concessions, the utility gradually decreases until it reaches 0. When \( \beta < 0 \), a small decrease in utility causes a significant decrease in the term’s score in the early negotiation stage, which means the concession rate is high in the beginning, but decreases as negotiation proceeds (e.g., when \( \beta = -8 \), if utility decreases from 1 to 0.9, the term’s score decreases from 1 to nearly 0.28). When \( \beta > 0 \), the concession rate is small in the beginning and increases gradually. Since a bigger concession rate accelerates the negotiation convergence, \( \beta \) is dynamically set based on a consumer’s negotiation desirability\(^8\) (DF, \( 0 < DF \leq 1 \)) and negotiation preference (i.e., the importance of negotiable terms \( w_j \)):

\[
\beta = \begin{cases} 
0.7e^{5(0.5-DF)} + 0.3e^{3w_j}, & DF \leq 0.5 \\
-0.7e^{5(0.5-DF)} - 0.3e^{3(1-w_j)}, & DF > 0.5 
\end{cases}
\]  

(3.35)

Algorithm 5 illustrates the proposed negotiation tactic, which is called UMI. This tactic chooses to make concessions with a probability calculated from the number of candidates (Line 1-3). If more candidates are available in the environment, more tradeoffs are made. However, if there is no space for tradeoff, the negotiation tactic chooses to play concession instead (Line 9-12). If there is no space for concession, the negotiation tactic returns false, indicating current proposal is the ultimatum and no more concessions can be made (Line 15-16). Figure 3.25 shows the flowcharts of playing concession and tradeoff tactics. The parameter \( n_i \) and \( \lambda \) represent the number of rounds that played concession/tradeoff tactics and the scale factor of the concession rate respectively. Here a greedy concession tactic is

\(^8\)The negotiation desirability reflects how eager a consumer wants to reach to an agreement as soon as possible
Algorithm 5 Perform Context-based Negotiation Tactic (UMI)

**Input:** number of candidates $N$, negotiable terms in received offer $x_{p\rightarrow g}^t[J]$, negotiable terms in last proposal $x_{g\rightarrow p}^{t-1}[J]$, negotiation desirability $d_f$, the number of rounds that played concession $k_c$ and the number of rounds that played tradeoff $k_t$, negotiation request $R_{eq}$.

**Output:** array $V'$ with values of terms produced by the tactic, a boolean value indicating if the tactic can be successfully performed.

1: $r \leftarrow \text{Random}(0,1)$
2: $P_{tr} \leftarrow 1 - 1/N$
3: if $r < 1 - P_{tr}$ then
4:   $k_c \leftarrow k_c + 1$
5:   $V' \leftarrow \text{playConcession}(k_c,d_f,x_{p\rightarrow g}^t[J],x_{g\rightarrow p}^{t-1}[J])$
6: else
7:   $k_t \leftarrow k_t + 1$
8:   $V' \leftarrow \text{playTradeOff}(k_t,d_f,x_{g\rightarrow p}^{t-1}[J])$
9: if $V'$ equals $x_{g\rightarrow p}^{t-1}[J]$ then
10:   $k_c \leftarrow k_c + 1$
11:  $k_t \leftarrow k_t - 1$
12:  $V' \leftarrow \text{playConcession}(k_c,d_f,x_{p\rightarrow g}^t[J],x_{g\rightarrow p}^{t-1}[J])$
13: end if
14: end if
15: if $V'$ equals $x_{g\rightarrow p}^{t-1}[J]$ then
16:   return false
17: else
18:   return true
19: end if

adopted to maintain the highest possible utility, which determines the degree of concession based on the comparison between the current expectation and the offer proposed by the opponent.

**Metaheuristic QoS Negotiation Tactic**

The UMI tactic relies on context information and consumers’ negotiation preference to adjust the concession rate, which may not be efficient if a consumer’s negotiation preference is unknown or the consumer treats all negotiable terms equally. From another perspective, negotiation with incomplete information can be modeled as an optimization problem, which tries to find the best solution that has the maximum utility for both negotiation parties from all feasible solutions under the partially known negotiation constraints. As described in
Chapter 2.3.2, metaheuristic approaches such as genetic algorithm (GA) and particle swarm optimization (PSO) have been applied in automatic negotiations to find a Pareto optimal solution. However, these approaches are either computationally intensive, or requires multiple negotiation rounds to find the optimal solution. Due to the simplicity and accuracy of ABC algorithm in terms of addressing multivariable problems [Karaboga and Akay, 2009], in this work, a modified Artificial Bee Colony Optimisation (ABC) algorithm is explored to seek a win-win solution from the solution domain.

The ABC algorithm abstracts solutions as food sources. The task of searching for food sources is performed by three types of specialized bees: scout bees, employed bees, and onlooker bees [Zhou et al., 2016]. The bees work cooperatively to find a food source that has the maximum fitness. The scout bees randomly search the environment to find a new food source. The employed bees exploit the discovered food sources and give information back to onlooker bees which are waiting in the hive. Based on the nectar amount and the position information, the onlooker bees decide if further exploitation is needed. Each food source is associated with an employed bee and an onlooker bee. Once a food source is exhausted and...
Algorithm 6 Gateway: Perform ABC-based Tactics

Input: Solutions $F[10]$, received offer of last round $x_{p\rightarrow g}^{r-1}$, the number of current round $r$, loop limit $Loop_{max}$, request $req$, similarity factor $\alpha$, best solution $F_{best}$

Output: The vector of updated negotiable terms $F_{best}$

1: $\alpha \leftarrow$ updateSimilarityFactor($r$)
2: if $F[n]$ is null then
3: Initialize solutions and bees ($req, x_{p\rightarrow g}^{t}$)
4: end if
5: Evaluation of solutions ($\alpha, req, x_{p\rightarrow g}^{t}$)
6: cycle $\leftarrow$ 0
7: while cycle $\leq Loop_{max}$ do
8: Employed bees phase($\alpha, req, x_{p\rightarrow g}^{t}$, $F[n]$)
9: $P_r \leftarrow$ Calculate selection probabilities($\alpha, F[n]$)
10: if Random($0, 1) < P_r$ then
11: Onlooker bees phase($\alpha, req, x_{p\rightarrow g}, F_{best}$)
12: end if
13: for all $F[i] \in F$ do
14: if $F[i].T_r > Tr_{max}$ then
15: Scout bee phase ($req, x_{p\rightarrow g}, F_{best}, F[i]$ )
16: end if
17: end for
18: Evaluation of solutions ($\alpha, req, x_{p\rightarrow g}$)
19: $F_{best} \leftarrow$ memorizeBestSolution($F[n]$)
20: end while
21: return $F_{best}$

abandoned by its employed bee, the employed bee transforms into a scout bee.

Algorithm 6 shows how negotiation gateways use ABC optimization to update the expectations of negotiable terms. Different combinations of negotiable terms comprises the solution domain. A possible solution is modeled as the position of a food source, which is evaluated by a fitness function. The ABC-based negotiation tactic defines each food source as $F = \{position, bee_e, bee_o, fit, T_i\}$. position is a k-dimension vector $\vec{V}_i = (v_{i,1}, v_{i,2}, ..., v_{i,k})$ representing a possible solution, which contains the expected values of negotiable terms ($i \in [1, n]$, $n$ is the number of food sources, $k$ is the number of negotiable terms), bee_e and bee_o are the associated employed bee and onlooker bee respectively, fit is the fitness value and $T_i$ is the number of times that a solution has been exploited. Initially, ten solutions are generated based on the user’s most preferred values and the first received counter
offer (Line 3). The elements in initial position vector $\vec{V}_i$ are computed using Equation 3.36:

$$\bar{v}_{i,j} = N\left(\frac{v_{prf,j} + x_{i,p\rightarrow g}^{I}[j]}{2}, |v_{prf,j} - x_{i,p\rightarrow g}^{I}[j]|\right)$$

$$v_{i,j} = \text{Min}\{\text{Max}\{\bar{v}_{i,j}, min_j\}, max_j\}$$

(3.36)

where $N(\cdot)$ denotes a Gaussian distribution with mean $(v_{prf,j} + x_{i,p\rightarrow g}^{I}[j])/2$ and variance $|v_{prf,j} - x_{i,p\rightarrow g}^{I}[j]|$, $v_{prf,j}$ is the user’s most preferred value of term $j$. $max_j$ and $min_j$ is the upper and the lower boundaries of the negotiation space of term $j$. The reason using Gaussian distributed values rather than the randomly distributed values defined in standard ABC algorithm is that this approach helps to avoid too much concessions in the early stage when the maximum number of iterations is limited. The standard ABC algorithm usually has thousands of iterations, which may introduce a large latency. Reducing the number of iterations reduces the solution accuracy, but decreases the computation complexity as well. To make the algorithm more lightweight, in each round, the maximum number of loops (Line 7) is defined as:

$$\text{Loop}_{max} = 2(mt + r)$$

(3.37)

where 2 is the scale factor, $r$ is the current number of negotiation rounds, $mt$ is a constant positive integer representing the minimum times a solution can be exploited initially. The sum of $mt$ and $r$ is defined as the limit of exploitation times as the negotiation processes. Equation 3.37 shows that more loops is introduced when $r$ is increasing.

Through the next repeated cycles, the ten solutions are modified by the searching processes of different bees (Line 8-17) and evaluated according to a fitness function (Line 18). The particular mechanism for finding a win-win solution is that each solution is evaluated by its utility and the absolute cosine similarity between the current solution and the counteroffer
proposed by the opponent. The fitness function of solution $\vec{V}_i$ is defined as:

$$fit(\vec{V}_i) = \begin{cases} 0, & V^g(\vec{V}_i^t) > V^g(\vec{V}_i^{t-1}) \\ f(\alpha, \vec{V}_i, x_{p\rightarrow g}^{t-1}), & \text{otherwise.} \end{cases}$$ \hspace{1cm} (3.38)$$

$$f(\alpha, \vec{V}_i, x_{p\rightarrow g}^{t-1}) = (1 - \alpha) \times V^g(\vec{V}_i^t) + \alpha \times \text{sim}(\vec{V}_i^t, \vec{S})$$

where $V^g(\vec{V}_i^t)$ is the score of $\vec{V}_i$ at time $t$ computed by Equation 3.32. $\vec{S}$ is the normalized vector of opponent’s expectation, which is extracted from the optimal counteroffer of last round $x_{p\rightarrow g}^{t-1}$. $\vec{V}_i$ is the normalized $\vec{V}_i$. $\alpha$ is the similarity factor representing the weight of making concessions ($\alpha \in (0, 1)$), which gradually increases from $C_0$ to $C_1$:

$$\alpha = C_0 + C_1 \frac{e^{\beta \hat{r}} - 1}{e^{\beta} - 1}, (0 < C_0 + C_1 < 1)$$ \hspace{1cm} (3.39)$$

where $\hat{r}$ is the ratio of current round to the maximum negotiation round. $\beta$ is an integer that controls the change rate of $\alpha$ ($|\beta| < 10$). The negative $\beta$ controls $\alpha$ increasing quickly at the beginning but getting slower as the negotiation proceeds, while positive $\beta$ does the opposite. In other words, the negotiation is more conservative when $\beta$ is positive. Equation 3.38 and Equation 3.39 shows that the weight of fitness evaluating criteria are dynamically changing as the negotiation proceeds. Also, the fitness is set to zero when the solution has higher utility than the last proposal since the solution is likely to be rejected by the opponent. The fitness function illustrates why more iterations are needed as the negotiation rounds increase (Equation 3.37). This avoids the tactic being too conservative in the early stage, and increases the chance of finding a better solution that has higher utility when the concession rate increases.

In the repeated iteration, the searching process is performed in three phases: employed bee phase (Line 8), onlooker bee phase (Line 10-11), and scout bee phase (Line 13). These phases are introduced as follows:

- **Employed Bee Phase:** Each employed bee searches for a new solution depending on...
the current solution $\vec{V}_i$ and another random solution $\vec{V}_m$ ($k \in \{1, ..., n\}, k \neq i$). For all elements $v_{i,j}$ in $\vec{V}_i$, new values are generated as follows [Zhou et al., 2016]:

$$\bar{v}_{i,j} = v_{i,j} + [2\text{Random}(0,1) - 1](v_{i,j} - v_{m,j})$$

$$v'_{i,j} = \text{Min}\{\text{Max}\{\bar{v}_{i,j}, min_j\}, max_j\}$$

(3.40)

where $v_{m,j}$ is the value of term $j$ in $\vec{V}_m$ ($j \in \{1, ..., k\}$), $\text{Random}(0,1)$ is a uniformly distributed random number in the range $[0, 1]$. If the new solution $\vec{V}'_i$ has a higher fitness, it replaces the old $\vec{V}_i$. Otherwise the exploitation time of the solution increases by 1.

- **Onlooker Bee Phase:** After an employed bee completes its searching process, the information of the current solution is shared with the associated onlooker bee. The onlooker bee decides whether or not to exploit it based on the probability computed by fitness:

$$\text{Prob}(F_i) = \frac{\text{fit}(F_i)}{\text{Max}_{l \in n}\{\text{fit}(F_l)\}}$$

(3.41)

Considering the limitation on the number of iterations, we use the maximum fitness as the denominator instead of using the sum of fitness defined in the standard ABC algorithm to increase the chance of discovering a better solution. Based on the probability and Roulette-wheel selection mechanism, the onlooker bee may further produce a modification on the current solution by following the same searching process defined in the employed bee phase.

- **Scout Bee Phase:** After all the onlooker bees are distributed, the solution whose exploitation time reaches the limit is regarded as exhausted, and the corresponding employed bees turn into scout bees to find a new solution. In the classic ABC algorithm, the scout bee randomly chooses a solution that satisfies the boundary constraint:

$$v'_{i,j} = \text{min}_j + \text{Random}(0,1)(\text{max}_j - \text{min}_j)$$

(3.42)
However, when the number of iterations is limited, a more efficient searching mechanism that can accelerate the convergence process is required. Inspired by the bare bones particle swarm algorithm [Kennedy, 2003], a Gaussian bare-bone searching equation is proposed by utilizing the global best solution and received counteroffer. For all elements $v_{i,j}$ in $\vec{V}_i$, new values are generated as follows:

$$v_{i,j} = N\left(\frac{v_{\text{best},j} + x_{p\rightarrow g}^{t-1}[j]}{2}, \left|v_{\text{best},j} - x_{p\rightarrow g}^{t-1}[j]\right|\right) \tag{3.43}$$

$$v'_{i,j} = \text{Min}\{\text{Max}\{\bar{v}_{i,j}, min_j\}, max_j\}$$

where $v_{\text{best},j}$ is the value of term $j$ in the global best solution. The solution $\vec{V}'_i$ is further compared with the random solution generated by Equation 3.42 to select the solution that has the higher fitness.

### 3.5 Chapter Summary

This chapter introduces *iNegotiate* and the design of its contributions that address open issues of SLA negotiation in the IoT environment.

To achieve semantic interoperability of automatic SLA management, *iNegotiate* formalizes SLAs of IoT services based on existing standard web service SLA specifications and widely-used IoT ontology models. To enable the distributed SLA negotiation in a dynamic large-scale environment, *iNegotiate* configures a network of negotiation gateways that autonomously manage registered SLA templates and coordinate negotiation tasks without a centralized infrastructure. In order to get the best possible solution within the limited negotiation time, *iNegotiate* uses an experience-based trust model to prioritize candidate service providers if a gateway has multiple candidates to negotiate with. The candidate that has a higher chance to make an agreement and to comply with the negotiated result is more likely to engage with the bilateral negotiation. To balance the trade-off between negotiation utility and success rate for negotiations with incomplete information, negotiation gateways use a deadline-aware negotiation strategy to dynamically adjust concession rate according to the
context information or the negotiation opponent’s behaviour. When the pre-defined negotiation time is due, *iNegotiate* aggregates negotiated solutions achieved by different gateways and returns the optimal solution to the consumer. Figure 3.26 compares *iNegotiate* features with the other approaches from the literature. The contributions of *iNegotiate* close the gap of current approaches with regard to SLA modeling and negotiation in dynamic large-scale IoT environments.
Chapter 4

Implementation

The design described in Chapter 3 illustrates how iNegotiate addresses SLA negotiation challenges in dynamic large-scale IoT environments. This chapter discusses the implementation of iNegotiate. Section 4.1 introduces the detailed architecture of iNegotiate. Section 4.2 presents the implementation of the iNegotiate model in the context of the Simonstrator platform. The details of the data model and operations performed by the system components are discussed in the section. Finally, Section 4.3 summarises this chapter.

4.1 iNegotiate Architecture

Figure 4.1 illustrates the architecture of iNegotiate, which is composed of five main components as follows:

**Interaction Handler**

This component manages the interactions between service consumers, service providers, and gateways. Messages are passed to different sub-components based on the message type and message operation code. The interaction handler has four sub-components:

- **System Configuration Manager:** This component processes configuration messages during the system initialization phase, which creates the negotiation overlay network by autonomously assigning controllers and coordinators according to Algorithms 2, 3 and 4 in Chapter 3.4.2. It maintains the gateway’s role information after the initialization
phase, which is used by other sub-components to correctly pass messages to different system components. For a follower gateway, the manager maintains its controller’s contact information including the communication interface (e.g., WiFi and Ethernet), network address, port number, and route table. For a controller gateway, the manager maintains the location and contact information of its followers, as well as the contact information of its coordinator. For a coordinator gateway, the manager maintains its controllers’ location, range and contact information.

- **Template Manager**: This component manages the template distribution phase in which the template registration requests are forwarded to different gateways according to the template distribution mechanism described in Figure 3.16, Chapter 3.4.2. The processing of template registration messages varies according to the gateway’s roles.
and the message’s operation code. Also, the component periodically checks the network connections with service providers and removes the templates whose providers are mobile and unreachable.

• **Request Manager**: This component manages the negotiation task allocation phase in which negotiation tasks are assigned to the gateways close to the requested service locations according to the request forwarding mechanism described in Figure 3.17, Chapter 3.4.2. This component collects the negotiated solutions achieved by different gateways, and returns the optimal solution to service consumers. It may also trigger a consumer locating process if the consumer is mobile and its network connection has changed after submitting a request. The processing of involved messages (i.e., negotiation request messages, mobile entity locating messages and signing request messages) varies according to the gateway’s roles and the message’s operation code.

• **Negotiation Manager**: This component manages the bilateral negotiation phase. It processes handshakes (i.e., negotiation customization) with candidate providers and passes negotiation messages to the negotiator instance for further processing. It may also trigger a provider locating process if there is no response during the negotiation customization phase. The processing of involved messages (i.e., negotiation customization messages, mobile entity locating messages and negotiation messages) varies according to the gateway’s roles and the message’s operation code.

**SLA Manager** This component manages SLA/templates and monitoring instances of currently executing SLAs. It consists of a template repository, an SLA repository, an SLA monitoring engine, and a QoS prediction engine. The template repository saves registered templates. If the gateway is a controller, the template repository also saves backup templates registered in its followers. The SLA repository manages outstanding SLAs by providing interfaces such as adding a new SLA and removing an existing SLA. Once an SLA comes into effect, the SLA monitoring engine initiates a new monitoring instance to collect observable run-time QoS according to the assessment information specified in the SLA. The QoS prediction engine uses QoS observations to predict possible quality degradation in the short future,
which is used by the trust evaluator to assess the integrity of a service provider. As described in Chapter 1.7, details about SLA monitoring and QoS prediction engines are outside the scope of this thesis.

**Template matchmaker**

This component identifies SLA templates of candidate service providers based on a request from a service consumer. This component consists of a service discovery engine and a candidate selector, which provide interfaces to discover and select candidate service providers. As described in Chapter 3.4.1, considering the computation capability of resource-constrained gateways, a multi-phase template matchmaking process is performed to gradually narrow down the search space according to the WIoT-SLA ontology. In iNegotiate, controllers use the service discovery engine to search for candidates based on the semantic relations of services’ input and output signatures. Followers use the candidate selector to filter the candidates based on the spatial/time features and semantic similarities. If multiple candidates are identified after the filtering process, the candidate selector ranks the candidates according to the trust credit provided by the trust evaluator and returns the Top-K candidates to the request manager. As described in Chapter 1.7, details about the service discovery engine is outside the scope of this thesis.

