On the Automated Analysis of WS-Agreement Documents

Applications to the Processes of Creating and Monitoring Agreements

Carlos Müller Cejás
International Doctoral Dissertation

Supervised by
Prof. Dr. Antonio Ruiz Cortés

UNIVERSIDAD DE SEVILLA
On the Automated Analysis of WS–Agreement Documents
Applications to the Processes of Creating and Monitoring Agreements

Thesis written by
Carlos Müller Cejás

and supervised by
Dr. Antonio Ruiz Cortés

Universidad de Sevilla
Dpto. de Lenguajes y Sistemas Informáticos
E.T.S.I. Informática

July 2013
Support: Pre-doctoral scholarship, and scholarships for research visits granted by the Andalusian Government and the University of Seville. Additional support for attending conferences provided by: the European Commission (FEDER); the European Network of Excellence in Software Services and Systems (S-Cube); the Spanish Government under CICYT projects: SETI (TIN2009-07366), and TAPAS (TIN2012–32273); and the Andalusian Government projects ISABEL (TIC-2533) and THEOS (TIC-5906).
El Dr. Don Antonio Ruiz Cortés, Profesor Titular del Área de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla,

**INFORMA**

que D. Carlos Müller Cejás, Ingeniero en Informática, ha realizado bajo mi supervisión el trabajo de investigación correspondiente a su Tesis Doctoral titulada:

*On the Automated Analysis of WS–Agreement Documents, Applications to the Processes of Creating and Monitoring Agreements*

Revisado el presente trabajo, estimo que puede ser presentado al tribunal que ha de juzgarlo, y autorizo la presentación de esta Tesis Doctoral en la Universidad de Sevilla.

Sevilla, Thursday 4th July, 2013

Fdo: Dr. Antonio Ruiz Cortés, Titular de Universidad, Área de Lenguajes y Sistemas Informáticos.
Yo, D. Carlos Müller Cejás con NIF número 75760633J,

DECLARO

mi autoría del trabajo que se presenta en la memoria de esta tesis doctoral que tiene por título:

On the Automated Analysis of WS–Agreement Documents, Applications to the Processes of Creating and Monitoring Agreements

Lo cual firmo,

Sevilla, Thursday 4th July, 2013

Fdo: D. Carlos Müller Cejás, en la Universidad de Sevilla
Universidad de Sevilla

The committee in charge of evaluating the dissertation presented by Carlos Müller Cejás in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Software Engineering, hereby recommends __________________ of this dissertation and awards the author the grade ____________________.

__________________________  ____________________________
Miguel Toro Bonilla             Antonio Brogi
Catedrático de Universidad       Professor
Universidad de Sevilla             University of Pisa, Italy

__________________________  ____________________________
Xavier Franch                   Andreas Metzger
Profesor Titular de Universidad  Academic Senior Councillor
Universitat Politècnica de Catalunya (UPC)           University of Duisburg-Essen,
Germany

__________________________
Rafael Corchuelo
Profesor Titular de Universidad
Universidad de Sevilla

To put record where necessary, we sign minutes in

__________________________
A toda mi familia por su apoyo y amor incondicional en todos los momentos en los buenos y en los no tan buenos.

En especial a mis abuelos Ángel y Charo, Nieves, Mª Pepa y Juan.
Agradecimientos

En primer lugar quiero agradecer a Antonio todo lo que hemos compartido en este tiempo en el que ha sido mucho más que mi director de tesis. Antonio ha sido, desde el principio, esa persona a la que admiras y de la que aprendes, ese consejero que siempre piensa en ti cuando habla contigo, ese compañero y amigo al que le puedes contar absolutamente todo y con el que compartes buenos, malos momentos, y deadlines hasta las 3 de la mañana.

Hay otras dos personas a las que quiero agradecer especialmente su apoyo todos estos años como son Manuel Resinas y Pablo Trinidad. Y lo hago por dos motivos distintos pero que considero igual de importantes. A Manolo me gustaría decirle que para mi es co-director de este trabajo de investigación y le agradezco todas y cada una de las reuniones, conversaciones, y discusiones que hemos tenido porque he aprendido mucho de él en todo este tiempo. Y a Pablo, porque para mi es una persona imposible de catalogar en estos agradecimientos ya que es más que un compañero y más que un amigo que tengo la suerte de conocer desde hace muchos años. Pablo, jamás se me olvidará esa comida de navidad de 2004 en “La Montanera” ;-).

Quiero agradecer las contribuciones que han hecho a este trabajo miembros del grupo ISA como son: Amador, por la ayuda en el estudio y comprensión de la especificación WS–Agreement; Octavio, por su dedicación desde mis inicios en la investigación sobre temporalidad en los SLAs; Sergio, por su ayuda a la hora de diseñar las pruebas de ADA; Pablo Fernández y José Antonio Parejo, por su trabajo en FAST y GDA que me ha servido como escenario de aplicación de nuestra propuesta.

A todos los que he mencionado antes quiero incluirles además en el mismo grupo de compañeros con los que comparto algo más que la profesión, como son: José María, Guti, Jesús, Cristina, David Ruiz, Adela, Bea, Joaquín, David Benavides, José, Ana Belén y Fabricia. Mención especial a Antonio Jurado y Pablo León por su dedicación y esfuerzo en el desarrollo de ADA. Gracias a todos vosotros por compartir vuestro tiempo conmigo en las comidas, charlas y reuniones de pasillo para arreglar el mundo.

No quiero dejar pasar la oportunidad de agradecer los valiosos comentarios que a lo largo del desarrollo de este trabajo de investigación me han hecho destacados investigadores de mi área como son: Andreas Metzger (For your valuable comments on the applicability of our analysis technique and your work in S-Cube, “danke sehr”), Antonio Brogi (For your suggestions and comments about our work, “grazie mille”), Rafael Corchuelo (Por la predisposición que siempre me has mostrado para ayudarme y aconsejarme), Xavier Franch “moltes gràcies per tot”, Miguel Toro (Por tu tiempo para co-
mentarte mi trabajo y tus valiosos comentarios al respecto), Fabio Casati and Vincenzo D’Andrea (For your kind attention in my stay at Trento, ”grazie mille”). Además tengo el honor de tener a casi todos ellos como miembros del tribunal evaluador de este trabajo de investigación.

También quiero agradecer a todos los miembros del departamento de Lenguajes y Sistemas Informáticos la acogida que me brindaron desde que llegué. En especial a mi compañero de despacho Dani, y dos personas con las que he tenido muy buen trato como Rafael Corchuelo y José Antonio Troyano.

No debo olvidarme de mis amigos del grupo GESSI de la UPC, dirigido por Xavi Franch, especialmente Marc Oriol y Jordi Marco a los que estimo como amigos, además de como colegas de profesión.

A mis amigos y familiares. En especial a mis padres Jorge y Lourdes, a mis hermanos, Jorge y Lourdes, a Anamari, y sobre todo a las dos personitas que más alegría me transmiten: Julia y Natalia.

Por último quiero agradecer a este documento de tesis la posibilidad de dejar por escrito estos agradecimientos a toda esta gente que tanto lo merece.
First of all, thank you Antonio, thanks to be not my thesis supervisor, but a "superadvisor". Antonio, I admire your perfect combination of humanity and leadership.

Secondly, I need to express my gratitude to two great persons: Manuel and Pablo Trinidad. Manuel, I consider you as a co-supervisor of this work and I really appreciate our meetings in which I have equally enjoyed and learnt. Pablo, you are more than a friend and more than a colleague. We have enjoyed together a 20 years friendship, that can be extended in, at least, a hundred years more.

Thirdly, I would like to mention some members of the ISA research group that have contributed somehow in this PhD. Amador Durán, for your valuable help in the study and understanding of the WS–Agreement specification. Octavio, for your dedication from my first steps in the study of time-aware SLAs. Sergio, for your helpful comments and review to enhance the verification of our reference implementation. Pablo Fernández and José Antonio, for your work in FAST and GDA that was really useful as application scenario for our proposal.

Fourthly, I thank each and every member of the ISA research group to share with me more than a career: José María, Guti, Jesús, Cristina, David Ruiz, Adela, Bea, Joaquín, David Benavides, José, Ana Belén y Fabricia. An special mention to Antonio Jurado and Pablo León for his dedication and effort in the development of our tooling support. Thank you all to spend, and occasionally waste, our time together at lunches, parties, conventional and hall meetings to fix the world.

It is the turn to leave a written proof of my gratitude for the valuable comments received from distinguished researches such as: Andreas Metzger, for your valuable comments on the applicability of our analysis technique and your work in S-Cube, “danke sehr”. Antonio Brogi, for your suggestions and comments about our work, “grazie mille”. Rafael Corchuelo, for your availability and helpful advises. Xavier Franch, for your dedication in my stay at the UPC, “moltes gràcies per tot”. Miguel Toro, for your dedication and valuable comments for this PhD. Fabio Casati and Vincenzo D’Andrea, for your kind attention in my stay at Trento, “grazie mille”. I have the honour to have most of you as members of the evaluation committee of my thesis.

I would also like to thanks each and every member of the LSI department of the University of Seville its welcome and comradeship from my first day. Specially to Dani, my Office partner, and two persons like Rafael Corchuelo and José Antonio Troyano.
I may remember in this acknowledgements the members of the GESSI research group of the UPC leads by Xavier Franch. Specially to Marc Oriol and Jordi Marco, I consider you not only as good colleagues, but also friends.

Thanks to my family and friends. Specially to my parents: Jorge and Lourdes, my brother Jorge, my sister Lourdes, Anamari, and particularly to the two little persons that make me always smile: Julia and Natalia.

Finally, thank this dissertation for the chance to put these acknowledgements in a document.
A medida que los SLAs empiezan a ser necesarios en servicios y aplicaciones usadas por el gran público y en ambientes XaaS donde existen penalizaciones por su incumplimiento, aumenta la demanda y la complejidad de los sistemas que dan soporte al ciclo de vida de los SLAs, en adelante, soluciones dirigidas por SLAs. Tras revisar el estado del arte, hemos concluido que existe un amplio espacio de mejora en las tecnologías para desarrollar soluciones dirigidas por SLAs con dos dimensiones claramente diferenciables: (1) los lenguajes para especificar SLAs, y (2) las técnicas para extraer información útil sobre los SLAs, que en adelante llamaremos técnicas de análisis.