**Trust Evaluator**

If a follower has multiple negotiation candidates, the trust evaluator would be queried to provide a trust credit to prioritize these candidates. This component consists of three sub-components: compliance records repository, negotiation records repository, and credit calculator. The credit calculator implements the trust model described in Chapter 3.4.3, assessing the trust credit of each candidate according to its historical negotiation and SLA compliance records. When an SLA is terminated, the SLA manager reports the SLA violation rate to the trust evaluator using the update interface. When a negotiation session finishes, the negotiator submits the accepted or rejected offer to the trust evaluator using the addRecord interface.

**Negotiator**

This component manages a negotiation session during the bilateral negotiation phase.
The negotiator instance is instantiated after a successful handshake with a candidate service provider, and saves the context information such as template id, negotiation deadline, received counteroffers, current negotiation round and previous proposals. It consists of three subcomponents: offer evaluator, decision-maker and proposal generator. The decision-maker implements the decision-making model logic outlined in Figure 3.23, and relies on the offer evaluator and proposal generator to assess received offers and propose new offers. The offer evaluator implements the functions that quantitatively evaluate an offer, including the scoring functions and the cost performance index defined in Chapter 3.4.4. The proposal generator implements the negotiation tactics that adjust expectations in each round, including the concession tactic, tradeoff tactic, context-based tactic, and ABC-based tactic.

4.2 Implementation of iNegotiate Model

This section illustrates the implementation of the iNegotiate model using the Simonstrator platform, which is an event-based simulator for distributed mobile applications implemented using Java [Richerzhagen et al., 2015]. The Simonstrator platform was selected because it provides a light-weight framework for developing distributed systems catering for mobile/static devices at different scales and devices’ unstable connectivity. This facilitates the evaluation of new protocols or system prototypes in a dynamic environment. Figure 4.2 shows the implementation of the iNegotiate model using the Simonstrator platform, which illustrates iNegotiate’s dependencies on core peculiarities provided by the Simonstrator framework including components, scheduling and instrumentation. iNegotiate’s active entities (i.e., gateway, service consumer and service provider) are modeled using the Host interface, which acts as the container that bridges required components. In Simonstrator, platform-specific functionalities such as sensor, service and network are provided by loosely coupled components. The composition of different components determines the host’s functionality within the distributed system. All active entities utilize run-time environment components, providing transport and network layer functionalities. The service providers and gateways furthermore utilize location sensors to facilitate template registration and negotiation task allocation. The sequential
Figure 4.2: iNegotiate prototype implemented in Simonstrator platform activities in iNegotiate are implemented in operations. Simonstrator’s scheduling mechanism provides relative time calculation required for delayed execution of customized operations, which guarantees the deterministic execution order of activities shown in Figure 4.3. The details of operations performed by iNegotiate components will be illustrated in Section 4.2.2. Simonstrator’s instrumentation component offers logging features for applications to analyze system performance. The push-based message analyzer is implemented to measure iNegotiate’s network overhead. The analyzer is triggered when a message has been sent, received and dropped. The details of iNegotiate messages will be illustrated in Section 4.2.1.

4.2.1 iNegotiate Data Model

Figure 4.4 shows the data model for iNegotiate messages. Each message has a sender/receiver identification that contains the contact information of an entity (i.e., a service consumer/provider or a gateway), and an operation ID used by the analyzer for instrumentation. As described in Chapter 3.4.2, these messages are:

- **Ping Message:** This message is used by the interaction handler to test network connections in neighbour inquiry operation, template verification operation and mobile entity locating operation. A ping message’s content is a “hello” string.

- **Configuration Message:** This message is used by the system configuration man-
ager to create a negotiation overlay network during the system initialization phase. A configuration message contains an operation code, a message propagation route, the maximum hop limit \( \text{ttl} \) and a message content specifying information relating to the advertised controller or coordinator (i.e., the number of connections and route). The pre-defined operation code set is \( \{ \text{CIM, CVM, RIM, RIMB, RVM, FRM} \} \). The operations associated with the message and the functionality represented by each operation
Figure 4.4: *iNegotiate* - messages

code are generalized in Table B.1 in Appendix B.

- **Template Registration Message**: This message is originally created by a service provider to register their SLA templates. This message is forwarded within the negotiation overlay network by the *template manager* to distribute the template according to the advertised service location. Also, this message can be created by a follower gateway to request its controller to remove a backup template if the provider of the template is mobile and unreachable. A template registration message contains an operation code, an SLA template, a message propagation route, and a customized message content created during the template distribution process, which specifies the minimum distance between a gateway and the requested service location. The pre-defined operation code set is \{ADV, TRG, CREG, RREG, FREG, REG, BAK, DEL\}. The operations associated with the message and the functionality represented by each operation code are generalized in Table B.2 in Appendix B.

- **Negotiation Request Message**: This message is originally created by a service con-
sumer to submit a negotiation request. This message will be forwarded within the negotiation overlay network by the request manager to distribute the request to gateways that are likely to solve the request. The negotiation request message contains an operation code, a request specifying functional and non-functional requirements, a message propagation route, and a customized message content created during the negotiation task allocation phase, which specifies the initiating gateway’s identifier or the candidate templates’ identifiers. The pre-defined operation code set is \{SUB, TREQ, CREQ, INS\}. The functionality represented by each operation code is generalized in Table B.3 in Appendix B.

- **Negotiation Customization Message:** This message is created by the negotiation manager to start a handshake with a candidate service provider. The negotiation customization message contains an operation code, the SLA template to negotiate, and a customized message content specifying the negotiation context such as negotiation protocol, SLA schema, and communication interface (e.g., WiFi, Ethernet, cellular data, etc.). The pre-defined operation code set is \{CUT, OK, REJ\}. The functionality represented by each operation code is generalized in Table B.4 in Appendix B.

- **Negotiation Message:** This message is exchanged between the negotiation manager and a candidate service provider during the bilateral negotiation phase. Also, this message can be created by a follower gateway to return the negotiation result to the initiating coordinator. The negotiation message contains an operation code, the negotiation instance ID, a set of negotiation offers, and the route to the initiating coordinator. The pre-defined operation code set is \{NEG, RES\}. The functionality represented by each operation code is generalized in Table B.5 in Appendix B.

- **Mobile Entity Locating Message:** This message can be either created by the negotiation manager of a follower gateway to locate a mobile service provider if there is no response during the handshake process, or created by the request manager of a coordinator gateway to locate a mobile service consumer if the consumer did not acknowledge the signing request. The mobile entity locating message contains an operation code,
the entity identifier, a message propagation route, and a customized message content specifying a handover request or message. The pre-defined operation code set is \{FTC, FTR, RINQ, CINQ, FINQ\}. The functionality represented by each operation code are generalized in Table B.6 in Appendix B.

- **Signing Request Message:** This message is created by the request manager of a coordinator gateway to send back the negotiation result to the consumer for approval. The signing request message contains a negotiated solution and a message content specifying whether or not the negotiation result is approved by the user.

### 4.2.2 \textit{iNegotiate} Class Diagram and Operations

Figure 4.5 illustrates \textit{iNegotiate}’s class diagram. During the system initialization phase, the system configuration manager first executes the neighbour inquiry operation to detect neighbouring gateways. Then it executes the controller allocation operation to search for a controller. Figure 4.6 shows the sequence diagram for the controller allocation operation. Each gateway initializes the cache with its own information and advertises the cached information to neighbours. The cached information is updated and propagated to neighbours once a better controller is identified. When the time is due and the cached controller is not itself, the gateway sends a configuration message to the cached controller to verify the commission.

If a gateway is assigned as a controller, the system configuration manager executes the coordinator inquiry operation to search for a coordinator. Figure 4.7 shows the sequence diagram for the coordinator allocation operation. A controller gateway multicasts a configuration message to neighbours. Receivers forward the message to their neighbours until the maximum hop limit is reached. During the message forwarding process, an Internet-connected gateway collects senders’ information for a short time (i.e., message synchronizing), and sends a reply to the controller advertising itself as a candidate coordinator. The controller updates the cache if the advertised coordinator is better. When the time is due, the controller sends a configuration message to the cached coordinator to verify the commission. Once a gateway is
Figure 4.5: iNegotiate - class diagram
selected as a coordinator, it broadcasts a configuration message to query other coordinators’ information.

After performing these two operations, the negotiation overlay network is created. The template manager executes the template distribution operation to distribute received templates to different gateways. Figure 4.8 shows the sequence diagram of this operation. The objects marked in blue, yellow, and red represents a follower, a controller, and a coordinator respectively. Firstly, the template registration request is forwarded to a coordinator. The coordinator identifies a gateway close to the service location in its managed sub-areas and asks other coordinators if they can find a gateway that is closer to the service location. When the time is due, the request is forwarded to the coordinator that reports the minimum distance. The coordinator further forwards the request to the follower that is closest to the service location through a controller. The follower registers the template in its local repository. If the provider is mobile, the template manager periodically checks the network connection with
When a consumer submits a negotiation request, the request manager forwards the negotiation task to different gateways by executing the negotiation task allocation operation. Figure 4.9 shows the sequence diagram for this operation. Firstly, the negotiation request is forwarded to a coordinator that acts as the initiating coordinator. The initiating coordinator propagates the request to other coordinators to identify the controllers whose range covers the requested service location. These controllers pass the request to their template matchmakers to discover the candidate templates by matching the input and output parameters. Then these controllers forward the request and candidate templates’ identifiers to the followers that register these candidate templates. The follower gateways execute the negotiation operation to process the request. The request manager of the initiating coordinator collects the negotiation results returned from different followers and sends back the solution with
Figure 4.8: iNegotiate - template distribution operation
the highest utility to the consumer when the time is due. If the consumer is mobile and the network connection has changed, the request manager executes the mobile entity locating operation to locate the consumer.

Figure 4.10 shows the sequence diagram for the negotiation operation. The follower selects the candidates by passing the request and candidates’ identifiers to the template matchmaker. The matchmaker retrieves the candidate templates from the SLA manager and filters them by performing semantic matchmaking. If a follower has multiple negotiation candidates, it passes the candidate template to the trust evaluator to prioritize them before starting bilateral negotiations. Then the follower sends the negotiation customization messages to the Top-K candidates as a handshake. If a successful handshake signal is received, the follower creates a new negotiator instance, which manages the negotiation session defined in Chapter 3.4.4. The negotiator evaluates received offers and proposes new offers by performing the pre-defined negotiation tactic. Once an acceptable offer is achieved, the negotiation result is recorded in the trust evaluator and returned to the initiating coordinator.

Figure 4.11 shows the sequence diagram for the mobile entity locating operation that locates a consumer after negotiation. The initiating coordinator triggers the process by sending a mobile entity locating message to its controllers. Each controller forwards the message to its followers, which contact the consumer by sending ping messages. If a follower receives an ACK from the consumer, it sends the signing request message to the consumer and replies to the initiating gateway. If the initiating gateway does not receive any response when the time is due, it propagates the message to other coordinators to search for the consumer in the whole overlay network.
Chapter 4. Implementation

Figure 4.9: iNegotiate - negotiation task allocation operation
Figure 4.10: *iNegotiate* - negotiation operation
Figure 4.11: *iNegotiate* - mobile entity locating operation
4.3 Chapter Summary

This chapter presents the implementation details of \textit{iNegotiate}. It starts by describing the system architecture and highlights five core components including the interaction handler, the SLA manager, the template matchmaker, the trust evaluator, and the negotiator. The interaction handler implements the message listener and the message processing methods according to the interaction rules defined in Chapter 3.4.2. The SLA manager maintains the SLA repository and SLA template repository, which store the current executing SLAs and registered templates respectively. The template matchmaker implements the semantic similarity checking described in Chapter 3.4.1, which identifies the negotiation candidates that have the potential to satisfy the consumer’s request. The trust evaluator implements the trust model defined in Chapter 3.4.3, which prioritizes negotiation candidates according to historical information. The negotiator implements the offer evaluation model, decision-making model and negotiation tactics defined in Chapter 3.4.4, evaluating received offers and proposing counteroffers. Then the chapter describes \textit{iNegotiate}'s implementation using the \textit{Simonstrator} simulator. The class diagrams are presented to describe \textit{iNegotiate} messages and how \textit{iNegotiate}'s components use these messages to offer its functionalities. The main operations such as negotiation overlay creation, template distribution, negotiation task allocation and mobile entity locating are presented using sequence diagrams, which illustrate the behaviours of involved components after sending/receiving a message and how they cooperate with each other to accomplish a negotiation task.
Chapter 5

Evaluation

This chapter evaluates the performance of iNegotiate from four perspectives: template match-making using WIoT-SLA ontology, negotiation candidates selection, bilateral negotiation strategy, and the distributed negotiation using the hierarchical negotiation overlay network (HNON). This chapter is organised as follows: Section 5.1 introduces the evaluation approaches and specifies a set of evaluation objectives. Section 5.2 details the evaluation study on the WIoT-SLA template matchmaking algorithm. Section 5.3 details the evaluation study on iNegotiate’s trust-based candidates selection model, Section 5.4 details the evaluation study on iNegotiate’s bilateral negotiation strategy, and Section 5.5 details the evaluation study on iNegotiate’s distributed negotiation model.

5.1 Evaluation Approach

As shown in Table 5.1, the evaluation of iNegotiate is classified into four studies that address the research questions proposed in Chapter 1.4, which are:

- **Study 1: Evaluation of WIoT-SLA Template Match-making**
  
  This study measures to what extent can the use of SLA templates to describe negotiable IoT services accommodate the requirements of finding the negotiation candidates that have the potential to match a consumer’s request (RQ. 1). An SLA template describes a service’s functional and non-functional features, as well as the negotiation information
such as negotiable terms and possible constraints. With a uniform SLA ontology, gateways can identify negotiation candidates by matching the request with registered SLA templates. As described in Chapter 3.4.1, iNegotiate proposes the WIoT-SLA ontology to facilitate the template matchmaking process. In this study, the WIoT-SLA template matchmaking algorithm is implemented and deployed on different types of devices to test its performance, and precision, recall, accuracy and average processing time are measured. Details of this study are described in Section 5.2.

- **Study 2: Evaluation of Trust-based Candidates Selection**
  This study measures to what extent can the use of historical information to select negotiation candidates improve both negotiation efficiency and the consumer’s satisfaction level (RQ. 3). In an open market where many third-party service providers offering similar services with different prices and quality levels, selecting negotiation candidates based on historical information may help to reduce the probability of negotiating with candidates that cannot fully satisfy a consumer’s request or overstate their service qualities for more business profit. As described in Chapter 3.4.3, iNegotiate selects negotiation candidates based on a trust credit analyzed from historical data. In this study, the trust-based candidate selection model is implemented and deployed on different devices to test its performance. The behaviours of different service providers are simulated using a dataset collected by invoking real IoT services. The evaluation metrics include success rate, negotiation utility, SLA compliance, and average execution time. Details of the study are described in Section 5.3.

- **Study 3: Evaluation of Bilateral Negotiation Strategy**
  This study measures to what extent can a negotiation strategy balance the tradeoff between success rate and negotiation utility in a time-constrained negotiation scenario. During the bilateral negotiation process, each negotiation participant applies a negotiation strategy to adapt their expectations to reach a global beneficial agreement, but in the meanwhile, keeps the highest possible utility. As described in Chapter 3.4.4, iNegotiate proposes a deadline-aware negotiation strategy to evaluate received offers
and propose new offers. This study simulates a multi-bilateral negotiation scenario where gateways use the proposed strategy to negotiate with multiple candidate service providers one after another. The negotiation performance is evaluated by success rate, negotiation utility, and average execution time. Details of the study are described in Section 5.4.

- **Study 4: Evaluation of Distributed Negotiation using HNON**

  This study measures to what extent communication problems during an SLA negotiation process can be addressed in a large-scale environment where negotiating entities may be mobile and distributed in different locations (RQ. 2). As described in Chapter 3.4.2, iNegotiate proposes a distributed SLA negotiation model that uses HNON to address communication problems between negotiation participants in a dynamic environment. This model is controlled by a protocol regulating the messages types, message interaction rules, and the operations to process the messages. In this study, the proposed negotiation model is implemented in the Simonstrator platform to simulate negotiations in a smart city environment where a number of gateways are deployed to process the requests received from simulated mobile service providers and consumers. The system performance is assessed under a set of test cases using different simulation configurations. The evaluation metrics include response delivery rate, request delivery rate, success rate, and the number of messages sent. Details of the study are described in Section 5.5.
<table>
<thead>
<tr>
<th>Research Question</th>
<th>Proposed Study</th>
<th>Performance Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ. 1: Negotiation object</td>
<td>Study 1: Evaluation of WIoT-SLA template match-making (Section 5.2)</td>
<td>Precision, Recall, Accuracy, Recall, Precision</td>
</tr>
<tr>
<td>RQ. 2: Negotiation protocol</td>
<td>Study 4: Evaluation of Distributed Negotiation using HNON (Section 5.5)</td>
<td>Request Delivery Rate, Success Rate, Response Delivery Rate, Request Delivery Rate</td>
</tr>
<tr>
<td>RQ. 3: Negotiation candidates selection</td>
<td>Study 2: Evaluation of Trust-based Negotiation Candidates Selection (Section 5.3)</td>
<td>Success Rate, SLA Compliance, Negotiation Utility, Average Execution Time</td>
</tr>
<tr>
<td>RQ. 4: Negotiation strategy</td>
<td>Study 3: Evaluation of Bilateral Negotiation Strategy (Section 5.4)</td>
<td>Success Rate, Negotiation Utility, Average Execution Time</td>
</tr>
</tbody>
</table>

Table 6.1: Research Questions and Corresponding Evaluation Studies
5.2 Evaluation of WIoT-SLA Template Match-making

Since consumers have no prior knowledge about all available services in a dynamic large-scale environment, they are unlikely to provide negotiation candidates’ information (i.e., negotiation with unknown participants) when requesting an SLA-supported service. Capacity-aware SOA assumes providers describe functional or non-functional service properties in SLA templates, and register the templates to the negotiation system so that their offerings can be acknowledged and discovered when compatible requests are received by the system. iNegotiate proposes WIoT-SLA ontology to describe IoT services and their negotiable features. This section measures the performance of template matchmaking efficiency using WIoT-SLA.

5.2.1 Performance Metrics

The template match-making process should be able to identify the candidate templates that have the potential to satisfy the consumer’s requirements. This study uses four performance metrics to evaluate matchmaking efficiency:

- **Average processing time**: This metric measures the average time that the WIoT-SLA template matchmaking takes to identify candidate services providers, which reflects the computation complexity of the matchmaking mechanism.

  \[
  Precision = \frac{|satisfyingCandidates \cap identifiedCandidates|}{|identifiedCandidates|}
  \]  

- **Precision**: This metric measures the percentage of identified candidates that can satisfy a consumer’s request, which is defined as:

  \[
  Precision = \frac{|satisfyingCandidates \cap identifiedCandidates|}{|identifiedCandidates|}
  \]  

- **Recall**: This metric measures the percentage of satisfying candidates that can be identified by the template matchmaking algorithm:

  \[
  Recall = \frac{|satisfyingCandidates \cap identifiedCandidates|}{|satisfyingCandidates|}
  \]  

- **Accuracy**: This metric measures the matchmaking accuracy. If the true positives (i.e.,
satisfying templates that correctly identified) are denoted as $tp$, the false positives are denoted as $fp$ (i.e., the identified candidate template can not satisfy the consumer’s request), the true negatives are denoted as $tn$ (i.e., the unsatisfying templates that are not identified), the false negatives are denoted as $fn$ (i.e., the satisfying templates that are not identified), the accuracy is defined as:

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn} \quad (5.3)$$

5.2.2 Experimental Setup

This study assumes that a user requests a hazardous gas detection service with functional requirements including minimum sample interval, maximum data deviation, access credential and data reporting protocol, and QoS requirements including price, availability, reliability and latency. The service discovery engine detects a set of templates that matches the consumer’s business goals based on the semantic relations between service parameters [Chen and Clarke, 2014]. These candidate templates may differ in terms of service features, QoS guarantees and negotiation constraints. Using an SLA template prototype of a gas detection service structured using WIoT-SLA, three datasets are created.