En cuanto a los lenguajes de especificación, las mejoras pueden venir de conseguir lenguajes i) independientes del dominio, ii) conformes a recomendaciones estándar, iii) con la expresividad necesaria para describir de manera sencilla los objetivos de nivel de servicio (SLOs), las restricciones y los períodos de validez de sus diferentes elementos, iv) que dispongan de criterios para verificar propiedades básicas tales como la consistencia, la conformidad y la optimalidad y, llegado el caso, explicar por qué no es posible verificarlas de un modo que permita al usuario un fácil diagnóstico y reparación.

En cuanto a las técnicas de análisis propuestas hasta la fecha, encontramos que presentan algunas de las carencias habituales de las tecnologías emergentes, por lo que la mejora puede venir de: desarrollar implementaciones de referencia lo suficientemente completas, con un diseño orientado a su reutilización, abiertas a ser extendidas de manera efectiva y, por último, disponer de un interfaz de usuario fácil de usar.

El objetivo global de esta tesis es mejorar el soporte actualmente existente para desarrollar soluciones dirigidas por SLAs tomando como referencia las mejoras anteriormente identificadas, centrando su estudio en la recomendación WS–Agreement, propuesta por la Open Grid Forum para especificar y crear acuerdos de nivel servicio.

Sus principales contribuciones son un lenguaje de especificación WS–Agreement compliant que hemos denominado iAgree, y un entorno para desarrollar soluciones dirigidas por SLAs que hemos denominado IDEAS que tiene como componente principal un analizador de documentos iAgree que hemos denominado ADA. Como contribuciones menores se han desarrollado soluciones para mejorar el soporte disponible actualmente para la creación y monitorización de acuerdos de nivel de servicios.

La clave de nuestra propuesta para mejorar el lenguaje de especificación de acuerdos pasa por definir lo que hemos dado a llamar un configuración WS–Agreement
(WSAC) que dispone de los sublenguajes necesarios para describir las distintas partes de un documento de acuerdo. En cuanto a nuestra propuesta de mejora para las técnicas de análisis, la clave ha sido organizar dichas técnicas en un catálogo de operaciones básicas de análisis que pueden ser combinadas para soportar soluciones dirigidas por SLAs más complejas. La aplicabilidad de nuestros resultados se restringe a aquellos SLAs que pueden interpretarse como problemas de satisfacción de restricciones, lo cual, en base a nuestra experiencia, es suficiente para abordar muchas situaciones de gran interés.
The need for Service Level Agreements (SLAs) to supervise the services consumption is increasing in business applications and Cloud scenarios due to the violation of SLA terms may carry penalties to the parties. Such a need for SLAs boosts the importance and complexity of SLA lifecycle supporting systems, *SLA-driven solutions* from now on. After a study of the existing literature we conclude that these SLA-driven solutions can be significantly enhanced in both: (1) the SLA specification languages, and (2) the techniques to extract useful information from the SLAs, *analysis techniques*, in advance.

In the one hand, we suggest that existing SLA specification languages can be improved by turning them into: i) domain-independent, ii) standard-recommendation-compliant, and iii) expressive-enough to ease the description of Service Level Objectives (SLOs), constraints, and validity periods for their comprised elements. In addition, the specification languages must define validity criteria for checking that supported SLAs satisfy some basic properties such as the consistency, and the compliance between SLA documents. Furthermore, explanations must be provided when the SLAs do not satisfy such basic properties. On the other hand, the proposed analysis techniques include some drawbacks that can be found in emerging techniques. To overcome such drawbacks there must be developed: a) fully-functional reference implementations, b) techniques with a reuse-oriented design, c) effective extension mechanisms, and d) user-friendly interfaces.

The main goal of this dissertation is to improve the current support to develop SLA-driven solutions by adopting the WS–Agreement standard recommendation and considering the aforementioned suggestions for improvement.

The main thesis contributions are a WS–Agreement-compliant language called *iAgree*, and an *Integrated Development Environment for SLA-driven Solutions* (IDEAS) whose main component is an *iAgree Document Analyser* (ADA). As minor contributions, two solutions have been developed to improve the existing tooling support to create and monitor SLAs.

The cornerstone of our proposal to improve the SLA specification languages has been the definition of the WS–Agreement Configurations that comprise the sublanguages needed to describe the different parts of WS–Agreement documents. Regarding our proposal to improve the analysis techniques, the key has been the organization of such techniques in a catalogue of basic analysis operations that can be combined to support more advanced SLA-driven solutions. The applicability of our results is limited to those SLAs that can be translated to a constraint satisfaction problem, that is
enough to support real-world SLAs, in our experience.
CONTENTS

I  Preface 1

1  Introduction 3
   1.1  Research Context .............................................. 4
   1.2  Thesis Goals .................................................. 6
   1.3  Contributions ................................................ 7
      1.3.1  Contributions regarding SLA Specification ............. 7
      1.3.2  Contributions regarding Automated Analysis ........... 9
      1.3.3  Contributions regarding SLA-driven Solutions Support. 9
   1.4  Thesis Context ............................................... 10
   1.5  Publications ................................................ 11
      1.5.1  Publications Supporting this Dissertation .............. 11
      1.5.2  Further Publications .................................... 13
   1.6  Outline ..................................................... 14

II  Background Information 17

2  WS–Agreement Documents Specification 19
   2.1  Introduction .................................................. 20
   2.2  Documents Structure .......................................... 21
      2.2.1  Service Terms ............................................. 21
      2.2.2  Guarantee Terms ......................................... 25
III Our Proposal

5 Motivation

5.1 Introduction .......................................................... 66
5.2 Problems Related to the SLA Specification .................. 66
  5.2.1 Lack of Full-WS–Agreement Support ....................... 67
  5.2.2 Lack of Support for Suitable Predicates .................. 69
  5.2.3 Lack of Domain-independent WSACs ...................... 69
  5.2.4 Lack of Time-aware WSACs ................................. 69
  5.2.5 Lack of Human-readable WSACs ......................... 70
  5.2.6 Lack of Validity Criteria .................................. 70
5.3 Problems Related to the Automated Analysis ............... 71
  5.3.1 Lack of a Fully-functional Reference Implementation .. 71
  5.3.2 Lack of a Reuse-oriented Design ......................... 71
  5.3.3 Lack of Publicly-available User-friendly Interfaces ...... 72
5.4 Problems Related to the SLA-driven Solutions Support ... 72
  5.4.1 Lack of Validation at Agreement Creation and Monitoring . 72
  5.4.2 Lack of an Integrated Development Environment ....... 72
5.5 Analysis of Current Solutions .................................... 73
  5.5.1 Solutions Dealing with SLA Specification ................ 73
  5.5.2 Solutions Dealing with Automated Analysis and SLA-driven Solutions Support ......................... 76
5.6 Overview of Our Solution ........................................ 84
5.7 Summary .............................................................. 86

6 iAgree. an Intermediate ws-AGREement configuration ...... 87
  6.1 Introduction ........................................................ 88
  6.2 Agreement Context Information .............................. 91
    6.2.1 WS–Agreement Protocol Information ................... 91
<table>
<thead>
<tr>
<th>6.2.2</th>
<th>Validity Periods</th>
<th>92</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.3</td>
<td>Metrics Catalogues</td>
<td>93</td>
</tr>
<tr>
<td>6.3</td>
<td>Service Terms</td>
<td>96</td>
</tr>
<tr>
<td>6.4</td>
<td>Guarantee Terms</td>
<td>98</td>
</tr>
<tr>
<td>6.5</td>
<td>Agreement Variants</td>
<td>100</td>
</tr>
<tr>
<td>6.6</td>
<td>Agreement Creation Constraints</td>
<td>104</td>
</tr>
<tr>
<td>6.7</td>
<td>Validity Criteria</td>
<td>105</td>
</tr>
<tr>
<td>6.7.1</td>
<td>Conflicts in iAgree Documents</td>
<td>106</td>
</tr>
<tr>
<td>6.7.2</td>
<td>Conflicts in Time-aware iAgree Documents</td>
<td>108</td>
</tr>
<tr>
<td>6.7.3</td>
<td>Scope of conflicts</td>
<td>111</td>
</tr>
<tr>
<td>6.7.4</td>
<td>Template Compliance</td>
<td>112</td>
</tr>
<tr>
<td>6.7.5</td>
<td>SLA Unfulfillment</td>
<td>113</td>
</tr>
<tr>
<td>6.8</td>
<td>Summary</td>
<td>115</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7</th>
<th>Automated Analysis of iAgree Documents</th>
<th>119</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>120</td>
</tr>
<tr>
<td>7.2</td>
<td>iAgree Formal Semantics</td>
<td>120</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Primary goal</td>
<td>120</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Mapping iAgree documents</td>
<td>122</td>
</tr>
<tr>
<td>7.3</td>
<td>Automated Analysis of Core Operations</td>
<td>124</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Geometrical interpretation of a CSP</td>
<td>124</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Checking and Explaining Inconsistencies</td>
<td>124</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Checking and Explaining Non-Compliant terms</td>
<td>126</td>
</tr>
<tr>
<td>7.4</td>
<td>Automated Checking of Validity Criteria</td>
<td>128</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Checking and Explaining Conditionally Inconsistent terms</td>
<td>128</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Checking and Explaining Dead terms</td>
<td>129</td>
</tr>
<tr>
<td>7.4.3</td>
<td>Checking and Explaining if a Document is Valid</td>
<td>130</td>
</tr>
<tr>
<td>7.5</td>
<td>Normalization Operations</td>
<td>131</td>
</tr>
</tbody>
</table>
CONTENTS

7.5.1 Variants Normalization ........................................ 131
7.5.2 Temporal Normalisation ....................................... 132
7.6 Summary ........................................................... 133