The first two datasets are designed to measure the computation cost of the multi-phase template matchmaking mechanism. Each dataset contains 30, 60 and 100 JSON-formatted SLA templates, which are different in terms of spatial features, configuration items, QoS parameters and negotiation constraints. In the first dataset (i.e., test case 1), only 20% of the services match the request, while in the second dataset (i.e., test case 2), the percentage is increased to 90% (i.e., 10% conflict). For the services that violate the request, 50% of them conflict with the spatial requirements and the rest conflict with the functional or QoS requirements. This design aims to detect whether the multi-phase service filtering based on spatial features and service names before calculating the correspondence between the request and available SLA templates can help to reduce the processing time.

The third data set (i.e., test case 3) is designed to measure the matchmaking precision, recall and accuracy. In the third dataset, 60% of the services match the request. 60% of the
Chapter 5. Evaluation

satisfying templates present the same service properties using the same or synonymous words (e.g., using robustness to represent reliability), 40% of them present the same service properties using synonymous names but different data types. 30% of the conflicting templates miss the required features and 70% of them have no space for further negotiation (i.e., conflicting negotiation constraints). Table C.1 in Appendix C lists the synonymous words used in test case 3.

Table 5.2: Hardware Configurations

<table>
<thead>
<tr>
<th>Experiment Device</th>
<th>Configuration</th>
</tr>
</thead>
</table>
| Desktop           | Model: Dell-OptiPlex-990  
OS: Windows 10  
CPU: Intel Core i7-2600  
RAM: 4GB DDR3 1333MHz |
| Laptop            | Model: 13-inch MacBook-Pro, early 2011  
OS: macOS High Sierra  
CPU: Intel Core i5  
RAM: 8GB DDR3 1333MHz |
| Raspberry Pi      | Model: Raspberry Pi 3 B  
OS: Raspbian Jessie Release 8.0  
CPU: 4xCortex-A7  
RAM: 1 GB |

The WIoT-SLA match-making algorithm is implemented in Java under Eclipse Mars2 IDE, and the third-party library WS4J\(^1\) is integrated to calculate the semantic similarity based on an auxiliary source WordNet\(^2\). Three different semantic relatedness methods are employed when measuring the matchmaking efficiency: path-length based similarity (PATH), Lin similarity (LIN) and WuPalmer similarity (WUP) [Slimani, 2013, Jurafsky and Martin, 2014]. The threshold that detects whether two terms are semantically matched is increased from 0.01 to 0.9 to measure the corresponding precision, recall and accuracy. The executable

\(^1\)https://code.google.com/archive/p/ws4j/ - Accessed on 22 Jan 2019  
\(^2\)https://wordnet.princeton.edu/documentation - Accessed on 15 Jan 2020
jar file is deployed on three devices: a Dell-OptiPlex-990 desktop\textsuperscript{3}, a 13-inch MacBook\textsuperscript{4}, and a Raspberry Pi\textsuperscript{5}. The description of these devices are shown in Table 5.2.

As summarized in Chapter 2.5, to our best knowledge, there are no IoT SLA specifications that support SLA creation through multi-round bilateral negotiations. Without any baselines, this study only measures the performance of WIoT-SLA template matchmaking under different circumstances using the metrics described in Section 5.2.1.

5.2.3 Results

Figure 5.1: Precision, recall and accuracy of test case 3

Figure 5.1 presents the matchmaking efficiency under different thresholds using the third dataset. The matchmaking with LIN similarity shows a high precision when the threshold is lower than 0.5, but its lower recall indicates that some satisfying candidate templates were incorrectly filtered out. The matchmaking with PATH similarity is sensitive to the threshold values, and its precision is lower than WUP and LIN. The matchmaking with WUP similarity shows a better and more stable performance in a wider range. It achieves the highest precision, recall and accuracy when the threshold is set around 0.7. However, even under the best situation, there are still about 25% SLA templates incorrectly matched to the request, which introduces unnecessary computation costs to the following negotiation.

Chapter 5. Evaluation

5.2 Evaluation of APT

(a) APT of different service scale
(b) APT using different datasets

Figure 5.2: Average processing time

phase.

Figure 5.2 shows the average processing time (APT) on each device as the scale of candidate services increases, and the APT using the first two datasets respectively. Figure 5.2(a) shows that for all the devices, the average processing time increases as the number of candidate templates increases. The service scale has a larger negative impact on resource-constrained devices like a Raspberry Pi. However, this negative impact can be slightly reduced by adopting the multi-phase matching mechanism since the conflicted templates can be filtered based on location information and semantic similarities of service terms without further processing. This is shown in Figure 5.2(b) where the average processing time is shorter if most of the candidate templates do not satisfy the consumer’s request. Figure 5.2 also shows that the responsiveness is highly dependent on the gateways’ computational capabilities. For instance, the APT on the desktop computer (622ms approx.) is about 12 times of that on Raspberry Pi-3 (7438ms approx.) when there are 100 candidate templates. This indicates that for resource-constrained devices, a more light-weight SLA matchmaking mechanism is required.

5.3 Evaluation of Trust-based Candidates Selection Model

The trust-based candidates selection mechanism is designed to address the problem of when service providers claim a service capability that is far beyond the actual capability or they have
flexible negotiation constraints that are not specified in SLA templates. This section measures to what extent can the proposed trust-based candidates selection mechanism improve both negotiation efficiency and consumer’s satisfaction in a such complex environment (RQ.3).

5.3.1 Performance Metrics

The proposed trust-based selection model aims to identify an optimal service provider from a set of candidates, considering that there is limited negotiation time. This study assumes that consumers expect a fast responses and the pre-defined negotiation time only allows for a negotiation with a single service provider. It is therefore important for the negotiation gateway to rank the candidate service providers in the correct order. To evaluate whether the proposed trust-based ranking method helps to improve negotiation efficiency, the evaluation metrics used were:

- **Negotiation success rate**: This metric measures the percentage of successful negotiations, which is defined as:

  \[
  \text{SuccessRate} = \frac{\text{Number of Successful Negotiations}}{\text{Total Number of Negotiations}}
  \]  

  \(\text{(5.4)}\)

- **Negotiation utility**: This metric measures the level of satisfaction of a consumer’s requirements, which is defined as:

  \[
  \text{NegotiationUtility} = \sum_{p_i \in Q} w_i (p_i - \text{res}_{p_i}) \quad \text{prf}_{p_i} - \text{res}_{p_i}
  \]  

  \(\text{(5.5)}\)

  where \(p_i\) is the negotiated value of a negotiable term, \(w_i\) is the weight of the negotiation term representing the consumer’s negotiation preference, \(\text{prf}_{p_i}\) and \(\text{res}_{p_i}\) are the consumer’s preferred value and reserved value respectively.

- **SLA compliance**: This metric measures whether a service provider keeps its promises after SLA negotiation, and is defined as:

  \[
  \text{SLA_{compliance}} = 1 - \text{violation rate}
  \]  

  \(\text{(5.6)}\)
where $\text{vio}_{\text{rate}}$ is the SLA violation rate defined in Equation 3.24.

- **Average execution time**: This metric measures the overhead introduced by the candidate selection algorithm.

### 5.3.2 Baseline Approaches

This study compares five approaches that use different criteria to select a candidate service provider:

- **Random selection**: This approach randomly selects a candidate service provider to start a bilateral negotiation, which provides a baseline to evaluate the trust model’s performance.

- **Competence-based selection**: This approach calculates the competence value of each candidate service provider by analyzing historical negotiation data, and selects the provider that has the highest competence to start a bilateral negotiation.

- **Utility-based selection**: This approach calculates the utility value of each candidate service provider by analyzing recently monitored QoS data, and selects the provider that has the highest utility to start a bilateral negotiation.

- **Reputation-based selection**: This approach calculates the reputation value of each candidate service provider by analyzing the violation rate of previously executed SLAs, and selects the provider that has the highest reputation to start a bilateral negotiation.

- **Credit-based selection**: This approach calculates the credit value of each candidate service provider by analyzing its competence, utility and reputation, and selects the provider that has the highest credit to start a bilateral negotiation.

In this study, each approach uses the same experiment configuration parameters described in Section 5.3.3.
5.3.3 Experimental Setup

To study whether the proposed candidate selection model achieves the design objectives stated in Chapter 3.1, a set of service providers that differ in terms of trustworthiness, service quality levels and negotiation constraints need to be simulated. The simulation of different service providers is based on an IoT dataset collected from IoT services using different real sensors⁶. Ten types of sensors were connected to an Arduino board, and they provided their data through RESTful services. A Linux workstation simulated a number of virtual users using Apache JMeter version 3.3⁷ to increase the workload. Services were invoked according to a Poisson Timer with \( \lambda \) equals to 100 ms and a delay offset of 300 ms. A Raspberry Pi placed four hops away from the services acted as the actual user who invoked the services every 5 minutes for two months. The Raspberry Pi and Arduinos were connected to a router by WiFi, while the Linux Workstation used a wired connection.

Table 5.3: Dataset Statistics

<table>
<thead>
<tr>
<th>Service</th>
<th>Observations</th>
<th>( \mu_{rt} )</th>
<th>( \sigma_{rt} )</th>
<th>( \mu_{th} )</th>
<th>( \sigma_{th} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service 1</td>
<td>5898</td>
<td>2.49</td>
<td>61.995</td>
<td>43.69</td>
<td>792.290</td>
</tr>
<tr>
<td>Service 2</td>
<td>6824</td>
<td>1.96</td>
<td>46.19</td>
<td>169.37</td>
<td>17382.72</td>
</tr>
<tr>
<td>Service 3</td>
<td>6810</td>
<td>1.98</td>
<td>43.58</td>
<td>70.88</td>
<td>1546.13</td>
</tr>
<tr>
<td>Service 4</td>
<td>6431</td>
<td>2.73</td>
<td>65.95</td>
<td>145.98</td>
<td>17435.53</td>
</tr>
<tr>
<td>Service 5</td>
<td>5861</td>
<td>2.48</td>
<td>64.998</td>
<td>44.66</td>
<td>788.11</td>
</tr>
<tr>
<td>Service 6</td>
<td>7468</td>
<td>2.26</td>
<td>50.50</td>
<td>166.77</td>
<td>19206.53</td>
</tr>
<tr>
<td>Service 7</td>
<td>6988</td>
<td>2.598</td>
<td>68.88</td>
<td>81.36</td>
<td>3494.96</td>
</tr>
<tr>
<td>Service 8</td>
<td>6875</td>
<td>2.17</td>
<td>49.31</td>
<td>70.97</td>
<td>1605.53</td>
</tr>
<tr>
<td>Service 9</td>
<td>6427</td>
<td>2.81</td>
<td>69.78</td>
<td>44.94</td>
<td>914.60</td>
</tr>
<tr>
<td>Service 10</td>
<td>5843</td>
<td>2.54</td>
<td>60.58</td>
<td>43.85</td>
<td>801.27</td>
</tr>
</tbody>
</table>

Table 5.3 gives an overview of the descriptive statistics of the dataset, showing the number of observations, and the mean and variance of QoS metrics. Two types of QoS metrics were collected at each timestamp: response time (ms) and throughput (kbps). Response time is the duration between a user sending a request and receiving a response, while throughput denotes

---

⁶https://vladdie@bitbucket.org/vladdie/qos_dataset.git
⁷http://jmeter.apache.org - Accessed on November 27, 2019
the data transmission rate (i.e., the size of the SOAP message divided by the response time). Figure 5.3 shows the QoS trends of two example services captured within the observation period, which indicates the time-varying nature of service qualities. The unpredictability of QoS changes may be caused by workload, network congestion, unstable wireless connection or the malfunction of devices. Providers need to consider the QoS variability when negotiating SLAs to ensure they commit to the agreement with consumers, but in the meanwhile, make the best use of resources to maximize the profit.

Based on the dataset, ten types of service providers are created, which are different in terms of honesty, performance, and cost. Since the dataset is skewed, two guarantee terms

<table>
<thead>
<tr>
<th>Provider</th>
<th>Honesty</th>
<th>QoS</th>
<th>Sample Rate</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provider 1</td>
<td>high</td>
<td>service 1</td>
<td>[30, 60]</td>
<td>[10, 15]</td>
</tr>
<tr>
<td>Provider 2</td>
<td>high</td>
<td>service 2</td>
<td>[30, 60]</td>
<td>[18, 23]</td>
</tr>
<tr>
<td>Provider 3</td>
<td>high</td>
<td>service 3</td>
<td>[30, 60]</td>
<td>[20, 25]</td>
</tr>
<tr>
<td>Provider 4</td>
<td>low</td>
<td>[0.95, 0.99]</td>
<td>[30, 60]</td>
<td>[15, 20]</td>
</tr>
<tr>
<td>Provider 5</td>
<td>low</td>
<td>[0.95, 0.99]</td>
<td>[30, 60]</td>
<td>[15, 20]</td>
</tr>
<tr>
<td>Provider 6</td>
<td>high</td>
<td>service 6</td>
<td>[30, 60]</td>
<td>[18, 23]</td>
</tr>
<tr>
<td>Provider 7</td>
<td>low</td>
<td>[0.95, 0.99]</td>
<td>[30, 60]</td>
<td>[15, 20]</td>
</tr>
<tr>
<td>Provider 8</td>
<td>low</td>
<td>[0.95, 0.99]</td>
<td>[30, 60]</td>
<td>[20, 25]</td>
</tr>
<tr>
<td>Provider 9</td>
<td>high</td>
<td>service 9</td>
<td>[30, 60]</td>
<td>[10, 15]</td>
</tr>
<tr>
<td>Provider 10</td>
<td>high</td>
<td>service 10</td>
<td>[30, 60]</td>
<td>[10, 15]</td>
</tr>
</tbody>
</table>
were designed for SLA negotiation: responsiveness guarantee and throughput guarantee, which stand for the possibility that the corresponding QoS metric satisfies the pre-defined target (e.g., response time lower than 3 seconds and throughput higher than 50 kbps). Each service provider is associated with a set of negotiation constraints that represent the acceptable ranges of negotiable terms. The pre-defined negotiable terms are QoS guarantees, sample interval, and cost. Table 5.4 shows the configuration of negotiation constraints for each provider type. For honest service providers, the negotiation constraints of QoS guarantees were dynamically determined by recent service performance, since an exaggerated promise may increase the risk of breaking SLA guarantees. For dishonest service providers, the QoS negotiation constraints were set to \([0.95, 0.99]\), which means the provider expects to assure, with 95% possibility, that the QoS satisfies the target, but the assurance can increase to 99% through negotiation. Based on the configuration of service providers, 200 requests are generated at different timestamps in the dataset. The negotiation constraints of a consumer were intersected with the constraints of at least 60% of the service providers. This enabled us to evaluate the trust model in terms of assessing a provider’s competence. During the bilateral negotiation stage, the service provider used a time-dependent linear negotiation tactic [Faratin et al., 1998] while the negotiation gateway used the context-based negotiation tactic described in Section 3.4.4. The desirability of a consumer reaching an agreement was set to 0.5. In this study, the persistence method is used to forecast QoS values, which uses the last observed time-series value \(y_i\) as its next predicted value \(y_{i+1}\).

To evaluate the overhead introduced by the candidate selection algorithm on resource-constrained devices, the trust model is deployed on a laptop computer\(^8\), a Raspberry Pi\(^9\), and an Android mobile phone\(^{10}\). The description of these devices is shown in Table 5.5.

### 5.3.4 Statistical Analysis

To analyze the relationship between each ranking criteria (i.e., competence, reputation, utility, and credit) and the negotiation performance, for a request at a specific timestamp, the

---

\(^8\)https://www.asus.com/Laptops/X751LA/ - Accessed on 28 Apr 2020  
\(^{10}\)https://consumer.huawei.com/nz/support/phones/honor-6/ - Accessed on 28 Apr 2020
Table 5.5: Hardware Configurations

<table>
<thead>
<tr>
<th>Experiment Device</th>
<th>Configuration</th>
</tr>
</thead>
</table>
| Laptop            | Model: Asus X751L  
                              OS: Windows 10  
                              CPU: 1.8 GHz, 2.40 GHz  
                              RAM: 8 GB               |
| Raspberry Pi      | Model: Raspberry Pi 3 B  
                              OS: Raspbian Jessie Release 8.0  
                              CPU: 1.2 GHz  
                              RAM: 1 GB               |
| Android Phone     | Model: HUAWEI Honor 6 H60-L01  
                              OS: Linux Ubuntu image on Android 4.4.2  
                              Processor: Hisilicon Kirin 920  
                              RAM: 3 GB               |

ranking result based on the actual negotiation performance is analyzed against the ranking result based on the different ranking criteria proposed in the trust model. This statistical analysis is based on two ranking metrics called Spearman’s Rank Correlation [Myers et al., 2013] and Kendall’s Tau-b coefficient [Kendall, 1948]. Spearman’s rank correlation can be understood as a rank-based version of Pearson’s correlation coefficient, which can be used for variables that are not normal-distributed and have a non-linear relationship [Myers et al., 2013]. It is defined as:

\[ r_s = \frac{\text{cov}(rg_x, rg_y)}{\sigma_{rg_x} \sigma_{rg_y}} \quad (5.7) \]

where \( \text{cov}(rg_x, rg_y) \) is the covariance of the rank variables, and \( \sigma_{rg_x} \) and \( \sigma_{rg_y} \) are the standard deviations of the rank variables. If the rank variables are distinct integers, it can be simply computed using Equation 5.8.

\[ r_s = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \quad (5.8) \]

where \( d \) is the pairwise distances of the rank variables \( rg_{x1} \) and \( rg_{y1} \), \( n \) is the number of samples.

Kendall’s tau-b calculates the dependence between ranked variables by measuring the degree of monotonic relationships between them, even if they are non-normally distributed [Kendall, 1948]. Any pair of observations from rank list \( rg_X \) and \( rg_Y \) are said to be concordant
if \( rg_{x_i} < rg_{x_j} \) and \( rg_{y_i} < rg_{y_j} \), or if both \( rg_{x_i} > rg_{x_j} \) and \( rg_{y_i} > rg_{y_j} \). The pair is said to be discordant, if \( rg_{x_i} > rg_{x_j} \) and \( rg_{y_i} < rg_{y_j} \), or \( rg_{x_i} < rg_{x_j} \) and \( rg_{y_i} < rg_{y_j} \). If \( rg_{x_i} = rg_{x_j} \), or \( rg_{y_i} = rg_{y_j} \), the pair is a tie. Kendall’s Tau-b is defined as:

\[
\tau = \frac{c - d}{\sqrt{C_n^2 - T} \sqrt{C_n^2 - U}} \\
T = \sum_t t(t - 1)/2 \\
U = \sum_u u(u - 1)/2
\]

where \( c \) is the number of concordant pairs, \( d \) is the number of discordant pairs, \( t \) is the number of observations of rank variables \( rg_x \) that are tied, and \( u \) is the number of observations of rank variable \( rg_y \) that are tied.

5.3.5 Result

![Evaluation of Service Providers](image)

Figure 5.4: Evaluation of service providers
Figure 5.4 shows the results for each provider using our trust model, where $\lambda$ in Equation 3.5 is set to 0.5. The variation of the values is caused by the change of service performance and the adjustment of providers’ negotiation constraints at different timestamps. Although the negotiation constraints of dishonest providers (4, 5, 7 and 8) make them the most likely to satisfy the consumers’ requests (Table 5.4), their credit values are bottom-ranked due to their low reputation value. These low credit values decrease their chances of being selected, which further have a negative impact on their competence in the future as past negotiation results fade over time (Equation 3.6).

Table 5.6 shows the negotiation performance using different provider selection approaches. The results show that the negotiation success rate of the credit-based ranking increases about 13.5% and the SLA compliance increases about 34.5% compared to random ranking, but with a tradeoff on negotiation utility. Credit-based ranking can balance the short-term benefit (i.e., success rate and negotiation utility) and long-term benefit (i.e., SLA compliance after negotiation) compared to random ranking. Among the factors that compose the credit value, competence shows a good performance compared to the other two factors. This is because in this study, 60% of the service providers are assumed to be honest, which means most of the service providers set their negotiation preferences based on true performance. This helps to avoid SLA violations after making an agreement. Ranking by competence is very effective if a service consumer is eager to make an agreement with a service provider. However, if most of the service providers are dishonest in the environment, only considering competence increases the risk of SLA violation. Ranking by reputation significantly increases negotiation failure, since it always chooses the providers who are offering stable, high-quality services. In reality, these services are more likely to have smaller and more stringent negotiation constraints. Negotiation failure happens when there is no intersected negotiation space between a provider and a consumer. Compared to reputation-based ranking, utility-based ranking selects service providers only based on the current service performance. This reduces the possibility of choosing a high-quality service provider who does not have any negotiation space with the consumer. Therefore, the utility-based ranking has a higher negotiation success rate compared to reputation-based ranking, which can better adapt to dynamic changes in the environment.
### Table 5.6: Performance Overview

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Success Rate</th>
<th>Negotiation Utility</th>
<th>SLA Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>competence (Eq.3.8)</td>
<td>0.94</td>
<td>0.6347±0.00258</td>
<td>0.90732</td>
</tr>
<tr>
<td>Utility (Eq.3.17)</td>
<td>0.7</td>
<td>0.60864 ± 0.00971</td>
<td>0.65605</td>
</tr>
<tr>
<td>Reputation (Eq.3.23)</td>
<td>0</td>
<td>N/A</td>
<td>0.76471</td>
</tr>
<tr>
<td>Credit (Eq.3.5)</td>
<td>0.885</td>
<td>0.4197 ± 0.00474</td>
<td><strong>0.97934</strong></td>
</tr>
<tr>
<td>Random</td>
<td>0.72832</td>
<td>0.56455 ± 0.00956</td>
<td>0.644654088</td>
</tr>
</tbody>
</table>

To analyze the contribution of each ranking criterion in making correct decisions, for a specific negotiation request, a provider’s current competence value, reputation value, utility value, and credit value are collected. Then, the multi-bilateral negotiations are performed to collect each provider’s negotiation performance. When the negotiated SLA is due, the providers are ranked according to different evaluation metrics and criteria. Table 5.7 and Table 5.8 show the Spearman’s Rank Correlation and Kendall’s Tau-b coefficient of the ranking results. The number in parentheses is the p-value. A p-value less than 0.05 indicates strong evidence that the two ranking variables are significantly correlated, while a p-value higher than 0.05 is not statistically significant. The result of the significance test is consistent with the results in Table 5.6. Both of the two ranking metrics show that competence is positively correlated with negotiation success rate, reputation is positively correlated with SLA compliance, and reputation is negatively correlated with the negotiation success rate. Spearman Rank Correlation also shows that credit is positively related to SLA compliance, which is consistent with our previous result that SLA compliance is improved about 34.5% compared to the random ranking. However, the contribution of utility is unclear using Kendall’s Tau coefficient, while Spearman Rank Correlation suggests the utility is negatively correlated with the success rate. This indicates that for long-term contract negotiation, considering a service’s instant stability is inefficient.

Table 5.9 shows the average execution time (ms) of calculating reputation, utility and competence on different devices. The calculation of utility introduces much more overhead.
Chapter 5. Evaluation

Table 5.7: Spearman’s Rank Correlation

<table>
<thead>
<tr>
<th>Criteria</th>
<th>SLA Compliance</th>
<th>Negotiation Utility</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>competence</td>
<td>0.27471</td>
<td>0.62614</td>
<td>0.79772</td>
</tr>
<tr>
<td></td>
<td>(0.4424)</td>
<td>(0.05278)</td>
<td>(0.005692)</td>
</tr>
<tr>
<td>utility</td>
<td>-2.01008</td>
<td>-0.20101</td>
<td>-0.64578</td>
</tr>
<tr>
<td></td>
<td>(0.5776)</td>
<td>(0.5776)</td>
<td>(0.0437)</td>
</tr>
<tr>
<td>reputation</td>
<td>0.77053</td>
<td>-0.680852</td>
<td>0.34188</td>
</tr>
<tr>
<td></td>
<td>(0.009099)</td>
<td>(0.03021)</td>
<td>(0.3336)</td>
</tr>
<tr>
<td>credit</td>
<td>0.65662</td>
<td>-0.16413</td>
<td>0.49383</td>
</tr>
<tr>
<td></td>
<td>(0.03917)</td>
<td>(0.6505)</td>
<td>(0.1469)</td>
</tr>
</tbody>
</table>

Table 5.8: Kendall’s Tau-b coefficient

<table>
<thead>
<tr>
<th>Criteria</th>
<th>SLA Compliance</th>
<th>Negotiation Utility</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>competence</td>
<td>0.24914</td>
<td>0.40452</td>
<td>0.68313</td>
</tr>
<tr>
<td></td>
<td>(0.3673)</td>
<td>(0.106)</td>
<td>(0.0167)</td>
</tr>
<tr>
<td>utility</td>
<td>-0.19377</td>
<td>-0.22473</td>
<td>-0.55301</td>
</tr>
<tr>
<td></td>
<td>(0.4832)</td>
<td>(0.3692)</td>
<td>(0.05271)</td>
</tr>
<tr>
<td>reputation</td>
<td>0.58132</td>
<td>-0.53936</td>
<td>0.29277</td>
</tr>
<tr>
<td></td>
<td>(0.03542)</td>
<td>(0.03114)</td>
<td>(0.3051)</td>
</tr>
<tr>
<td>credit</td>
<td>0.52595</td>
<td>-0.13484</td>
<td>0.42289</td>
</tr>
<tr>
<td></td>
<td>(0.05702)</td>
<td>(0.59)</td>
<td>(0.1385)</td>
</tr>
</tbody>
</table>

Table 5.9: Average Execution Time

<table>
<thead>
<tr>
<th>Device</th>
<th>Time_{rep}</th>
<th>Time_{util}</th>
<th>Time_{cmp}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop</td>
<td>0.002 ms</td>
<td>10.118 ms</td>
<td>1.6945 ms</td>
</tr>
<tr>
<td>Android Phone</td>
<td>0.0225 ms</td>
<td>100.141 ms</td>
<td>1.3795 ms</td>
</tr>
<tr>
<td>Raspberry PI</td>
<td>0.0245 ms</td>
<td>326.5375 ms</td>
<td>3.123 ms</td>
</tr>
</tbody>
</table>

than the other two criteria while its contribution is very limited. This indicates that for resource-constrained devices, evaluating a service’s temporal performance is too heavyweight
Figure 5.5: Average Execution Time in Milliseconds

and unnecessary. Figure 5.5 shows the average execution time of credit calculation as the simulation continues. For all the devices, more overhead is introduced when the number of received requests increases. This may be caused by the increasing number of monitoring instances, especially for long-term contracts. Figure 5.6 show the CPU and memory usage of the Android phone and Raspberry Pi during the simulation. The memory usage increases for both devices as time passes, which is consistent with the increasing execution time. This implies that for resource-constrained IoT gateways, more investigations are required to improve the efficiency of the SLA monitoring mechanism.
5.4 Evaluation of \textit{iNegotiate}'s Bilateral Negotiation Strategy

Bilateral negotiation with incomplete information needs to balance the tradeoff between negotiation utility and success rate. A larger concession can make the offer more likely to be accepted by the service provider but reduces the consumer’s satisfaction level. \textit{iNegotiate} proposes a deadline-aware negotiation strategy, which uses a context-based tactic (UMI) or an artificial bee colony algorithm-based tactic (ABC) to generate a new proposal in each round. This section measures the performance of the proposed negotiation strategy by simulating a multiple bilateral negotiation scenario.

5.4.1 Performance Metrics

The negotiation strategy aims to reach an agreement when the service provider and consumer have intersected negotiation spaces, while keeping the negotiation utility as high as possible. To evaluate whether the negotiation tactics defined in the strategy model have a good balance between the success rate and negotiation utility, the evaluation metrics used were:

- **Negotiation Utility**: This metric measures the level of satisfaction against the consumer’s requirements after bilateral negotiations, which is calculated by Equation 5.5.

- **Success Rate**: This metric measures the ratio of successful bilateral negotiations, which is calculated by Equation 5.4.
• **Negotiation Round**: This metric measures the number of rounds the bilateral negotiation session takes to reach an agreement.

• **Average Execution Time**: This metric measures the overhead introduced by the negotiation tactic.

### 5.4.2 Baseline Approaches

This study compares the proposed context-based negotiation strategy and ABC-based negotiation strategy with four baseline approaches that have shown good performance in terms of utility and successful deals in existing literature [Zheng et al., 2014, Faratin et al., 1998]:

- **Game theory-based mixed approach (UMC)**: This baseline plays concession or tradeoff tactics with a certain probability in each round. It was implemented by following the algorithm proposed by Zheng et al. [Zheng et al., 2014]. In this study, the probability of playing tradeoff tactics is set to 0.5.

- **Behaviour-dependent relative tit for tat approach (BDR)**: This baseline makes concessions based on the previous offers received from the negotiation opponent. It was implemented by following the algorithm proposed by Faratin et al. [Faratin et al., 1998]. Since the condition of applying this approach is \( r > 2 \) where \( r \) is the negotiation rounds, this strategy did not make any concessions in the first two rounds in this study.

- **Time-dependent linear approach (TDL)**: This baseline simply makes linear concessions depending on the remaining negotiation time. It was implemented by following the algorithm proposed by Faratin et al. [Faratin et al., 1998]. In this study, the remaining negotiation time is estimated by the remaining negotiation rounds.

- **Resource-dependent patient approach (RDP)**: This baseline makes concessions according to the quantity of available resources. It was implemented by following the algorithm proposed by Faratin et al. [Faratin et al., 1998]. In this study, the resources are modeled as the number of negotiation candidates. The algorithm’s parameter \( \mu \) is set to 7.
• **Context-based mixed approach (UMI):** This approach makes concessions and tradeoffs according to context information including time, available resources, and the consumer’s negotiation preference. It includes normalizing negotiable terms’ values and modifying them using the context-based utility function defined in Chapter 3.4.4.

• **ABC-based mixed approach (ABC):** This approach uses a modified artificial bee colony optimization algorithm to generate counter offers when the consumer’s negotiation preference is not specified. As described in Chapter 3.4.4, it simulates a set of bees to search for solutions that are more likely to be accepted by both parties according to the fitness values of discovered solutions.

In this study, each approach uses the same experiment configuration parameters described in Section 5.4.3.

### 5.4.3 Experimental Setup

To measure the performance of the proposed negotiation strategy, this study simulates a multi-bilateral negotiation scenario where a gateway negotiates with different candidate service providers and makes an agreement with the provider that offers the highest utility. In the study, a consumer requests a hazardous gas monitoring service at a random location within a rectangular area where latitude varies from 53.33385 to 53.35556, and longitude varies from −6.27963 to −6.23328. The negotiation request is generated by randomly assigning values to SLA terms according to the predefined variation range listed in Table 5.10.

Based on the request, a set of candidate service providers that have the potential to satisfy the consumer’s requirements are simulated according to the configuration policies outlined in Table 5.11. These service providers are classified into three groups based on the service level they can provide: high-performance (HP) services, moderate-performance (MP) services, and low-performance (LP) services. The intersection degree of negotiation space between negotiating parties is set to 0.7, 0.4 and 0.2 for HP, MP and LP service providers respectively. For example, if the negotiation space of accuracy is set to [0.84, 0.97] in the request, the corresponding negotiation spaces for HP, MP and LP providers are set to
### Table 5.10: Configuration of Negotiation Constraints for Each Request

<table>
<thead>
<tr>
<th>SLA Term</th>
<th>Requirement</th>
<th>Variation Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Price ($)</strong></td>
<td>Reserved value</td>
<td>[8, 13]</td>
</tr>
<tr>
<td></td>
<td>Preferred value</td>
<td>[20, 25]</td>
</tr>
<tr>
<td><strong>Sample rate</strong></td>
<td>Reserved value</td>
<td>[30, 40]</td>
</tr>
<tr>
<td></td>
<td>Preferred value</td>
<td>[60, 80]</td>
</tr>
<tr>
<td><strong>Spatial requirement</strong></td>
<td>Service location</td>
<td>latitude: [53.33385, 53.35556]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>longitude: [-6.27963, -6.23328]</td>
</tr>
<tr>
<td></td>
<td>Acceptable distance</td>
<td>1.1 km</td>
</tr>
<tr>
<td><strong>Availability (%)</strong></td>
<td>Reserved value</td>
<td>[0.75, 0.85]</td>
</tr>
<tr>
<td></td>
<td>preferred value</td>
<td>[0.9, 0.99]</td>
</tr>
<tr>
<td><strong>Response time (ms)</strong></td>
<td>Reserved value</td>
<td>[60, 80]</td>
</tr>
<tr>
<td></td>
<td>Preferred value</td>
<td>[130, 150]</td>
</tr>
<tr>
<td><strong>Accuracy (%)</strong></td>
<td>Reserved value</td>
<td>[0.75, 0.85]</td>
</tr>
<tr>
<td></td>
<td>Preferred value</td>
<td>[0.9, 0.99]</td>
</tr>
</tbody>
</table>

* In this experiment, sample rate refers to the number of samples per minute.

[0.85, 0.98], [0.79, 0.92], and [0.74, 0.87] respectively. Also, mobile service providers and static service providers are different in their price models and services’ spatial features. The price of a mobile service linearly depends on the standard Euclidean distance between the consumer’s requirements and provider’s offerings; while the price of a static service is restricted by a range if the price is negotiable (PIN), or a static value if it is non-negotiable (PNN). The spatial feature of a mobile service is flexible, and it can be dynamically configured according to the consumer’s demand. The probability of a mobile service provider satisfying a consumer’s spatial requirement is set to 90%, 50%, 20% for HP, MP and LP providers respectively. If the requested service location is not acceptable, the mobile service provider offers a random location within 1km around the requested location. Compared to mobile services, the spatial feature of a static service has a much smaller negotiation space. HP static service providers have six service instances uniformly distributed in the area, while MP and LP providers have four and two service instances respectively. This study assumes two different negotiation environments: resource-limited and resource-rich. In the resource-limited negotiation environment, one mobile provider (MP) and one static provider (MP-PNN) are selected as negotiation candidates. In the resource-rich negotiation environment,
Table 5.11: Configuration of Service Providers Based on Requests

<table>
<thead>
<tr>
<th>Provider Type</th>
<th>Intersection Degree</th>
<th>Service Location</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP</td>
<td>0.7</td>
<td>6 locations</td>
<td></td>
</tr>
<tr>
<td>MP</td>
<td>0.4</td>
<td>4 locations</td>
<td></td>
</tr>
<tr>
<td>LP</td>
<td>0.2</td>
<td>2 locations</td>
<td></td>
</tr>
<tr>
<td>Mobile</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HP</td>
<td>0.7</td>
<td>90% acceptance</td>
<td></td>
</tr>
<tr>
<td>MP</td>
<td>0.4</td>
<td>50% acceptance</td>
<td></td>
</tr>
<tr>
<td>LP</td>
<td>0.2</td>
<td>20% acceptance</td>
<td></td>
</tr>
</tbody>
</table>

* $k$ and $b$ are the coefficients calculated from price’s variation range.

Six mobile providers (LP, MP, and HP) and six static providers (LP-PIN, LP-PNN, MP-PIN, MP-PNN, HP-PIN, HP-PNN) are selected as negotiation candidates.

Two experiments are performed to measure the performance of the UMI tactic and ABC tactic under different negotiation environments respectively. Table 5.12 outlines the simulation parameters of the first experiment. $\lambda$ is the concession rate that is used to update negotiation utility in each round (Figure 3.25). The consumer’s negotiation desirability ($DF$) and negotiation preference (i.e., the weight values of negotiable terms) are specified in the request. In this experiment, the variable parameters are the negotiation environment, the maximum negotiation rounds, and the consumer’s negotiation desirability. A set of test cases are designed to compare UMI’s performance with other baseline approaches under the impact of different parameter combinations.

Table 5.12: Simulation Parameters of Experiment 1: Evaluation of UMI-based Strategy

<table>
<thead>
<tr>
<th>Negotiation environment</th>
<th>resource-rich, resource-limited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum negotiation rounds</td>
<td>10, 20</td>
</tr>
<tr>
<td>Concession rate ($\lambda$)</td>
<td>0.06</td>
</tr>
<tr>
<td>Consumer’s desirability ($DF$)</td>
<td>0.1, 0.5, 0.9</td>
</tr>
<tr>
<td>Consumer’s negotiation preference</td>
<td>availability: 0.2  response time: 0.2  accuracy: 0.3  sample rate: 0.1  price: 0.2</td>
</tr>
<tr>
<td>Test cases</td>
<td>Environment X Rounds X Desirability</td>
</tr>
</tbody>
</table>
Table 5.13 outlines the simulation parameters of the second experiment. $C_0$, $C_1$, $mt$, and $\beta$ are the algorithm parameters used in Equation 3.39 and Equation 3.37. In this experiment, the variable parameters are the negotiation environment and the maximum negotiation rounds. A set of test cases are designed to compare ABC’s performance with other baseline approaches under the impact of different parameter combinations.

Table 5.13: Simulation Parameters of Experiment 2: Evaluation of ABC-Based Strategy

<table>
<thead>
<tr>
<th>Negotiation environment</th>
<th>resource-rich, resource-limited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum negotiation rounds</td>
<td>10, 20</td>
</tr>
<tr>
<td>Algorithm parameters</td>
<td></td>
</tr>
<tr>
<td>$C_0 = 0$ (Equation 3.39)</td>
<td></td>
</tr>
<tr>
<td>$C_1 = 0.9$ (Equation 3.39)</td>
<td></td>
</tr>
<tr>
<td>$\beta = 1$ (Equation 3.39)</td>
<td></td>
</tr>
<tr>
<td>$mt = 5$ (Equation 3.37)</td>
<td></td>
</tr>
<tr>
<td>Test cases</td>
<td>Environment X Rounds</td>
</tr>
</tbody>
</table>

The simulations are implemented with Java using Eclipse Mars IDE. To evaluate the overhead introduced by different negotiation tactics, all the test cases are performed on a laptop, a Raspberry Pi, and an Android phone. The description of these devices is shown in Table 5.5. Each test case was repeated 100 times to reduce the chance variation.

5.4.4 Statistical Analysis

A statistical analysis was performed to verify whether the observed results of the evaluated approaches are significantly different. Kruskal-Wallis test [Kruskal and Wallis, 1952] extends the Mann–Whitney U test, which works as a non-parametric alternative to the One Way ANOVA. It assesses the differences between three or more independently-sampled groups on a single, non-normally distributed continuous variable. The null hypothesis specifies that these groups are subsets of the same population. By combining the groups into a single group and ranking the data from 1 to $N$ ($N$ is the total number of observations across all groups), the test statistic $H$ can be calculated as follows:

$$H = (N - 1) \frac{\sum_{i=1}^{g} n_i (\bar{r}_i - \bar{r})^2}{\sum_{i=1}^{g} \sum_{j=1}^{n_i} (r_{ij} - \bar{r})^2}$$  \hspace{1cm} (5.9)
where \( n_i \) is the number of observations in group \( i \), \( \bar{r}_i \) is the average rank of all observations in group \( i \), \( r_{ij} \) is the rank of observation \( j \) from group \( i \), \( \bar{r} \) is the average of all the \( r_{ij} \). If there is no ties in the data, the test statistic \( H \) can be calculated as follows:

\[
H = \frac{12}{N(N+1)} \sum_{i=1}^{g} n_i \bar{r}_i^2 - 3(N + 1) \quad (5.10)
\]

This study applies the Kruskal-Wallis test on the analysis of success rate, negotiation utility, and negotiation rounds. MATLAB\(^{11} \) is used to calculate the \( p\)-value. The \( p\)-value smaller than 0.05 (i.e., the confidence level is set to 95\%) indicates the null hypothesis is rejected.

### 5.4.5 Result of Experiment 1: Evaluation of UMI-based Strategy

Figure 5.7 shows the negotiation results in the resource-limited environment. UMI-0.1, UMI-0.5, UMI-0.9 represent the UMI algorithm with 0.1, 0.5 and 0.9 desirability respectively. Compared to other baseline approaches, UMI demonstrates good performance in balancing success rate and negotiation utility, especially when the bilateral negotiation only allows a small number of interactions. TDL shows a stable performance despite the limitation on negotiation deadlines, but its negotiation utility is relatively low for both cases. UMC achieves the highest utility, which is about 55\% higher than UMI in short term negotiations (i.e., the maximum negotiation rounds equals to 10), but its success rate is 44\% lower than UMI. However, when more interactions are allowed in the negotiation process, the utility of UMC diminishes while the utility of UMI has a small increment \( (DF = 0.1) \) or maintains the same \( (DF = 0.5 \) or 0.9), which outperforms other approaches. Figure 5.7 also shows that negotiation rounds is proportional to negotiation utility. This is because negotiation participants are more conservative in making concessions when more interactions are allowed, trying to keep the highest possible utility.