8 SLA-driven Solutions Support .................................. 135
  8.1 Introduction ...................................................... 136
  8.2 Supporting SLA-driven Solutions ............................... 138
    8.2.1 Agreement Creation SLA-driven Solutions .............. 139
    8.2.2 Agreement Monitoring SLA-driven Solutions .......... 147
  8.3 ADA Catalogue of Analysis Operations ..................... 150
    8.3.1 iAgree Validity Criteria Checkers ..................... 151
    8.3.2 Choco Adapter ............................................ 152
    8.3.3 iAgree Normalisation and CSP Mapping ............... 153
    8.3.4 iAgree Validator .......................................... 153
    8.3.5 Fulfillment Analyser .................................... 154
    8.3.6 iAgree Document Manager ............................... 154
    8.3.7 iAgree Search Engine .................................. 156
  8.4 iAgree Studio .................................................. 156
  8.5 Summary ........................................................ 159

IV Final Remarks ....................................................... 163

9 Conclusions and Future Work .................................. 165
  9.1 Conclusions .................................................... 166
  9.2 Discussion, limitations and extensions .................... 168

V Appendices .......................................................... 171
  A iAgree Abstract Syntax ....................................... 173
# List of Figures

1.1 Overview of the contributions of this doctoral thesis .......................... 8

2.1 UML metamodel of the WS–Agreement schema ................................. 22

2.2 Template of AmazonS3 service scenario in iAgree (see Chapter §6) .... 24

2.3 WS–Agreement structure and sublanguages ............................... 27

2.4 An example of nested term compositors, inspired in AmazonS3 scenario 30

2.5 Enumeration of all variants defined by the TCs in document of Fig. §2.4 30

2.6 Example of a non-compliant agreement offer .................................... 32

2.7 Examples of disjoint, non-disjoint, and periodical validity periods .... 36

3.1 A typical WS–Agreement interaction to create agreements .............. 42

3.2 State model of the SLAs states ................................................. 44

3.3 State model of the guarantee terms states .................................. 44

3.4 WSAG4J architecture provided by the authors at the project homepage 45

4.1 Online simple monthly calculator for AmazonS3 storage service .... 50

4.2 The operation detects an inconsistency in the input ...................... 52

4.3 The operation detects a non-compliance between the inputs ........ 52

4.4 The operation detects an SLA violation between the inputs ........... 53

4.5 The operation explains an SLA violation between the inputs ........... 54

5.1 Optional terms sets described by using an optional terms compositor . 67

5.2 Alternative terms sets described by using an alternative terms compositor 68

5.3 Excerpt of an AmazonS3 template with scoped terms ............... 68

5.4 Compliance between templates and offers .................................... 77
5.5 An example of inconsistency explanation .......................... 79
5.6 Explanation of conflicts that do not wholly invalidate documents . . . . 80
5.7 Explanation of non-compliance between WS–Agreement documents . . 82
5.8 Overview of the contributions of this doctoral thesis .................... 86
6.1 iAgree sublanguages completing the WS–Agreement schema ............. 89
6.2 Excerpt of AmazonS3 service scenario template in iAgree ................ 90
6.3 Schema of the context information supported by iAgree .................. 91
6.4 Protocol information of the AmazonS3 template in iAgree ................. 92
6.5 Protocol information of an offer to be sent to Amazon in iAgree ........... 92
6.6 Abstract syntax and example of validity periods in iAgree ................. 92
6.7 VPs included in the context information of AmazonS3 in iAgree ........... 93
6.8 Metrics included in the context information of AmazonS3 in iAgree ....... 94
6.9 General iAgree metrics catalogue ....................................... 94
6.10 Amazon metrics catalogue ................................................ 95
6.11 Schema of the service terms supported by iAgree ....................... 96
6.12 Abstract syntax and example of SDTs in iAgree ........................... 97
6.13 Service Terms of AmazonS3 template in iAgree .......................... 98
6.14 Schema of the guarantee terms supported by iAgree ..................... 99
6.15 Guarantee Terms of AmazonS3 template in iAgree ....................... 102
6.16 Agreement variants inspired in the AmazonS3 scenario, in iAgree ....... 103
6.17 Schema of the creation constraints supported by iAgree ................. 104
6.18 Example of creation constraints in iAgree ............................... 105
6.19 Creation Constraints of AmazonS3 template in iAgree .................... 106
6.20 Document with an inconsistency between a GT and a creation constraint 107
6.21 Document with a dead GT caused by another GT ....................... 107
6.22 Document with a conditionally inconsistent GT ........................... 108
6.23 Document with an inconsistent GT and CC whose VPs overlap ........... 109
6.24 Document with undercovering of the global period . . . . . . . . . . . . 110
6.25 Document with overcovering of the global period . . . . . . . . . . . . . 110
6.26 Template with a partial dead term (G9.2) by the creation constraint . . 111
6.27 Agreement offer with a partial inconsistency . . . . . . . . . . . . . . . . 112
6.28 Agreement offer compliant with template of Figure §6.30 . . . . . . . . . 113
6.29 Agreement offer non-compliant with template of Figure §6.30 . . . . . . 113
6.30 Excerpt of the Template of AmazonS3 service scenario of Figure §2.2 . . 114
6.31 AmazonS3 SLA with GT affecting different service operations . . . . . . 116