Figure 5.8 shows the utility-changing trend during the bilateral negotiation process in the resource-limited environment. Each graph shows the mean utility observed in each round.

\(^{11}\)https://www.mathworks.com/products/matlab.html
with the error band representing the standard errors. UMI-0.5 in Figure 5.8a shows a similar changing pattern as UMC except that it makes a big concession in the last negotiation round. This is because UMI applies a deadline-aware decision-making model that generates a solicited offer as the ultimatum using reserved values for conflicting terms when the deadline
Chapter 5. Evaluation

(a) Utility-changing trend of different approaches (maxRound=10)

(b) Utility-changing trend of different approaches (maxRound=20)

Figure 5.8: Utility-changing trend in the resource-limited environment approaches. The difference between UMI-0.1 and UMI-0.9 indicates that UMI makes more concessions in early rounds if the consumer is eager to reach an agreement in a short time (i.e., large desirability).

Figure 5.9 shows the negotiation results in the resource-rich environment. Compared to other baseline approaches, UMI demonstrates good performance in both success rate and negotiation utility despite the limitation on the negotiation deadline. Similar to the previous result, TDL has a high success rate (90% approx.) but low negotiation utility (0.4 approx.) for both cases. UMC only has 46.8% success rate when the maximum number of negotiation rounds is 10, but the utility of successful negotiations is the highest (0.61 approx.). For long-term negotiations in the resource-rich environment, UMI outperforms other approaches. It maintains a high success rate (90% approx.) and the utility is about 46% higher than other approaches.

Figure 5.10 shows the utility-changing trend during the bilateral negotiation process in the resource-limited environment. The curves of UMI-0.5 and UMI-0.9 are quite different from the curves in Figure 5.8. This is because, in the resource-rich environment, UMI is more likely to play tradeoffs rather than concessions. UMI with higher desirability makes large concessions in the early rounds trying to reach an agreement as soon as possible. The concession degree gradually reduces as the negotiation proceeds, which avoids losing too
Figure 5.9: Negotiation performance in the resource-rich environment

much utility. UMI-0.1 has a more smooth curve because its concession degree is small in the early rounds and gradually increases as the negotiation proceeds. Figure 5.8 and Figure 5.10 demonstrate that the strategies using multiple combinations of tactics (i.e., UMC and UMI) outperform the strategies using only a single tactic (i.e., BDR, TDL, RDP). For example,
Chapter 5. Evaluation

(a) Utility-changing trend of different approaches (maxRound=10)

(b) Utility-changing trend of different approaches (maxRound=20)

Figure 5.10: Utility-changing trend in the resource-rich environment

TDL only makes concessions depending on the remaining negotiation rounds, which means it cannot adapt to different negotiation contexts. The regular monotonic utility-changing trend may be used by the negotiation opponent to predict its negotiation preference, which increases the probability of accepting an offer with lower utility. UMC and UMI combine concession and tradeoff tactics to increase the chance of finding a solution with higher utility. The random factor introduced during the process makes its utility curve more irregular than other approaches. UMI additionally models context information in the utility function to dynamically adjust concessions and tradeoffs according to different negotiation requirements, which means it has better performance in finding a balance between negotiation utility and success rate under different situations.

The Kruskal-Wallis test was performed to identify whether the performances of evaluated approaches are statistically significantly different. Figure 5.11 and Figure 5.12 present the comparison of ranked means, \( p-values \) calculated from Kruskal-Wallis test, and whether or not the null hypothesis \( H_0 \) is rejected. The vertical lines projected from comparison intervals (i.e., the horizontal lines with markers) are used to identify whether these approaches are statistically significantly different. Two approaches are not significantly different if their projected lines intersect with each other. In this experiment, \( H_0 \) is rejected in all the test cases except for the success rate of long term negotiations in resource-limited environment.
Figure 5.11: Multiple comparison of ranked means (resource-limited environment)
Chapter 5. Evaluation

(a) Ranked means of utility (maxRound=10)

(b) Ranked means of utility (maxRound=20)

(c) Ranked means of success rate (maxRound=10)

(d) Ranked means of success rate (maxRound=20)

(e) Ranked means of negotiation rounds (maxRound=10)

(f) Ranked means of negotiation rounds (maxRound=20)

Figure 5.12: Multiple comparison of ranked means (resource-rich environment)
The result of the statistical analysis is consistent with the previous discussions. For short term negotiations in the resource-limited environment, UMI-0.1, UMI-0.5, UMI-0.9 and TDL have statistically significantly better performance on success rate than RDP, UMC and BDR. For long term negotiations (i.e., where the number of negotiation rounds is large) in the resource-limited environment, except for maintain a high success rate, UMI-0.1’s negotiation utility is statistically significantly better than UMI-0.5, UMI-0.9, UMC, TDL and RDP. For short term negotiations in the resource-rich environment, UMI-0.1, UMI-0.5, UMI-0.9 and UMC have statistically significantly better performance on negotiation utility than BDR, TDL and RDP. UMI-0.1, UMI-0.5, UMI-0.9 and TDL have statistically significantly better performance on success rate than RDP, UMC and BDR. For long term negotiations in the resource-rich environment, except for maintaining a high success rate, UMI-0.1’s negotiation utility is statistically significantly better than other approaches. Although UMI-0.5 and UMI-0.9 have a lower negotiation utility compared to UMI-0.1, their utilities are still statistically significantly higher than UMC, BDR, TDL and RDP. For both environments, UMI requires more negotiation rounds to achieve the higher utility.

To evaluate the overhead introduced by different approaches, the execution time is measured on three devices that may be used as negotiation gateways. Figure 5.13 shows the average execution time of evaluation approaches on different devices. The five approaches have a similar responsiveness under different scenarios. The device’s storage and computation capabilities have a large effect on the strategy’s responsiveness. UMI has better performance on a Raspberry PI and on a laptop computer, but introduce slightly more overhead on the Andriod phone. For all the test cases, the execution time increases as more candidate service providers present in the environment. This indicates that bilateral negotiation is a computationally expensive process, especially on resource-constrained devices. The increment of candidate service providers introduced about 8 times the overhead on a Raspberry Pi for long term negotiations, while on the laptop computer, only about 5 times overhead increment is introduced.
Chapter 5. Evaluation

(a) Average execution time in resource-limited environment (maxRound=10)

(b) Average execution time in resource-rich environment (maxRound=10)

(c) Average execution time in resource-limited environment (maxRound=20)

(d) Average execution time in resource-rich environment (maxRound=20)

Figure 5.13: Average execution time on different devices
5.4.6 Result of Experiment 2: Evaluation of ABC-based Strategy

Figure 5.14: Negotiation performance in the resource-limited environment

Figure 5.14 shows the negotiation results using the ABC-based strategy in the resource-limited environment. For short-term negotiations, ABC gets a moderate utility, which is lower than the utility of UMC and BDR, but it maintains a much higher success rate than these two approaches. Figure 5.15a shows the utility-changing trend of optimal solutions under this situation. ABC concedes more in the early negotiation rounds but becomes more conservative as the negotiation proceeds. Once the negotiation deadline is approaching, it becomes more
inclined to concede again, trying to reach an agreement with the service provider. Since ABC is not aware of the consumer’s negotiation preference, it controls the balance between success rate and negotiation utility by dynamically changing fitness values and restricting the number of search iterations. As shown in Equation 3.38 and Equation 3.39, when $C_0 = 0, C_1 = 0.9$, the fitness value in the early negotiation rounds mainly depends on the solution’s utility. Equation 3.37 shows that fewer iterations are allowed during the process, which prevents ABC from being too greedy or getting trapped to a local optimal. As the negotiation proceeds, the fitness value depends more on the similarity between the detected solution and counter offer proposed by the negotiation opponent. More search iterations are introduced during the process, allowing employed bees and onlooker bees to explore more solutions than the earlier rounds, which increases the chance of finding a solution that is more likely to be accepted by the negotiation opponents without losing too much utility. This process is similar to the negotiation with the tradeoff tactic that the utility remains at a similar level from round 3 to round 6. In the final negotiation round, ABC makes the largest possible concession to maximize the likelihood of the last offer being accepted by the negotiation opponent. Figure 5.14 also shows that for long-term negotiations, ABC demonstrates better and more stable performance in terms of both utility and success rate, but it requires more negotiation rounds to keep the highest possible utility. Figure 5.15b shows the utility change of optimal solutions
under this situation. Similarly, ABC makes concessions in the early/ending rounds but is more conservative in the middle rounds than other tactics, which means it maintains a higher utility compared to other approaches.

Figure 5.16: Negotiation performance in the resource-rich environment

Figure 5.16 shows the negotiation results in the resource-rich environment. Compared to other baseline approaches, ABC demonstrates good performance in both success rate (90% approx.) and negotiation utility (0.6 approx.) despite the limitation on the negotiation deadline. Figure 5.17 shows the utility-changing trend of optimal solutions in the resource-
Chapter 5. Evaluation

(a) Utility-changing trend of different approaches (maxRound=10)

(b) Utility-changing trend of different approaches (maxRound=20)

Figure 5.17: Utility-changing trend in the resource-rich environment

rich environment. Different from other approaches, the utility curves of ABC in Figure 5.17 have different shapes to the curves in Figure 5.15. This is because ABC has no fixed rules on making concessions or tradeoffs like other approaches. It relies on the negotiation opponent’s counteroffer to propose new offers, which means changes to its utility is highly affected by the opponent’s negotiation model. Although BDR also adjusts concessions based on the recent counteroffers proposed by the negotiation opponent, it only imitates the opponent’s behaviour, while ABC combines the opponent’s counteroffer with negotiation deadline and its self utility to search for a win-win solution that is acceptable for both parties. This makes ABC more adaptable to the negotiation environment and achieve a higher negotiated utility than BDR. Also, the irregular utility change in each round makes it hard for the negotiation opponent to predict the concession, therefore the risk of accepting an offer with lower utility is reduced.

The Kruskal-Wallis test was performed to identify whether the performances of evaluated approaches are statistically significantly different. Figure 5.18 and Figure 5.19 present the comparison of ranked means, $p$-values calculated from Kruskal-Wallis test, and whether or not the null hypothesis $H_0$ is rejected. In this experiment, $H_0$ is rejected in all the test cases except for the success rate of long term negotiations in the resource-limited environment. The result of the statistical analysis is consistent with the previous discussions. For short term
(a) Ranked means of utility (maxRound=10)

(b) Ranked means of utility (maxRound=20)

(c) Ranked means of success rate (maxRound=10)

(d) Ranked means of success rate (maxRound=20)

(e) Ranked means of negotiation rounds (maxRound=10)

(f) Ranked means of negotiation rounds (maxRound=20)

Figure 5.18: Multiple comparison of ranked means (resource-limited environment)
Chapter 5. Evaluation

(a) Ranked means of utility (maxRound=10)

(b) Ranked means of utility (maxRound=20)

(c) Ranked means of success rate (maxRound=10)

(d) Ranked means of success rate (maxRound=20)

(e) Ranked means of negotiation rounds (maxRound=10)

(f) Ranked means of negotiation rounds (maxRound=20)

Figure 5.19: Multiple comparison of ranked means (resource-rich environment)
negotiations (i.e., when the number of negotiation rounds is small) in the resource-limited environment, ABC and TDL have statistically significantly better performance on success rate than RDP, UMC and BDR. For long term negotiations in the resource-limited environment, except for maintaining a high success rate, ABC’s negotiation utility is statistically significantly better than other approaches. For short term negotiations in the resource-rich environment, ABC and UMC have statistically significantly better performance on negotiation utility than BDR, TDL and RDP. ABC and TDL have statistically significantly better performance on success rate than RDP, UMC and BDR. For long term negotiations in the resource-rich environment, except for maintaining a high success rate, ABC’s negotiation utility is statistically significantly better than other approaches. For both environments, ABC requires more negotiation rounds to achieve higher utility.

Finally, to evaluate the overhead introduced by different approaches, the execution time is measured on three devices that may be used as negotiation gateways. Figure 5.20 shows the average execution time of evaluation approaches on different devices. ABC introduces more overhead than other approaches on resource-constrained devices like Raspberry PI and Android phone. The overheads of the five approaches have no obvious differences on the laptop computer, while the overhead of ABC increases faster as the device’s computational and storage capabilities degrade. For example, ABC introduces approximately 17.7%, 44.1% and 76.9% more overhead than TDL on the laptop, Android phone and Raspberry PI respectively for long-term negotiations in the resource-rich environment. If the negotiation gateway is a resource-constrained device like a Raspberry Pi, a tradeoff should be made between negotiation efficiency and responsiveness when choosing negotiation strategies.
Chapter 5. Evaluation

(a) Average execution time in resource-limited environment (maxRound=10)

(b) Average execution time in resource-rich environment (maxRound=10)

(c) Average execution time in resource-limited environment (maxRound=20)

(d) Average execution time in resource-rich environment (maxRound=20)

Figure 5.20: Average execution time on different devices
5.5 Evaluation of Distributed Negotiation using HNON

Given the scale of IoT devices likely to be deployed in different locations and the presence of mobile service providers/consumers that have limited communication ranges, iNegotiate proposes a distributed negotiation model that uses a self-organized hierarchical negotiation overlay network (HNON) to manage SLA templates, negotiation tasks and communications between negotiation entities. This section measures the performance of the proposed negotiation model by simulating distributed negotiations in a dynamic environment where mobile/static service providers/consumers communicate with gateways through WiFi. The negotiation environment is simulated on the Simonstrator platform [Richerzhagen et al., 2015] introduced in Chapter 4.2. This study outlines two evaluation experiments. The first experiment (Section 5.5.5) compares iNegotiate’s performance against a baseline approach under different simulation environments. The second experiment (Section 5.5.6) measures the impact of different environmental factors on system performance.

5.5.1 Performance Metrics

This study assumes a smart city environment where service providers, distributed in different locations, advertise their SLA templates to the gateway network and wait for negotiation requests. In such an environment, allocating a negotiation task to the gateways that can communicate with candidate service providers is a prerequisite for solving the request. iNegotiate specifies a location-based template/request distribution mechanism to increase the chance of finding candidate service providers and the gateways that can communicate with them. The SLA templates and gateways’ location information are managed in HNON, which facilitates the request forwarding process. To measure the performance of iNegotiate’s distributed negotiation model, the evaluation metrics used were:

- **The number of messages**: This metric measures the number of messages sent by gateways, which reflects network consumption in the simulations. This metric is used in the first experiment.
- **Success rate**: This metric measures the percentage of successfully solved requests,
which is defined as:

\[
\text{SuccessRate} = \frac{\text{Number of Solved Requests}}{\text{Total Number of Requests}}
\]  

(5.11)

This metric is used in both of the experiments.

- **Request delivery rate:** This metric measures the proportion of requests successfully received by the gateway network. The unreceived requests will cause negotiation failures that are independent of system performance. The request delivery rate is defined as:

\[
\text{RequestsDeliveryRate} = \frac{\text{Number of Delivered Requests}}{\text{Total Number of Requests}}
\]  

(5.12)

This metric is used in both of the experiments.

- **Response delivery rate:** This metric measures the proportion of responses successfully received by consumers, which reflects the system performance in addressing mobility problems. The response delivery rate is defined as:

\[
\text{ResponseDeliveryRate} = \frac{\text{Number of Delivered Response}}{\text{Number of Responses Sent by Gateways}}
\]  

(5.13)

This metric is used in both of the experiments.

### 5.5.2 Baseline Approaches

To demonstrate the feasibility and efficiency of using HNON to process negotiation tasks, this study compares iNegotiate’s distributed negotiation model against a baseline approach, which processes requests without using HNON. The negotiation process of the baseline approach is similar to iNegotiate. There is a system initialization phase that builds the communications between different negotiation gateways to enable a distributed negotiation scenario, a template distribution phase that allows gateways to acquire available service providers and their offerings in the environment, a negotiation task phase that propagates a request to several gateways to search for possible candidate service providers, a bilateral negotiation
Table 5.14: Different operations of FDP and *iNegotiate*

<table>
<thead>
<tr>
<th>Phase</th>
<th>Operation overview</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>FDP</strong></td>
</tr>
<tr>
<td>System initialization</td>
<td>- Gateways send ping messages to detect their neighbours</td>
</tr>
<tr>
<td></td>
<td>- Gateways create negotiation overlay network by exchanging configuration messages</td>
</tr>
<tr>
<td></td>
<td><strong>iNegotiate</strong></td>
</tr>
<tr>
<td></td>
<td>- Gateways send ping messages to detect their neighbours</td>
</tr>
<tr>
<td></td>
<td>- Gateways create negotiation overlay network by exchanging configuration messages</td>
</tr>
<tr>
<td>Template distribution</td>
<td>- Templates are saved at the gateways that receive the registration requests</td>
</tr>
<tr>
<td></td>
<td><strong>iNegotiate</strong></td>
</tr>
<tr>
<td></td>
<td>- Templates are saved at the followers that close to the service location</td>
</tr>
<tr>
<td>Negotiation task allocation</td>
<td>- Requests are propagated to neighbours until the message’s maximum hop is reached</td>
</tr>
<tr>
<td></td>
<td>- Negotiation results are collected and selected by the gateway that receives the request</td>
</tr>
<tr>
<td></td>
<td><strong>iNegotiate</strong></td>
</tr>
<tr>
<td></td>
<td>- Requests are propagated to controllers in sub-areas close to the requested service location and distributed to different followers</td>
</tr>
<tr>
<td></td>
<td>- Negotiation results are collected and selected by the initiating coordinator</td>
</tr>
<tr>
<td>Mobile entity locating</td>
<td>- The messages are propagated to neighbours until the message’s maximum hop is reached or a gateway successfully connects to the entity</td>
</tr>
<tr>
<td></td>
<td><strong>iNegotiate</strong></td>
</tr>
<tr>
<td></td>
<td>- The messages are firstly propagated to nearby sub-areas. If the entity is unreachable, the message is propagated to other sub-areas.</td>
</tr>
</tbody>
</table>

Phase that a gateway negotiates with a candidate service provider based on a consumer’s request, and a mobile entity locating phase that forwards a request/response to a mobile service provider/consumer. Although both approaches regulate the bilateral negotiation phase using the WS-Agreement Negotiation protocol [Waeldrich et al., 2011], the operations of the other four phases are different. *iNegotiate* creates HNON by assigning different roles to gateways and uses the HNON to organize SLA templates and forward negotiation tasks, while the baseline approach employs a fully decentralized architecture where no gateways are specified to cluster SLA templates or control message flows. Instead, it supports gateways to interact, in a peer-to-peer manner, with their neighbours to process negotiation requests. Therefore in this study, this baseline approach is referred to as the fully decentralized negotiation protocol (FDP). Table 5.14 outlines the different operations of these two approaches\(^\text{12}\).

\(^{12}\)Details of the baseline approach are published in [Li and Clarke, 2018]
During the simulation, each approach uses the same experiment configuration parameters described in Section 5.5.3.

5.5.3 Experimental Setup

The simulation environment is configured as Dublin city centre, where a set of static gateways are distributed in a grid topology. Some gateways can access the Internet through the Ethernet interface while others communicate through WiFi. Service providers and consumers are simulated to advertise SLA templates and request services. Service providers (mobile or static) and consumers (mobile) are randomly distributed in the environment. All the mobile entities follow a random movement pattern provided by Simonstrator. Consumers and mobile service providers are connected to the gateway network via WiFi while static service providers are connected to the gateway network via Ethernet. Considering the possible network congestion, gateways use the UDP send-and-reply mode to send messages, with the maximum time to wait for the ACK set to 2 seconds. To avoid the message flooding, the maximum hop of messages is restricted to 8, and the maximum round for bilateral negotiation is set to 10. The maximum waiting time for receiving the negotiation result is set to 2 minutes, which means if a consumer does not receive any signing request message within 2 minutes, a timeout failure is reported. Different from best-effort services, SLA-supported services require a negotiation process in which multiple interactions with the service provider is performed. Considering some contract-involved services (e.g., the vehicle service provided by Uber) set 2 minutes as the maximum waiting time [ube, 2020], this thesis assumes that the two-minute latency is acceptable for consumers.

The simulation uses a data set composed of 136 service prototypes created for service discovery in a smart city environment [Cabrera et al., 2018]. These prototypes specify service types, service locations and QoS parameters. Based on the prototypes, each service provider’s SLA template is generated by randomly assigning values to negotiable QoS parameters according to a predefined variation range. The spatial features of static services are non-negotiable, while the service locations of mobile services are flexible, which can be

---

13https://gitlab.scsc.tcd.ie/smartcitySD/data/services-dataset/-/tree/master/services
Table 5.15: Simulation Parameters of Experiment 1: Comparison with the Baseline

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of consumers</td>
<td>100</td>
</tr>
<tr>
<td>Number of gateways</td>
<td>150, 350</td>
</tr>
<tr>
<td>Number of service providers</td>
<td>200, 1000, 2000</td>
</tr>
<tr>
<td>Proportion of mobile service providers</td>
<td>50%</td>
</tr>
<tr>
<td>Moving speed</td>
<td>car</td>
</tr>
<tr>
<td>Test cases</td>
<td>Gateways X Providers</td>
</tr>
</tbody>
</table>

tailored according to consumers’ requirements. Each provider selects a random SLA template from the data set and sends a template registration message to advertise the selected template. Gateways perform template verification operations every 15 seconds. 100 service consumers are simulated to submit negotiation requests to the gateway network. To eliminate the negative impact on system performance introduced by irreconcilable conflicts between negotiation parties (e.g., disjoint negotiation spaces, unsupported functional or non-functional features), each negotiation request is generated based on a random template registered in the gateway network. This guarantees that the request is resolvable in the environment, a successful negotiation should be achieved if the service provider receives the request.

Table 5.15 outlines the simulation parameters for the first experiment, showing differing values for the number of gateways and the number of service providers. A set of test cases is designed to compare iNegotiate against the baseline approach under the impact of different parameter combinations. In this experiment, mobile entities move with a car speed that varies from 16.7 m/s to 27.8 m/s.

For the second experiment, four groups of test cases were designed to analyze the influence of different environmental factors on iNegotiate. Table 5.16 outlines the simulation parameters of the different groups of test cases. The first group measures the changes in system performance as the number of gateways increases. The simulation environment has a small number of service providers (i.e., 200 service providers), and half of the gateways have Internet connections. 50% of the service providers are mobile. All the mobile entities move with a speed that varies from 16.7 m/s to 27.8 m/s. The second group measures the changes in system performance as the proportion of Internet-connected gateways increases. The simulated environment is resource-constrained, where both the number of gateways and
Table 5.16: Simulation Parameters of Experiment 2: \textit{iNegotiate} Performance Analysis

<table>
<thead>
<tr>
<th>Group</th>
<th>Number of gateways</th>
<th>Number of providers</th>
<th>Proportion of mobile providers</th>
<th>Moving speed</th>
<th>Percentage of Internet connections</th>
<th>Test cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>150, 250, 350</td>
<td>200</td>
<td>1/2</td>
<td>car</td>
<td>50%</td>
<td>Gateway</td>
</tr>
<tr>
<td>Group 2</td>
<td>150</td>
<td>200</td>
<td>1/2</td>
<td>car</td>
<td>50%, 90%</td>
<td>Internet</td>
</tr>
<tr>
<td>Group 3</td>
<td>150, 350</td>
<td>150</td>
<td>0, 1/3, 2/3, 1</td>
<td>car</td>
<td>50%, 90%</td>
<td>Gateway X Mobile Provider X Internet</td>
</tr>
<tr>
<td>Group 4</td>
<td>150, 350</td>
<td>150</td>
<td>1</td>
<td>car, bike, walk</td>
<td>50%, 90%</td>
<td>Gateway X Speed X Internet</td>
</tr>
</tbody>
</table>

The number of service providers are small (i.e., 150 gateways and 200 service providers). 50% of the service providers are mobile, and all the mobile entities move with a speed that varies from 16.7\,m/s to 27.8\,m/s. The \textbf{third group} measures the changes in system performance as the proportion of mobile service providers increases. Four mobility scenarios are simulated as follows: (1) all the service providers are static (i.e., fully-static scenario); (2) a third of the service providers are mobile (i.e., partially-static scenario); (3) two-thirds of the service providers are mobile (i.e., partially-mobile scenario); and (4) all the service providers are mobile (fully-mobile scenario). The mobile entities move with a speed that varies from...
16.7\text{m/s} \text{ to } 27.8\text{m/s}. \text{ To analyze whether increasing the number of gateways or Internet connections can help to manage the mobility problem, the system performance of each scenario was measured under different environments where the number of gateways and their Internet connections are different. The last group measures the changes in system performance as the mobile entities’ moving speed decreases. All the simulated service providers are mobile, and three movement scenarios are defined as follows: (1) all the mobile entities move with a car speed that varies from 16.7\text{m/s} \text{ to } 27.8\text{m/s}; (2) all the mobile entities move with a bike speed that varies from 3.6\text{m/s} \text{ to } 4.2\text{m/s}; (3) all the mobile entities move with a walk speed that varies from 0.42\text{m/s} \text{ to } 0.5\text{m/s}. \text{ To analyze whether increasing the scale of gateways or Internet connections can help to manage the mobility problem, the system performance of each scenario was measured under different environments where the number of gateways and their Internet connections are different.}

The experiments were run on an Asus X751L laptop, which has a Windows 10 OS, Intel Core i7-4500U CPU, and 8GB RAM\textsuperscript{14}. To reduce the chance variation, each test case was repeated 10 times.

5.5.4 Statistical Analysis

This section outlines a statistical analysis performed to verify whether the observed results of the evaluated approaches are significantly different. Since the distribution of sample data determines which statistical test can be used to analyze the results, a normality test is performed to check whether the sample data can be modeled by a normal distribution. If the data follows a normal distribution then a parametric test (e.g., 2-sample t-test and one-way ANOVA) was used, otherwise, a non-parametric test (e.g., Mann–Whitney U test and Kruskal-Wallis test) was performed. Considering the small sample size in each experiment ($n = 10$), the Shapiro-Wilk test was used to check the normality of sample data [Shapiro and Wilk, 1965].

Mann–Whitney U test is a non-parametric test that assesses the differences between two independently sampled groups [Mann and Whitney, 1947]. The null hypothesis specifies that

\textsuperscript{14}Asus X751L details: https://www.asus.com/Laptops/X751LA/ - Accessed on 28 Apr 2020
these two groups are sampled from populations with identical distributions, which can be interpreted as values in group A are significantly different from those in group B. The procedure of the test involves pooling the observations from the two samples into one combined sample and calculating the test statistic denoted as $U$, which is the smaller value of $U_1$ and $U_2$ defined as follows:

\[
U_1 = n_1n_2 + \frac{n_1(n_1+1)}{2} - R_1 \\
U_2 = n_1n_2 + \frac{n_2(n_2+1)}{2} - R_2
\]  

(5.14)

where $R_1$ and $R_2$ are the sum of the ranks for group A and group B respectively, $n_1$ and $n_2$ are the sample size of each group. R\(^{15}\) is used to calculate the $p$-value based on Equation 5.14. The $p$-value under 0.05 (i.e., the confidence level is set to 95%) indicates the rejection of the null hypothesis. To quantitatively measure the strength of the statistical claim made by the Mann–Whitney U test, the effect size estimate $r$ is calculated as the $Z$ statistic divided by the square root of the sample size $N$ [Fritz et al., 2012]:

\[
r = \frac{Z}{\sqrt{N}}
\]  

(5.15)

Generally, $r \geq 0.5$ indicates a large effect, $r \in [0.3, 0.5)$ indicates a moderate effect, and $r \in [0.1, 0.3)$ indicates a small effect.

5.5.5 Result of Experiment 1: Comparison with the Baseline

Figure 5.21 shows the system performance under different test cases. Each graph shows the mean value observed from 10 replicated test cases with the standard error. As Figure 5.21a shows, iNegotiate has a higher success rate compared to FDP despite the number of gateways and the number of service providers. The success rate of FDP highly depends on the number of service providers in the environment, while iNegotiate relies more on the number of gateways. Since all the requests are resolvable in the experiment, the main reasons that cause negotiation failures are the negotiation timeout and incorrect request forwarding.

\(^{15}\)https://www.r-project.org/about.html - Accessed on 4 May 2020
Figure 5.21: System performance with variable number of gateways/providers

(a) Success rate

(b) Request delivery rate

(c) Response delivery rate

(d) Number of messages
A negotiation failure may happen under the following situations: (1) mobile entities can not connect to any gateway to submit their requests; (2) gateways fail to return the negotiation result to a consumer because the consumer is mobile, with limited communication range; (3) a request is not forwarded to the gateway that registers the candidate service.

The first situation is likely to occur when the number of gateways is insufficient to cover the negotiation environment. This is shown in Figure 5.21b where the request delivery rate improves by approximate 26% when the number of gateways increases from 150 to 350 with 200 service providers for both approaches. However, Figure 5.21b also indicates that the first situation is not the main reason that causes the differences in success rate between those two approaches.

The second situation happens when a consumer moves to an area that the gateway network can not reach, or the gateway network fails to forward the negotiation result to the gateway that can connect to the consumer. Figure 5.21c shows the response delivery rate that reflects a protocol’s ability to address communication problems introduced by mobile consumers. Compared to FDP, iNegotiate has a higher chance of sending negotiation responses to mobile consumers despite the gateway network size and the service scale. However, iNegotiate’s ability to address mobile communication problems is closely related to the gateway network size. The more gateways in the environment, the higher chance that mobile entities would connect to another gateway after moving a distance.

The third situation depends on the protocol’s request forwarding mechanism. As stated in Chapter 2.3.3, a distributed negotiation protocol needs to consider the tradeoff between efficiency and network consumption. In this experiment, the maximum message hop is set to 8 to avoid message flooding. Under this restriction, FDP does not have a sufficiently effective template distribution and request forwarding mechanisms to find the negotiation candidates that match the consumers’ requests, resulting in a low success rate. iNegotiate uses coordinators to collect information about gateways in different sub-areas and uses controllers to cluster SLA templates so that gateways have more information about other gateways as to where to forward the messages. However, the creation of a negotiation overlay network and collaborations between different types of gateways (i.e., controller, coordinator and follower)
introduce more communication overhead than FDP. Figure 5.21d shows the total number of messages sent by gateways under different test cases. For both approaches, the number of messages increases with the gateway network size and the scale of service providers. This increment happens because more messages are exchanged in the system initialization phase, and the increment of mobile service providers increases the number of template registration messages and mobile entity locating messages. When there are 150 gateways in the environment, \textit{iNegotiate}'s communication overhead is about 9\% more than FDP regardless of the number of service providers, while when there are 350 gateways and 200 service providers in the environment, \textit{iNegotiate}'s communication overhead is about 1.5 times more than FDP. However, the difference in communication overhead between these two approaches has narrowed down to 57\% with 2000 service providers. This indicates that although more gateways require more configuration messages for negotiation overlay network maintenance, the overlay network will not introduce too much traffic when there is a large number of service providers, and it effectively improves the success rate.

Since the observed results do not follow a normal distribution after applying Shapiro-Wilk tests, the Mann-Whitney U test was performed to verify whether the differences between the mean values are statistically significant. Table 5.17 shows the \(p\)-values obtained from the statistical test, and the corresponding size effect estimates are highlighted with bold. The request delivery rates of the two approaches are exactly the same, therefore the \(p\)-values and the size effect estimates can not be calculated. For all the metrics, the differences are statistically significant since the \(p\)-values are smaller than 0.05.

Although \textit{iNegotiate} achieves a higher success rate compared to FDP, it can not guarantee a successful negotiation when the request is actually resolvable in the environment. Section 5.5.6 will analyze the influence of different environmental factors on the system performance including the gateway network size, the number of service providers, the proportion of Internet-connected gateways, the proportion of mobile service providers and the moving speed of mobile entities.
Table 5.17: Statistical Analysis on *iNegotiate*’s performance

<table>
<thead>
<tr>
<th>Metric</th>
<th>200 Service Providers</th>
<th>1000 Service Providers</th>
<th>2000 Service Providers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>150</td>
<td>350</td>
<td>150</td>
</tr>
<tr>
<td>Success Rate</td>
<td>0.0001301 (0.863)</td>
<td>0.0001756 (0.847)</td>
<td>0.0001717 (0.85)</td>
</tr>
<tr>
<td>Request Delivery Rate</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Response Delivery Rate</td>
<td>0.0001678 (0.85)</td>
<td>0.0001668 (0.85)</td>
<td>0.0001659 (0.85)</td>
</tr>
<tr>
<td>Number of Messages</td>
<td>0.0001827 (0.845)</td>
<td>0.0001827 (0.845)</td>
<td>0.0001827 (0.845)</td>
</tr>
</tbody>
</table>

5.5.6 Result of Experiment 2: *iNegotiate* Performance Analysis

Figure 5.22 shows *iNegotiate*’s performance with a variable number of gateways. When the number of gateways increases from 150 to 250, the success rate increases by approximately 62%. When the number of gateways increases from 250 to 350, the success rate increases by approximately 45%. As described in Section 5.5.5, the increment in success rate is because a more dense gateway network increases the chance of connecting with mobile/static entities distributed in different locations. This is illustrated in the request delivery rate and the response delivery rate, which increase by 16% and 57% respectively when the number of gateways increases from 150 to 350.
Since the request forwarding and mobile entity locating process require the cooperation of coordinators deployed in different sub-areas, increasing the number of gateways also brings more Internet-connected gateways to the environment, which contributes to the system performance. This is verified by the simulation result of test cases in group 2. As Figure 5.23 shows, increasing the proportion of Internet-connected gateways from 50% to 90% can improve the success rate and response delivery rate by 86.3% and 52.8% respectively, even with a small number of gateways (i.e., 150 gateways). A sufficient number of Internet-connected gateways ensures each controller can find a coordinator to exchange messages with other sub-areas, therefore increasing the chances of correctly forwarding a request.

![Figure 5.23: System performance with variable number of Internet-connected gateways](image)

Figure 5.23: System performance with variable number of Internet-connected gateways

Figure 5.24 shows iNegotiate’s performance with different proportions of mobile service providers. Generally, mobility has a negative impact on the success rate. The reduction in success rate is caused because a negotiating gateway and a mobile service provider may disconnect during the bilateral negotiation process. All the subfigures in Figure 5.24 show a similar changing trend, suggesting that increasing network size or Internet connections will not mitigate the performance degradation introduced by mobile service providers. Although iNegotiate proposes a mobile entity locating mechanism to reconnect to mobile entities before and after bilateral negotiations, the handover process in which a negotiation session is transferred from one negotiating gateway to another without disconnecting the session is not considered. Once a mobile service provider moves out of the negotiating gateway’s communication range, a timeout failure will occur. However, the success rate in the fully-mobile
Chapter 5. Evaluation

203

(a) Performance with small network size and limited Internet connections

(b) Performance with small network size and adequate Internet connections

(c) Performance with large network size and limited Internet connections

(d) Performance with large network size and adequate Internet connections

Figure 5.24: System performance under different mobility scenarios
scenario is slightly improved compared to the partial-mobile scenario, especially when the number of gateways is large (Figure 5.24c and Figure 5.24d). This may be caused by incorrect template registrations of static services. As described in Chapter 3.4.2, SLA templates and negotiation requests are propagated in the gateway network according to location information. For static services whose spatial features are non-negotiable, once a template is registered in a gateway, it is unlikely that it would be transferred to another gateway unless the service provider updates the template with a new service location. The consistency between the template distribution and request forwarding mechanisms determines the likelihood of discovering a service as a negotiation candidate when it matches a request.

After analyzing the propagation route of template registration messages, an incorrect or invalid registration may happen under the two following circumstances: (1) there is a shortage of coordinators in the gateway network to propagate registration/negotiation requests over different sub-areas; (2) coordinators are more likely to forward SLA templates to the controllers with long ranges rather than the controllers that manage the gateway closest to the service location; (3) a template is registered in a gateway where a compatible request will not be forwarded to. Figure 5.25 shows an example under the second circumstance. Although $c$ is the gateway closest to the service location, the template is registered in gateway $d$ since the range of the controller $b$ does not cover the service location. Coordinator $d$ only propagates the registration request to controller $a$ since the minimum distance of sub-area
Figure 5.26: An invalid template registration scenario

A (i.e., $DF_A$) is smaller than the estimated minimum distance of sub-area B (i.e., $DF_B$). However, this incorrect registration will not affect the success rate since negotiation requests are only propagated to controllers whose range cover the requested service locations. Figure 5.26 shows the third situation where both of the two sub-areas do not cover the service location. Although the template is registered in the closest gateway $c$ because $DF_B$ is smaller than $DF_A$, the gateway network fails to identify this template as a negotiation candidate for a compatible request because neither controller $a$ nor $b$ receives the request. This indicates that relying on controllers’ ranges to forward requests cannot guarantee to find a service provider who can satisfy a consumer’s requirements even if the provider is available in the environment.

Figure 5.27 shows $iNegotiate$’s performance when mobile entities move with different speeds. Figure 5.27a and Figure 5.27b indicate that when the number of gateways in the environment is insufficient, the entities moving at walking speed achieve the highest success rate compared to the entities moving with car/bike speed. This is because these entities are less likely to move out of the gateways’ communication ranges after negotiations, which makes it easier for the initiating coordinator to locate them without asking other coordinators. Figure 5.27b shows that increasing the proportion of Internet-connected gateways can help to increase the response delivery rate, especially for the entities moving at car speed. This is because fast-moving entities have a higher chance of connecting to gateways in different
(a) Performance with small network size and limited Internet connections

(b) Performance with small network size and adequate Internet connections

(c) Performance with large network size and limited Internet connections

(d) Performance with large network size and adequate Internet connections

Figure 5.27: iNegotiate performance under different movement scenarios
sub-areas. The adequate Internet-connected gateways help the initiating coordinator forward the response to the sub-area where the entity is located, which increases the probability of successfully delivering the response. However, Figure 5.27c and Figure 5.27d show that if the network size is large, there is no obvious difference in system performance between different moving scenarios. Generally, the effect of entities’ moving speed on success rate is limited (i.e., approximately 5% maximum). The results of test cases in group 3 and test cases in group 4 indicate that iNegotiate’s limitation in solving communication problems introduced by mobile entities is the lack of a handover process that can transfer an unfinished negotiation session to another gateway when the provider moves out of the negotiating gateway’s communication range.

### 5.5.7 Evaluation Summary

This chapter presented the evaluation of iNegotiate, and includes four studies assessing the performance of WIoT-SLA template matchmaking, the trust-based candidates selection model, the bilateral negotiation strategy, and the distributed negotiation model.

The evaluation of the WIoT-SLA template matchmaking process addresses research question RQ.1, proposed in Chapter 1.4. A set of SLA templates were created according to WIoT-SLA ontology to simulate various similar services advertised by different service providers. These templates are different in feature names, resource configurations, data types and negotiation constraints. Three different semantic relatedness methods are integrated into the matchmaking algorithm to test the probability of correctly identifying candidate service providers. The evaluation results show that the matchmaking algorithm achieves the best performance when using WUP relatedness with the semantic similarity threshold set to 0.7.

The evaluation of iNegotiate’s trust-based candidates selection model addresses research question RQ.3, proposed in Chapter 1.4. A set of service providers are simulated based on the QoS dataset collected by invoking real IoT services. Each service provider has dynamic negotiation constraints that vary over time, and some providers dishonestly promise high QoS levels they cannot guarantee. iNegotiate’s trust evaluation model is implemented to select the optimal candidate service provider before a bilateral negotiation. The negotiation
result of trust-based candidate selection is compared against the results of competence-based selection, reputation-based selection, utility-based selection and random selection. The evaluation results show that in the trust model, the competence assessment effectively enhances negotiation efficiency and reputation assessment effectively reduces SLA violation rates. The trust model linearly combines competence assessment with reputation assessment, which can balance the tradeoff between negotiation efficiency and SLA compliance. However, the utility-based selection makes no obvious contribution to improving negotiation performance, and introduces more overhead than the competence-based selection and reputation-based selection, suggesting that utility assessment is unnecessary in the trust model.