7.1 Outline of our approach to automate the analysis of iAgree documents . 121
7.2 Example of a CSP mapping from Template of Fig. §6.2 . . . . . . . . . . . 124
7.3 Graphical representation for CSPs . . . . . . . . . . . . . . . . . . . . . . 125
7.4 Checking and explaining inconsistencies of Template of Figure §6.20 . . . 125
7.5 Examples for the non-compliance between Figures §6.30 and §6.29 . . . 127
7.6 Checking and explaining the cond. inconsistent term of Figure §6.22 . . . 129
7.7 Checking and explaining the dead term of Template of Figure §6.21 . . . 130
7.8 Excerpt from the example of nested term compositors, of Figure §6.16 . . 132
7.9 Applying temporal normalisation to a document with three GTs . . . . . 133

8.1 Layers of IDEAS development environment. . . . . . . . . . . . . . . . . 138
8.2 Template to test WSAG4J with some intentional inconsistencies . . . . 140
8.3 Explanations for the document of Figure 8.1 . . . . . . . . . . . . . . . . . 140
8.4 Offer to test WSAG4J with some intentional inconsistencies . . . . . . . 141
8.5 Explanations for the document of Figure §8.4 . . . . . . . . . . . . . . . . 141
8.6 SOA-Governance system architecture . . . . . . . . . . . . . . . . . . . . . 143
8.7 Template of Node_M to provide ADAaaS as a Service for Node_1 . . . . 144
8.8 Template modelled by the students for PayPal service with a dead term 146
8.9 IDEAS components diagram. . . . . . . . . . . . . . . . . . . . . . . . . . 150
8.10 iAgree Studio user interface navigability. . . . . . . . . . . . . . . . . . . 160
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.11</td>
<td>Edition capabilities of the iAgree front-end to try ADA</td>
<td>161</td>
</tr>
<tr>
<td>8.12</td>
<td>An iAgree scenario launched to explain conflicts if they are found.</td>
<td>162</td>
</tr>
<tr>
<td>8.13</td>
<td>An iAgree scenario to explain compliance problems.</td>
<td>162</td>
</tr>
<tr>
<td>9.1</td>
<td>Technique to support time-aware iAgree documents in ADA.</td>
<td>169</td>
</tr>
<tr>
<td>9.2</td>
<td>iAgree editor with analysis support for time-aware documents.</td>
<td>170</td>
</tr>
<tr>
<td>B.1</td>
<td>Excerpt of AmazonS3 service scenario template in <em>iAgree</em></td>
<td>182</td>
</tr>
<tr>
<td>D.1</td>
<td>Document with a dead GT caused by another GT</td>
<td>189</td>
</tr>
<tr>
<td>D.2</td>
<td>Document with a conditionally inconsistent GT</td>
<td>190</td>
</tr>
<tr>
<td>D.3</td>
<td>Excerpt of the Template of AmazonS3 service scenario of Figure §2.2</td>
<td>192</td>
</tr>
<tr>
<td>D.4</td>
<td>Excerpt of the CSP generated from template of Figure §D.3</td>
<td>193</td>
</tr>
<tr>
<td>D.5</td>
<td>Agreement offer non-compliant with template of Figure §6.30</td>
<td>193</td>
</tr>
<tr>
<td>D.6</td>
<td>CSP generated from agreement offer of Figure §D.5</td>
<td>194</td>
</tr>
<tr>
<td>F.1</td>
<td>SALMonADA components diagram reusing ADA Core components</td>
<td>203</td>
</tr>
<tr>
<td>G.1</td>
<td>Template got from error guessing technique</td>
<td>222</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Sublanguages supported by implicit WSACs of related approaches . . . 28
2.2 Time-Awareness of current SLA & web services specifications ........ 38
4.1 ResponseTime<100 in getRate operation using event-calculus [63] ... 54
4.2 Support provided by the existing analysis techniques .................. 61
5.1 Support for Sublanguages and Problems of the implicit WSACs proposed. 74
5.2 Support provided by the existing analysis techniques .................. 78
7.1 Mapping from iAgree to CSP ........................................ 123
8.1 Catalogue of analysis operations ..................................... 151
B.1 WS–Agreement schema to iAgree mapping: Metadata and Compositors 180
B.2 WS–Agreement schema to iAgree mapping: Terms and Creation Constraints .................................................. 181
G.1 Inputs and outputs for ADA analysis operations addressed in the suite . 220
G.2 Number of inputs got from each testing technique for the ADA analysis operations ................................................. 224
G.3 Four of the test cases included in the suite ............................... 225
In this chapter, we introduce the research context and we outline the work performed. In Section §1.1, we describe both, the concepts of the research context which frame the scope of our work, and we mention the drawbacks of the related work that are required to understand the contributions of the dissertation. In Section §1.2 we state the goals we face in this dissertation and the hypothesis that supports our research. In Section §1.3, we present a summary of our contributions. Section §1.4 introduces a brief description of the context in which our work has been developed. Section §1.5 enumerates the publications that support this PhD. Finally, in Section §1.6 we describe how the content of this dissertation is organised.
CHAPTER 1. INTRODUCTION

1.1 Research Context

It is commonly accepted that Service Level Agreements (SLAs) establish both, the service functionality and guarantees on monitorable service properties, whose values may vary through the SLA lifetime. Such monitorable properties, which are commonly used to describe non-functional properties (a.k.a. quality aspects) [102], may be either domain-independent such as the response time and the time to failure, or domain-dependent such as the read and write requests for storage services, or the memory size, number and the type of CPUs for grid computing services.

As the increasing use of SLAs in commercial services and applications in which the violation of SLA terms may imply a penalty for the parties, the complexity and demand of systems supporting the SLA lifecycle also increases. We refer to such systems as SLA-driven solutions, from now on. Our study of the different approaches reveals that current techniques to develop SLA-driven solutions can be improved in some aspects such as the specification languages, the techniques to extract useful information from the SLAs, analysis techniques from now on, or the tooling support.

Regarding the SLA specification languages, a variety of approaches can be found in the literature to describe SLAs by means of natural language in agreements intended to be used only by humans; formal languages with the aim of analyzing some properties of the SLAs [17, 26, 116]; and XML documents in standardization efforts aimed at making the interoperability between consumers and providers easier [10, 37, 56, 97]. Among these standardization efforts WS–Agreement [10] stands out. WS–Agreement is a proposed recommendation of the Open Grid Forum that provides an overall structure to describe and serialize agreements, and a protocol for creating and monitoring them. The agreement creation is based on the so-called templates and agreement offers, which jointly the created agreements we refer as WS–Agreement documents in this dissertation.

The overall structure for WS–Agreement documents is proposed by means of an XML-based schema with up to eight extension points that makes possible to use the recommendation in a virtually unlimited number of different ways [11]. Such extension points must be filled by means of sublanguages, for which the recommendation does not provide any default sublanguage, nor imposes additional constraints on their expressiveness. Thus, WS–Agreement does not provide a fully-fledged language for specifying WS–Agreement documents. Furthermore, there exist an unbounded number of combinations of sublanguages that may conform one of such fully-fledged languages. To refer to any of these combinations we have coined the expression WS–Agreement Configuration or WSAC for short.

\footnote{From now on, we will use SLA and agreement as synonyms in this dissertation.}

\footnote{Such as Amazon (aws.amazon.com/s3-sla) or Google (www.google.com/apps/intl/en/terms/sla.html)}

\footnote{www.gridforum.org}
A WSAC must include sublanguages for the two mandatory extension points of the WS–Agreement schema, namely the service description terms (SDTs), and the Service Level Objectives (SLOs). SDTs define the service functionality, and SLOs the guarantees on the monitorable service properties. Examples of these sublanguages can be found in the literature. For example, the WSAC provided by WSAG4J framework\(^4\) uses Job Submission Description Language (JSDL) \(^5\) for the SDTs and Java Expression Language (JEXL) \(^5\) for the SLOs; and the WSAC by Oldham et al.\(^{[100]}\) called SWAPS, uses WSDL-S/OWL for the SDTs, and the WSLA expression language by Keller and Ludwig\(^{[56]}\) for the SLOs.

In this dissertation we focus on some issues related to the specification, the analysis and the tooling support for WSACs. Regarding specification, some drawbacks appear when we try to write WS–Agreement documents in real-world scenarios are following outlined.

- Most of the proposed WSACs do not support WS–Agreement elements such as the qualifying conditions (QCs) and term compositors (TCs), that allow the description of conditional and optional terms, respectively. QCs express a precondition under which the SLO holds. For instance, in the Amazon Simple Storage Service (AmazonS3) SLA, the service response time may depend on the interface protocol selected by the user to consume the service. Thus, if the selected interface is SOAP the response time could be higher than whether the selected interface is REST. Such an interface selection would be the QC that enables a given guarantee on the response time. In turn, TCs are needed to compose the optional customer support facilities offered by Amazon such as the access to technical support that can be performed online or by phone.

- They are not expressive enough to specify arithmetic-logic SLOs, QCs, and constraints. For instance, the AmazonS3 SLA assures a data durability\(^6\) of 99.99 % for reduced redundancy storage (RRS) services. Furthermore, the price policy considers both, the RRS, and the storage size required by the user. These conditions that could be expressed as \(RRS = \text{true} \iff (\text{DataDurability} >= 99.99\%) \text{ AND } (\text{StoragePrice} = 0.05 \times \text{StorageSize})\) are not not supported by most of the existing approaches.

- They do not provide validity criteria so there is no way to guarantee that a document has no semantic errors such as dead terms, inconsistent terms.

- They do not allow specifying validity periods for each single SLO.