The evaluation of iNegotiate’s negotiation strategy addresses research question RQ.4, proposed in Chapter 1.4. A multi-bilateral negotiation scenario is simulated where a gateway negotiates with multiple candidate service providers that have different negotiation constraints. Two experiments were designed to test the performance of the context-based strategy and ABC-based strategy. The negotiation efficiency is compared against four other approaches: a game theory-based mixed approach, a behaviour-dependent approach, a time-dependent approach, and a resource-dependent approach. The evaluation results show that both the context-based strategy and ABC-based strategy have better performance in balancing the tradeoff between success rate and negotiation utility. They both can adapt to the negotiation environment to achieve a high success rate, and reduce the loss of utility when more iterations are allowed in the negotiation session. The overhead introduced by context-based strategy is similar to other approaches, while ABC-based strategy introduces more overhead on resource-constrained devices.

The evaluation of iNegotiate’s distributed negotiation model addresses research question RQ.2, proposed in Chapter 1.4. A dynamic negotiation environment where service providers and consumers can be mobile and connect to gateways using WiFi is simulated on the Simmonstrator platform. Two experiments were designed to test the system performance. The first experiment compares the iNegotiate’s negotiation model with a baseline approach that processes negotiation requests without using HNON. The evaluation results show that the iNegotiate achieves a much higher success rate in different negotiation environments, but it
requires more messages to coordinate negotiation tasks and locate mobile entities. The second experiment analyzes the influence of environmental factors on the system performance. The evaluation results show that the number of gateways, especially the number of Internet-connected gateways, has a great impact on the success rate. However, increasing the gateway network size can not fully address the communication problems introduced by mobile entities. A negotiation handover process is required to transfer the negotiation session to another gateway when the network connection between the current negotiating gateway and the mobile service provider is lost.
Chapter 6

Conclusion

This thesis investigates distributed SLA negotiation in dynamic large-scale IoT environments. Current research on SLAs and SLA negotiations focus on cloud services and web services, which are insufficient to accommodate IoT domain-specific characteristics such as distributed large-scale resources and dynamic negotiation environments. To address the problems of SLA negotiation in IoT environments, this thesis proposes iNegotiate, which is a distributed SLA negotiation system that uses a negotiation gateway network to automatically tailor service properties with service providers according to a consumer’s request before creating an SLA. iNegotiate configures a negotiation overlay network where gateways collaboratively process requests based on service locations and perform distributed multi-bilateral negotiations to search for a global beneficial agreement with the highest possible utility. The previous chapter discussed iNegotiate’s performance with a set of simulation experiments. This chapter summarizes the thesis, describes the contributions and limitations of iNegotiate, and outlines future research directions.

6.1 Thesis Summary

Introduction Chapter 1 introduced the motivation for, and challenges of, applying SLA and SLA negotiations in service-oriented IoT environments. It analyzed the limitations of existing solutions in addressing these challenges and identified the requirements of SLA negotiation
Chapter 6. Conclusion

according to IoT’s characteristics. Four research questions with regard to negotiation objects, negotiation protocols, negotiation candidates selection, and negotiation strategies are proposed to accommodate the identified requirements. Based on these research questions, this chapter presented the hypotheses, objectives, and contributions of the thesis.

State of the Art Chapter 2 described current research related to SLA and SLA negotiations in different computing environments, and analyzed the extent to which the state of the art meets the requirements of automatic SLA negotiation proposed in Chapter 1.3.4. This chapter first reviewed the work of SOA-based IoT systems and presented the existing SLA languages, SLA management mechanisms, and IoT ontologies that abstract things and data as services. Then it studied how current negotiation frameworks resolve conflicts between service consumers and service providers using different negotiation protocols and negotiation strategies, and identified the limitations of these approaches in addressing SLA negotiation problems in large and dynamic IoT environments. Finally, this chapter explored different provider selection approaches and specified their drawbacks as a negotiation candidate selection mechanism.

Design Chapter 3 introduced the design objectives and design decisions for iNegotiate according to the research questions and knowledge gaps identified in Chapter 1 and 2. Then it presented iNegotiate’s system model and the detailed design of each component. iNegotiate uses the WIoT-SLA ontology to formalize SLAs, SLA templates and negotiation offers, which facilitate automatic template matchmaking and SLA negotiation. iNegotiate creates a hierarchical negotiation overlay network based on the actual gateway network topology to effectively propagate messages in a dynamic large-scale environment where negotiation participants may be mobile and each gateway only has a partial knowledge about the environment. In order to get the best possible solution within a limited negotiation time, iNegotiate prioritizes candidate service providers using an experience-based trust model and selects the Top-K candidates with which to start bilateral negotiation sessions. To balance the trade-off between negotiation utility and success rate for bilateral negotiations with incomplete information, iNegotiate applies a deadline-aware negotiation strategy to dynamically adjust
concessions according to context information or the negotiation opponent’s behaviour.

**Implementation** Chapter 4 presented *iNegotiate*’s main components including the interaction handler that processes the messages from negotiation entities or other gateways, the SLA manager that stores SLA and SLA templates, the template matchmaker that identifies candidate service providers according to WIoT-SLA ontology, the trust evaluator that assesses candidate service providers based on historical records, and the negotiator that analyzes received offers and proposes new counter offers according to the negotiation strategy. This chapter also described how *iNegotiate*’s distributed negotiation model was implemented on the Simonstrator platform and the operations relating to receiving a particular type of message.

**Evaluation** Chapter 5 evaluated how well *iNegotiate* addresses the research questions proposed in Chapter 1.4. Four evaluation studies are described to present the simulation experiments, performance metrics, and evaluation results with regard to *iNegotiate*’s template matchmaking mechanism, distributed negotiation protocol, trust-based candidates selection and bilateral negotiation strategy. Results show that WIoT-SLA template matchmaking achieves the best performance using WUP relatedness with the similarity threshold set to 0.7. The distributed negotiation protocol improves the success rate compared to the purely decentralized negotiation protocol at the cost of more exchanged messages. The competence assessment and reputation assessment in the trust-based candidate selection model enhance the negotiation efficiency and SLA compliance respectively. And, both the context-based negotiation strategy and ABC-based negotiation strategy show how *iNegotiate* adapts to different negotiation environments, which achieves a higher success rate compared to other baseline approaches.

### 6.2 Discussion

This thesis proposes *iNegotiate* according to the requirements of automatic SLA negotiation in dynamic IoT environments. Chapter 5 demonstrates the performance of *iNegotiate* from
Chapter 6. Conclusion

This section outlines the contributions of this thesis and discusses iNegotiate’s limitations based on the evaluation results.

6.2.1 Thesis Contributions

This thesis has made four contributions with regard to automating SLA negotiation in dynamic IoT environments. The first contribution is the design of the WIoT-SLA ontology. Previous SLA languages target cloud services or web services, while the IoT domain-specific properties are not considered. A uniform SLA ontology is a prerequisite to achieve semantic interoperability and automate SLA management activities. iNegotiate proposes an SLA ontology based on existing web service SLA specifications to formalize the SLA, SLA template, and negotiation offer, allowing service providers to express their offerings in a standardized way. In an SLA template, the service terms that describe an IoT service’s functional and non-functional features facilitate automatic template matchmaking when discovering negotiation candidates. The negotiation context and creation constraints that specify a negotiation protocol, negotiation interface, and negotiable terms support negotiation-based SLA creation. The guarantee terms and quality metrics that define QoS guarantees, assessment intervals, and measurement metrics can be used to configure the SLA monitoring instance. Chapter 5.2 reports the template matchmaking efficiency using the WIoT-SLA ontology. The limitations of this contribution are discussed in Section 6.3.1 and Section 6.3.6.

The second contribution is the distributed SLA negotiation model designed for dynamic IoT environments. Existing negotiation approaches for IoT assume a centralized cloud-based architecture, which may be impractical because of the scale of localized resources distributed in different locations, and the presence of wireless connected negotiation entities that may be mobile and have limited communication ranges. iNegotiate proposes a distributed negotiation protocol that allows gateways to communicate through an overlay network to cooperatively accomplish negotiation tasks. The negotiation overlay network is created during the system initialization phase, which divides the environment into a set of sub-areas. Each sub-area has a controller that clusters SLA templates registered in the local area, a coordinator that
propagates messages to other sub-areas, and a set of followers that negotiate with candidate service providers. SLA templates are distributed to gateways close to the service locations and replicated in the controllers, which facilitate request forwarding and negotiation candidates detection. A layered-based communication mechanism is designed where coordinators forward a request to controllers of subareas near the service location, and controllers forward the request to followers that register the candidate templates. This design reduces unnecessary message transmissions in the negotiation task allocation phase and mobile entity locating phase. Chapter 5.5 reports the negotiation success rate and communication overhead using the distributed negotiation protocol. The limitations of this contribution are discussed in Section 6.3.3 and Section 6.3.4.

The third contribution is the negotiation candidate selection mechanism that selects trustworthy candidate service providers with whom to negotiate when multiple candidates are detected. Previous service selection approaches use service matchmaking or a reputation system to identify optimal services, which ignore the negotiation performance of corresponding service providers. iNegotiate proposes a negotiation-oriented trust model that evaluates candidate service providers based on their historical records. The trust model is composed of a competence assessment model and an integrity assessment model, which infer the possibility of successful negotiation and the possibility of a service provider keeping its promises, respectively. The past negotiation result, the SLA violation rate of previously offered services, and the current service performance are analyzed to avoid making an agreement with a fraudulent service provider that uses deceptive advertisements to attract service consumers. Chapter 5.3 reports the efficiency of the trust-based candidate selection mechanism. The limitations of this contribution are discussed in Section 6.3.6.

The fourth contribution is the negotiation strategy that evaluates received offers and generates counter offers in the bilateral negotiation process. Previous negotiation strategies designed for the cloud SLA negotiation do not consider IoT service properties, and they are either too heavyweight for the IoT SLA negotiation, or not flexible enough to adapt to the environment changes. To quantitatively evaluate a negotiation offer, iNegotiate defines three scoring functions targeting different negotiable service terms. To improve the negoti-
ation success rate, *iNegotiate* uses a deadline-aware decision-making model that chooses to accept/reject a received offer or propose a new counter offer based on the offer evaluation results and the state of the received offer. To maintain the highest possible utility, *iNegotiate* either uses a context-based negotiation tactic to makes concessions or tradeoffs according to the negotiation deadline, available resources, and the consumer’s negotiation preference or uses a bio-inspired negotiation tactic to search for a globally beneficial solution that is acceptable for both parties. Chapter 5.4 reports the performance of the proposed negotiation strategy using the two different negotiation tactics. The limitations of this contribution are discussed in Section 6.3.5 and Section 6.3.6.

### 6.3 Limitations and Future Work

This thesis shows the feasibility and efficiency of *iNegotiate*, a distributed system for automatic SLA negotiation in dynamic IoT environments. This section will discuss the limitations of *iNegotiate* and identify future research directions.

#### 6.3.1 SLA Template Matchmaking and Knowledge Ontology Maintenance

*iNegotiate* identifies candidate service providers by matching a request with registered SLA templates according to the WIoT-SLA’s multi-phase template matchmaking mechanism. Simulation results in Chapter 5.2 show that filtering templates based on semantic similarity falsely identifies some candidate templates that cannot satisfy the consumer’s request. This incorrect matchmaking not only introduces unnecessary computation costs in the subsequent candidate selection process and bilateral negotiation phase, but also decreases the consumer’s satisfaction level. An SLA specification tool integrated with a global knowledge ontology that defines IoT service’s functional and non-functional features may help to solve the problem. Service providers and service consumers can use the tool to acquire the service terms defined in the knowledge ontology and choose the most suitable term with the most appropriate meaning to create an SLA template or a request.

Future research will develop an SLA specification tool to simplify the process of creating
SLA templates/requests for service providers/consumers. Except for formalizing SLA templates according to the WIoT-SLA ontology, this tool will also provide a global knowledge ontology to generalize IoT services’ features and QoS metrics. This reduces the possibility of service providers and consumers using different terms to describe the same service feature, which further eliminates the necessity of checking semantic similarities. Different from the WIoT-SLA ontology that specifies the structure of SLAs and SLA templates, the global knowledge ontology defines a set of commonly-used service terms, which can be summarized from the registered templates. The service terms that represent similar functional or non-functional features can be identified and clustered under the same category to facilitate the term selection and template matchmaking. Under this assumption, gateways can identify candidate service providers by checking whether or not two terms belong to the same category rather than calculating the semantic similarities between them, which may help to reduce the processing time and increase the matchmaking precision. Considering the evolution of service technology and the possible new features that may emerge in the future, the global knowledge ontology should be continuously updated and maintained to increases the chance of service providers and consumers finding the appropriate service terms. Semantic negotiation [Comi et al., 2015] and machine learning techniques [Cassar et al., 2013] are the possible ways to automatically enrich the global knowledge ontology. For instance, a service provider creates a new term based on the structure of SLA terms defined in WIoT-SLA ontology, the SLA specification tool can compare the term with existing terms according to the attributes defined in it (e.g., the measurement metrics of a service property, or the value and data type of a configuration item). If the specification tool finds the new term is closely related to another existing term, it can verify the result with the service provider by triggering a semantic negotiation process. If the service provider agrees that these two terms have similar meaning, they can be clustered to the same category in the global knowledge ontology.

Inspired by the bounding sphere used for real-time collision detection in game development, iNegotiate uses a 2D circular region to model service coverage (i.e., the service coverage in Chapter 3.4.1). Although this design can reduce computation complexity during the template matchmaking and SLA negotiation process, this assumption may be unpractical for IoT
services that have irregular sensing ranges or the services whose coverage can be affected by environment factors such as wind or thunderstorm. As an existing semantic model for IoT services, IoT-Lite uses circle, polygon and rectangle to model device coverage [Bermudez-Edo et al., 2016]. But the tools that can discover whether a point belongs to an area is not clearly specified. How to balance the tradeoff between the computation cost and the precise definition of service coverage is challenging, which is worth to be explored in future work.

6.3.2 Negotiation Task Allocation

iNegotiate distributes template registration requests and SLA negotiation requests through the hierarchical negotiation overlay network (HNON) based on location information. The simulation results on the Simonstrator platform demonstrate that the current request forwarding mechanism in HNON may cause invalid template registration problems as presented in Figure 5.26 (Chapter 5.5.6), which further causes negotiation failures. Future work will improve the negotiation task allocation phase by adding a process that forwards a request to the closest sub-area if there is no sub-area that covers the requested service location.

6.3.3 Mobile Entity Management

iNegotiate uses HNON to manage the communications between gateways and negotiation entities. Simulation results in Chapter 5.5 show that iNegotiate achieves a higher success rate and response delivery rate compared to the baseline approach. However, iNegotiate can not fully address the communication problems introduced by mobile service providers, especially when the number of gateways is insufficient to cover the negotiation environment. A timeout failure still occurs if a provider moves out of the negotiating gateway’s communication range during the bilateral negotiation phase. Future research will extend iNegotiate’s distributed negotiation model with a handover process that can transfer an unfinished bilateral negotiation session to another gateway to avoid the session being terminated by timeout errors. A soft handover mechanism that allows parallel connections between the service provider and some gateways may help to provide effective communication support for bilateral negotiations with mobile service providers. Also, current IoT applications have mobile devices acting as
gateways [Sigwele et al., 2018]. Using these mobile devices to relay messages may help to improve the success rate in a resource-limited environment. Future work will integrate mobile gateways into the negotiation overlay network to propagate undelivered requests/responses to sub-areas that have coordinators, taking advantage of their movement through the environment.

6.3.4 Network Efficiency

iNegotiate is a distributed negotiation framework that implements a hybrid centralized-decentralized architecture using HNON, which allows gateways to autonomously organize SLA templates and coordinate negotiation tasks without a centralized infrastructure. The simulation results on the Simonstrator platform demonstrate that the hybrid architecture improves the negotiation success rate compared to a fully decentralized architecture when the maximum message hop is restricted. However, it introduces a larger number of exchanged messages between gateways (Chapter 5.5), and the number of gateways has a significant impact on network efficiency. This is because gateways spread their physical network topology to other gateways to create HNON in the system initialization phase. Also, iNegotiate uses a layer-based communication mechanism where the follower layer propagates messages to the coordinator layer through the controller layer, but iNegotiate only considers communication efficiency between followers and controllers, and efficiency between controllers and coordinators when creating HNON. Future work will optimize the message propagation route between the follower layer and the coordinator layer to avoid unnecessary message transmissions.

Due to the huge revolution in wireless technology introduced by 5G, recent research discussed IoT systems that use the 5G network to achieve high bandwidth, fast data transmission rate, reliable communication, massive connection, and low-latency [Cheng et al., 2018, Al-Turjman et al., 2018, Chettri and Bera, 2019, Ahad et al., 2020]. In the 5G IoT architecture, sensors can rely on IoT gateways to transmit data to 5G base stations via the 5G communication link [Chettri and Bera, 2019]. Considering the problem of explosive data volume produced in IoT, 5G communication proposed the ultra-dense Heterogeneous Networks (HetNets) that combines different-sized cells to achieve a higher spectrum efficiency.
[Chettri and Bera, 2019, Al-Turjman et al., 2018]. The small cell (i.e., picocell, femtocell, and microcell) base stations are deployed within short ranges to transmit data with a carrier frequency about 3.5 GHz, while the large macro base stations that have massive Multiple-Input–Multiple-Output (MIMO) antennas with the additional capability of beamforming, are deployed within a distance of kilometers with the carrier frequency of 28 GHz and above. This mm-wave communication is a key technology of the 5G network in enhancing the network connection capacity, network coverage, communication reliability, and energy efficiency. With the market transitioning to 5G, HetNets can be expected to provide connection interfaces for SLA negotiation. Based on the similarity of hierarchical structure between HNON and HetNets, it is possible to merge HNON to HetNets to take advantage of benefits brought by HetNets. For example, the large macro base stations can work as coordinators to propagate messages between different sub-areas since macrocell has a large communication range. Base stations in the Femtocell and Picocell can work as followers to perform bilateral negotiations because they may be closer to the negotiation entities (i.e., consumers and providers). The microcell base stations can work as controllers since microcell covers a relatively larger area (e.g., several hundred meters) than a picocell but has a shorter communication range compared to the macrocell. Considering the significant QoS improvement using the 5G network, it is reasonable to hypothesize that deploying inNegotiate on 5G base stations can help to reduce timeout failures. However, the inter-cell interference coordination schemes and the load-balancing mechanism used in HetNets (e.g., Cell Range Expansion) [Lee et al., 2015] may cause communication interruption and inconsistency problems when allocating negotiation tasks. Future work will investigate the most recent 5G solutions and explore the feasibility of using 5G base stations as inNegotiate gateways.

6.3.5 Composite Service Negotiation

inNegotiate uses a deadline-aware negotiation strategy to balance the tradeoff between success rate and negotiation utility. Simulation results in Chapter 5.4 show that inNegotiate achieves a higher success rate compared to other approaches. This is because inNegotiate tries to make an agreement with a service provider in the last negotiation round by making the
biggest possible concession as an ultimatum. However, this might be an issue for composite service negotiation where service consumers only provide the global negotiation boundary but the individual negotiation boundary for each atomic service is unknown. Future research will extend the negotiation strategy with a negotiation boundary decomposition mechanism [Richter et al., 2012] that dynamically allocates negotiation boundaries of atomic services as the negotiation proceeds according to the global negotiation constraints specified by the service consumer.

6.3.6 Computational Efficiency

The evaluation studies presented in Chapter 5 show that \textit{iNegotiate}'s template matchmaking (Chapter 3.4.1), trust-based candidate selection (Section 3.4.3), and ABC-based negotiation strategy (Section 3.4.4) introduce a large latency on resource-constrained devices like the Raspberry Pi, especially when there is a large number of service providers in the environment. Although \textit{iNegotiate} allows computation tasks to be processed on different gateways by organizing SLA templates in the negotiation overlay network according to the service locations (i.e., the multi-phase template matchmaking is collaboratively performed by controllers and followers, and the negotiation request is distributed to different followers according to the templates they registered), the gateways’ computation capabilities are not considered when creating the negotiation overlay network. To improve the computational efficiency of \textit{iNegotiate}, two following aspects are worth to be explored: (a) Making full use of available resources in the environment. Current fog-based IoT environments have various devices that can possibly be used as negotiation gateways; (b) Proactive negotiation based on environment context and historical negotiation records.

Considering the large amount of data produced by IoT devices and the requirements such as mobility support, location awareness and low latency contributed by IoT applications, fog computing and edge computing have emerged as new paradigms to resolve the issues in cloud-based IoT by extending both computation and storage services in cloud to the network edge [Shahzadi et al., 2019]. In fog computing, LAN hardware (e.g., IoT gateways, wireless routers, cell base stations, WiFi access points, etc.) can be used as \textit{Fog nodes} that are
deployed close to end users to carry out computation tasks. As devices become more and more advanced, modern IoT envisions a fog-based solution where a geo-distributed network of smart gateways provides intelligence (e.g., data filtering, data pre-processing, device monitoring, resource management, etc.) locally to satisfy the QoS requirements of latency-sensitive applications [Rahmani et al., 2018, Krishnan and Vasudevan, 2019]. These gateways may be varied in terms of computation capabilities due to the different requirements of IoT applications [Papcun et al., 2019]. Considering the possibility of deploying iNegotiate on these heterogeneous smart gateways, future research will investigate the creation of a capacity-aware negotiation overlay network where negotiation gateways are classified into different roles according to both network topology and their computation capabilities. For example, powerful devices can be assigned as controllers to perform computation-intensive tasks such as template matchmaking and SLA monitoring; an SLA template can be registered in several followers close to the service location so that the controller can dynamically allocate a negotiation task to the follower that has the minimum workload. A load balancing mechanism similar to adaptive routing [Kotagi et al., 2017] during the negotiation task allocation phase may be a possible way to reduce processing latency and increase the system’s ability to process concurrent requests. Since implementing iNegotiate in the event-based Simonstrator platform can not measure the processing latency when the gateway network is composed of different types of devices, a real-world implementation is required to test its performance in addressing scalability issues and detect possible bottlenecks.

The current version of iNegotiate only performs reactive negotiations, which means a bilateral negotiation is triggered only after receiving a negotiation request from a consumer. Considering the template matchmaking and bilateral negotiation may be time-consuming in a sub-area that has insufficient powerful gateways, proactive negotiation that performs bilateral negotiations before receiving the actual requests may be a possible way to enhance the system responsiveness. For instance, the regular behaviour patterns of users may bring periodic requests [Cabrera and Clarke, 2019] (e.g., a user requests a parking service around location A every Friday evening and requests a traffic monitoring service every morning from Monday to Friday), which can be used as a basis to perform proactive negotiation. Future
research might explore the machine learning techniques that can predict the possible request based on historical negotiation records, and perform the proactive negotiation based on the predicted request to get the latest possible offer.

6.3.7 Fault Tolerance

iNegotiate relies on the logical HNON to process requests, and the HNON is automatically created based on the physical network topology. However, due to the possible temporal device malfunction and the unforeseen events that may happen in the environment (e.g., the power outage caused by a thunderstorm), gateways may be online or offline at any time. The connection problems within the gateway network will cause timeout failures, which decreases the negotiation success rate. The network topology changes under one of the following situations: (a) A new device joins the gateway network; (b) An existing device stops working as a negotiation gateway (i.e., fully disconnected); (c) An existing device loses its Internet connection, but it can still communicate with other gateways through WiFi (i.e., partially disconnected).

Generally, the new joiners have no impact on the system performance since it has not been acknowledged by the logical HNON. But the disappeared gateways and interrupted Internet connections may significantly decrease the negotiation success rate, especially when the gateway is a controller or a coordinator. This is because the HNON relies on coordinators and controllers to propagate requests and allocate negotiation tasks. Losing connections with those two types of gateways will make requests unable to reach to the appropriate followers. To avoid the timeout failures caused by disconnected gateways, the HNON should be constantly updated according to the dynamic changes of the underlying physical network. Therefore, rather than only executing the HNON creation algorithms (i.e., Algorithm 2, 3 and 4) in the system initialization phase, these algorithms should be periodically executed to adjust the role assignment based on the run-time network status. Only the gateways available in the network will participate in the overlay creation process. This allows the HNON to acknowledge the new joiners and ignore the currently off-line gateways. The state-controlled message propagation mechanism used in these algorithms prevents the operation
from introducing too much traffic if there are no big changes in the underlying network. However, periodically creating the HNON can not fully address the problems introduced by disappeared gateways. The template distribution mechanism should also be updated accordingly to ensure a registered SLA template can be discovered when a compatible request comes. Since SLA templates are registered in followers and clustered in controllers, every time a gateway is assigned as a new controller (i.e., the gateway receives a commission verification message whose operation code is CVM), the followers should send a copy of their registered template to the new controller to facilitate the matchmaking process. If a gateway is no longer a controller (i.e., the gateway does not receive any commission verification message), it deletes the follower list and removes all backup templates. Although periodically creating HNON and adjusting template distribution may solve the problems of disappeared controllers and disappeared coordinators, the disappeared followers may further cause data lost using this mechanism since the controller may be changed after re-creating the HNON, and the new controller can not get the templates registered in the disappeared follower. A possible solution to this problem might be registering an SLA template in multiple nearby followers in each sub-area rather than just registering it in a single follower. The replicated template registration not only reduces the risk of losing registered SLA templates, but also allows the controllers to dynamically select the optimal follower to perform bilateral negotiations according to runtime workload and device status. Future work will extend iNegotiate protocol with the fault tolerance mechanism discussed above, and implement it in the Simonstrator platform to test its feasibility and efficiency in a highly dynamic environment.

6.3.8 Security and Privacy

As described in Chapter 3.4.1, iNegotiate does not participate in the actual service delivery process, it only communicates with the candidate service providers to tailor the service features that can satisfy both parties before the service delivery. Therefore the data security, user authentication, and privacy protection during the service provisioning time are outside the scope of iNegotiate. However, iNegotiate should consider the security problems during SLA negotiations because the negotiation efficiency and users’ satisfaction levels are closely
related to the service information disclosed in SLA templates. If a registered template is modified by malicious competitors such as decreasing the possible QoS guarantees or removing some important QoS features, this service may lose negotiation opportunities since the template is filtered out during the matchmaking process. In other words, when the SLA templates are publicly available, the privacy of service providers and data integrity may become a major concern.

To reduce this potential risk, iNegotiate should have security support that considers confidentiality and integrity. The SLA and SLA templates should be hidden confidentially from malicious attackers and only authorized users are allowed to interact with the gateway network. Existing fog-based IoT systems have discussed the usage of secure hash algorithm-1 (SHA-1), advanced encryption standard (AES) [Hu et al., 2017], lightweight privacy preserving data aggregation (LPDA) scheme [Lu et al., 2017] to deliver integrity, confidentiality, and availability. Also, as introduced in Chapter 2.2.4, considering the distributed nature of IoT systems, the combination of smart contracts and blockchain may pave the way for automatic SLA management in an open and trustless service market [Uriarte et al., 2020]. In blockchain systems, the distributed ledger and time-stamped blocks that are linked together using cryptographic hashes prevent the historical records from being modified by malicious nodes. The public-private key pairs ensure that all data of the network is secured with strong cryptographic encryption and only the authorized users can decrypt the information. Users are anonymized by using addresses (i.e., public keys) as their accounts, which cannot be easily traced back to their owners without out-of-network information. The consensus algorithms such as practical byzantine fault tolerance (PBFT) [Castro et al., 1999], proof of elapsed time (PoET) [Chen et al., 2017] and proof-of-work (PoW) [Nakamoto, 2009] prevent a single node from dominating the entire blockchain network and manipulating the transaction history for its own benefit. Existing research has discussed the probability of transforming an SLA into a smart contract, which is executed in the blockchain to raise trust in data integrity and SLA assessment [Uriarte et al., 2020]. However, due to the limitations of blockchain such as low transaction confirmation speed and inability to interact effectively with the outside world, performing on-chain automatic SLA negotiation is still challenging at the moment.
[Uriarte et al., 2018]. Future work will explore the blockchain techniques that can possibly be used by iNegotiate to secure template registrations and negotiation interactions. For instance, the communication between a negotiation entity (i.e., service provider/consumer) and iNegotiate can be protected using public-private key pairs. A negotiation entity needs to register in iNegotiate by exchanging its public key with the gateway network before submitting a request. Based on the public key, the gateway network generates a unique identifier for each negotiation entity. Every time a new message is received from the entity, the gateway network can verify the confidentiality and authenticity of received data using the public key and the corresponding signatures in the message. Another possible solution to assure the integrity of registered SLA templates is the use of cryptographic hash. When an SLA template is registered or updated by the service provider, the cryptographic hash of the template will be distributed to all the gateways within the local sub-area. During the valid time of the template, some randomly selected gateways verify the template registered in the controller/follower by re-calculating the cryptographic hash at regular intervals. If there is any inconsistency, the corrective action (e.g., send a template verification message to the provider, reset the controller/follower and restart the system configuration phase) will be triggered.

6.3.9 Summary

The simulation results in Chapter 5 demonstrate the feasibility of iNegotiate in addressing SLA negotiation problems in dynamic IoT environments. Generally, iNegotiate has a certain level of requirements on the gateways’ computation and network capabilities to achieve a good performance. It demonstrates a high success rate when the gateway network size is sufficient to cover the negotiation environment. When there is a shortage of gateways in the environment, iNegotiate requires a certain amount of internet-connected gateways to correctly propagate messages. The responsiveness of iNegotiate is closely related to the gateway’s computation capability. For example, iNegotiate may introduce large latency on Raspberry Pis depending on the number of service providers, while the latency on laptops is acceptable.
Although formalizing SLA templates using the WIoT-SLA ontology can help to identify the candidate service providers that have the potential to satisfy a consumer’s functional and non-functional requirements, the use of semantic similarity checking in iNegotiate can not guarantee precise matchmaking, which introduces unnecessary computation costs. This indicates that creating an SLA specification tool and a global knowledge ontology may be a possible solution to improve template matchmaking efficiency. Although iNegotiate creates a negotiation overlay network to distribute SLA templates and negotiation requests, the gateways’ computation capabilities are ignored when allocating negotiation tasks, which may cause large latency in candidate selection phase and bilateral negotiation phase. Considering the IoT environment may have various types of devices that can be used as negotiation gateways, the capacity-aware negotiation overlay network where computation tasks are distributed to different gateways according to the gateways’ capabilities and their current workload may help to reduce processing time. iNegotiate also shows limitations when addressing mobility problems. Extending iNegotiate’s protocol with a handover process and integrating mobile gateways to the negotiation overlay network are possible solutions to reduce timeout failures. Currently, iNegotiate does not support SLA negotiation of composite services. This may be solved by extending the negotiation strategy with a negotiation boundary decomposition mechanism.
Appendix A

Rough Set Theory

In rough set theory, an information system can be represented in the form of an $m \times n$ decision table [Skowron et al., 2002][Jensen and Shen, 2007], which is simply denoted as $IS = (U, A, f)$, where $U$ is the universe domain of discourse with $m$ objects (i.e., observations in different rows), $A$ is a finite set of attributes consisting of $n - 1$ conditional attributes $C$ and a decision attribute $D$ (i.e., attributes in different columns). Each attribute $a \in A$ has a set of values $V_a$, $f$ is the function that denotes the map of $u \times A \rightarrow V$. With any $R \subseteq C$, the equivalence relation $IND(R)$ defined in Equation A.1 generates the partition $U/R$ where $x$ and $y$ are indiscernible by attributes from $R$.

$$IND(R) = \{(x, y) \in U^2 | \forall a \in R, f_a(x) = f_a(y)\}$$ (A.1)

The expression $Pos_C(D)$ represents the positive region of the partition $U/D$ with respect to condition attributes $C$, which is the collection of objects from $U$ that can be uniquely classified to different blocks of $U/D$ by means of $C$. The degree of dependency of $D$ on $C$ is denoted as $\gamma_C(D)$, which is calculated by Equation A.2.

$$\gamma_C(D) = \frac{\text{card}(Pos_C(D))}{\text{card}(U)}$$ (A.2)
where $\text{card}(-)$ denotes the cardinality of a set. The significance of a conditional attribute $c$ ($c \in C$) can be obtained by measuring the change of $\gamma_C(D)$ after removing the attribute from the set of considered conditional attributes $C$:

$$\text{Sig}(c) = \gamma_C(D) - \gamma_{C-\{c\}}(D)$$  \hspace{1cm} (A.3)

where $\text{sig}(c)$ is the significance of conditional attribute $c$, which presents the dependency of decision attribute $D$ on condition attribute $c$. A higher change in the dependency indicates how more important is attribute $c$. If the significance is 0, the attribute is dispensable.

With regard to the example shown in Table 3.1, the set of conditional attributes are $C = \{$Availability, Latency, Price$\}$. the indiscernible partitions created by $\text{IND}(C)$ is $U/C = \\{$\{1\}, \{2, 3\}, \{4\}, \{5, 7\}, \{6, 8\}\} \ \text{where the number is the index of the objects in the table}$. Similarly, the indiscernible partitions created by $\text{IND}(D)$ is $U/D = \\{$\{1, 4, 5, 8\}, \{2, 3, 6, 7\}\}$, then the positive region of the partition $U/D$ with respect to $C$ is:

$$\text{Pos}_C(D) = \{\{1\} \cup \{2, 3\} \cup \{4\}\}$$

$$= \{1, 2, 3, 4\}$$

The degree of dependency of decision from attributes $C$ is:

$$\gamma_C(D) = \frac{\text{card}(\{1, 2, 3, 4\})}{\text{card}(\{1, 2, 3, 4, 5, 6, 7, 8\})} = \frac{1}{2}$$

The degree of dependency of $D$ on each attribute $c_i$ is:

$$\gamma_{C-\{c_1\}}(D) = \frac{\text{card}(\{1, 4\})}{\text{card}(\{1, 2, 3, 4, 5, 6, 7, 8\})} = \frac{1}{4}$$

$$\gamma_{C-\{c_2\}}(D) = \frac{\text{card}(\{1, 2, 3, 4\})}{\text{card}(\{1, 2, 3, 4, 5, 6, 7, 8\})} = \frac{1}{2}$$

$$\gamma_{C-\{c_3\}}(D) = \frac{\text{card}(\emptyset)}{\text{card}(\{1, 2, 3, 4, 5, 6, 7, 8\})} = 0$$
and the significance of the three attributes are:

\[ \text{Sig}(c_1) = \gamma_C(D) - \gamma_{C-{c_1}}(D) = \frac{1}{4} \]
\[ \text{Sig}(c_2) = \gamma_C(D) - \gamma_{C-{c_2}}(D) = 0 \]
\[ \text{Sig}(c_3) = \gamma_C(D) - \gamma_{C-{c_3}}(D) = \frac{1}{2} \]

Therefore the most important conditional attribute in Table 3.1 is \( c_3 \) (i.e., price), and the least important conditional attribute is \( c_2 \) (i.e., latency).
Appendix B

Operation Code of Messages

Table B.1: Configuration Message and the Associated Operations

<table>
<thead>
<tr>
<th>Associated Operation</th>
<th>Operation Code</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller Allocation</td>
<td>CIM</td>
<td>The sender advertises a new information about a candidate controller</td>
</tr>
<tr>
<td></td>
<td>CVM</td>
<td>The sender selects the receiver as its controller and verifies the commission with the receiver</td>
</tr>
<tr>
<td>Coordinator Allocation</td>
<td>RIM</td>
<td>The sender is looking for a coordinator</td>
</tr>
<tr>
<td></td>
<td>RIMB</td>
<td>The sender advertises itself as a candidate coordinator</td>
</tr>
<tr>
<td></td>
<td>RVM</td>
<td>The sender selects the receiver as its coordinator and verifies the commission with the receiver</td>
</tr>
<tr>
<td></td>
<td>FRM</td>
<td>The sender has been selected as a new coordinator and broadcasts this information to other coordinators</td>
</tr>
</tbody>
</table>
### Table B.2: Template Registration Message and the Associated Operations

<table>
<thead>
<tr>
<th>Associated Operation</th>
<th>Operation Code</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADV</td>
<td>A provider submits a registration request and the request needs to be forwarded to a coordinator</td>
</tr>
<tr>
<td></td>
<td>TRG</td>
<td>The sender inquires a coordinator about whether a closer gateway is detected</td>
</tr>
<tr>
<td></td>
<td>CREG</td>
<td>The sender inquires a controller about the minimum distance between the service location and its followers</td>
</tr>
<tr>
<td></td>
<td>RREG</td>
<td>The sender requests the receiver to forward the template to the controller that reports the minimum distance</td>
</tr>
<tr>
<td></td>
<td>FREG</td>
<td>The sender requests the receiver to forward the template to the follower closest to the service location</td>
</tr>
<tr>
<td></td>
<td>REG</td>
<td>The sender requests the receiver to register the template</td>
</tr>
<tr>
<td></td>
<td>BAK</td>
<td>The sender requests the receiver to backup the template</td>
</tr>
<tr>
<td></td>
<td>DEL</td>
<td>The sender requests the receiver to delete a backup template</td>
</tr>
</tbody>
</table>

### Table B.3: Negotiation Request Message and the Associated Operations

<table>
<thead>
<tr>
<th>Associated Operation</th>
<th>Operation Code</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SUB</td>
<td>A consumer submits a negotiation request and the request needs to be forwarded to a coordinator</td>
</tr>
<tr>
<td></td>
<td>TREQ</td>
<td>The sender asks the receiver to forward the request to a controller whose range covers the requested location</td>
</tr>
<tr>
<td></td>
<td>CREQ</td>
<td>The sender asks the receiver to search for candidate service providers within the local area</td>
</tr>
<tr>
<td></td>
<td>INS</td>
<td>The sender asks the receiver to negotiate with the candidate service providers</td>
</tr>
</tbody>
</table>
Table B.4: Negotiation Customization Message and the Associated Operations

<table>
<thead>
<tr>
<th>Associated Operation</th>
<th>Operation Code</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negotiation Customization</td>
<td>CUT</td>
<td>The sender tries to customize the negotiation context with the receiver and requests a handshake signal</td>
</tr>
<tr>
<td></td>
<td>OK</td>
<td>Successful handshake signal, which means the opponent is ready to take bids</td>
</tr>
<tr>
<td></td>
<td>REJ</td>
<td>Unsuccessful handshake signal, no further negotiation</td>
</tr>
</tbody>
</table>

Table B.5: Negotiation Message and the Associated Operations

<table>
<thead>
<tr>
<th>Associated Operation</th>
<th>Operation Code</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negotiation</td>
<td>NEG</td>
<td>The sender proposes offers to the receiver</td>
</tr>
<tr>
<td></td>
<td>RES</td>
<td>The sender returns the negotiation result to the receiver</td>
</tr>
</tbody>
</table>

Table B.6: Mobile Entity Locating Message and the Associated Operations

<table>
<thead>
<tr>
<th>Associated Operation</th>
<th>Operation Code</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile Entity Locating</td>
<td>FTC</td>
<td>The sender requests the receiver to forward the message to its coordinator</td>
</tr>
<tr>
<td></td>
<td>FTR</td>
<td>The sender requests the receiver to trigger the entity locating process</td>
</tr>
<tr>
<td></td>
<td>RINQ</td>
<td>The sender requests the receiver to locate the entity in its accessible areas</td>
</tr>
<tr>
<td></td>
<td>CINQ</td>
<td>The sender requests the receiver to locate the entity in its managed area</td>
</tr>
<tr>
<td></td>
<td>FINQ</td>
<td>The sender requests the receiver to test the network connection with an entity</td>
</tr>
</tbody>
</table>
Appendix C

Synonymous Words in Study 1

Table C.1: Synonymous Words List

<table>
<thead>
<tr>
<th>Collection Index</th>
<th>Synonymous Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>price, fare, cost, quotation, expenditure, budget, expense, charge, rate, fee, disbursement, pay</td>
</tr>
<tr>
<td>2</td>
<td>availability, usability, accessibility, workability</td>
</tr>
<tr>
<td>3</td>
<td>reliability, security, robustness, faultless</td>
</tr>
<tr>
<td>4</td>
<td>latency, responsiveness, timeliness</td>
</tr>
<tr>
<td>5</td>
<td>credential, authorization, certification, identity</td>
</tr>
<tr>
<td>6</td>
<td>deviation, variance, accuracy, precision</td>
</tr>
</tbody>
</table>
Bibliography


Proceedings of the 4th international workshop on middleware for service oriented computing, pages 1–6.


processing and large-scale data analytics for smart city applications. In *poster session, European Conference on Networks and Communications*.


[Villalonga et al., 2010] Villalonga, C., Bauer, M., Aguilar, F. L., Huang, V. A., and Stro-


power generators. In 2008 International Symposium on Intelligent Information Technology Application Workshops, pages 859–862. IEEE.