\(^{4}\)Developed by GRAAP group http://packcs-e0.scai.fraunhofer.de/wsag4j/index.html.  
\(^{5}\)http://commons.apache.org/jexl/  
\(^{6}\)AmazonS3 provides a highly durable storage infrastructure designed for mission-critical and primary data storage. Thus, systematic data integrity checks, and checksums on all network traffic are performed to detect corruption of data packets when storing or retrieving data.
Regarding analysis, Martín-Díaz et al. pioneered [65, 66, 67, 116] the organization of such techniques in the so-called analysis operations to determine: (1) if a document is consistent (i.e. it does not include semantics contradictions between the terms), (2) if an agreement offer is compliant with a template, and (3) the optimal SLA that can be obtained from a template and a set of compliant offers [116]. Some approaches can be found in the literature providing techniques to detect inconsistencies that affect the whole agreement [17, 26, 116]; or the non-compliance between agreement documents [100]. Other proposals go further and they also report some explanations for inconsistencies at runtime [63, 124].

In this dissertation we focus on adding new analysis operations related to the diagnosis of semantic errors and non-compliance situations. In this sense, current semantic errors are limited to basic inconsistencies and therefore they do not take into account the conflicts that arise when qualifying conditions and term compositors. Last, but not least, hoping avoid reinventing the wheel, we also focus in how to introduce best practices from software design to develop new analysis operations not from scratch, but from a commonly accepted operations catalogue. For instance, the detection of SLA violations at runtime can be formulated by using the operation that allows to check the consistency of an agreement.

Finally, regarding the tooling support, probably the main obstacle at this point is that the current tooling support is quite immature. To the best of our knowledge, there not exist tools for specifying and validating WS–Agreement documents, although there are a plethora of approaches focus on solving numerous SLA-related problems for simpler models than WS–Agreement. In this sense, solving become WS–Agreement-awareness these problems will allow devising a more useful tooling support.

1.2 Thesis Goals

We have studied the state of the art with regard to all the issues described in the previous section, and we have identified a set of problems that constitute the goals to be achieved in this thesis, namely:

1. To know and analyze to what extent current proposals support the WS–Agreement specification.

2. To provide a fully-fledged language to specify WS–Agreement documents supporting all the elements of the specification.

3. To define a set of validity criteria for WS–Agreement documents.

4. To know and analyze the analysis techniques currently used in SLA-driven solutions.
1.3. CONTRIBUTIONS

5. To design and develop a set of analysis operations to check the validity criteria of WS–Agreement documents.

6. To identify the usefulness of analysis operations in supporting the creation and monitoring of agreements.

These goals can be summarised in a main goal stated as follows:

**Dissertation Goal**

*To improve the current support to develop SLA-driven solutions focused in WS–Agreement recommendation.*

1.3 CONTRIBUTIONS

Our approach to address the aforementioned goals is highly inspired in the solutions proposed for the automated analysis in SLAs [68, 115, 116], features models [12, 132], and Business Process Management (BPM) [1, 20]. In all these cases, the general approach was to extend existing models; to propose highly expressive notations, in order to represent such extensions; and to interpret analysis operations as queries on a formal representation (usually by means of Constraint Satisfaction Problems (CSP) [135]) of such models and their instances. The overall result is conceptually called *Integrated Development Environment for SLA-driven Solutions (IDEAS)*. The specific contributions of this thesis are depicted in Figure §1.1 and summarized below.

1.3.1 Contributions regarding SLA Specification

Our first contribution (C1 in figure §1.1) is the development of a new WSAC called *Intermediate ws–AGREEment configuration (iAgree)*, which allows us to specify the three kinds of WS–Agreement documents: templates, offers and agreements. The main features of iAgree are:

1. Highly WS–Agreement-compliant: iAgree supports all the elements mentioned in WS–Agreement specification, including QCs, TCs, terms scope, rewards and penalties⁷.

⁷Semantic errors related to rewards and penalties are not still included.
CHAPTER 1. INTRODUCTION

Figure 1.1: Overview of the contributions of this doctoral thesis

2. High and extensible expressiveness: iAgree allows specifying SLOs, QCs, and CCs by using a very expressive expression language. iAgree has been designed following the Semantics Priority Principle by Hofstede and Proper in [51], thus iAgree semantics assumes the existence of an ideal underlying CSP solver able to deal with any kind of constraint. Current version of iAgree uses Choco Solver and according to the criteria we have used to evaluate similar approaches from literature, we can claim that it is very expressive.

3. Domain-independent: two reasons endow iAgree with a very-high domain-independence namely, the aforementioned extensibility and expressiveness; and the use of externally defined catalogues of metrics whose application domains and operators are just limited by the underlying solver.

4. Time-aware: iAgree allows describing validity periods of the different agreement elements in a very flexible way and independently for each SLO and creation constraint.

5. Human-readable: iAgree provides a human-readable syntax.

The second contribution of this thesis (C2 in figure §1.1) is the identification of a set of validity criteria for iAgree documents, which allows us to identify up to 4 kind 

---

8See Section §8.3 to know the expression language currently supported
of semantic errors: inconsistencies, dead terms, conditionally inconsistent terms, and non-compliant terms. As far as we know, we are pioneers at defining semantic errors for WSACs.

1.3.2 Contributions regarding Automated Analysis

The first contribution (C3 in figure §1.1) aimed at providing support for automated analysis, consists of the definition of formal semantics for iAgree. This provides the iAgree constructions with precise meaning, and eases the automated extraction of information from documents written in iAgree (a.k.a. iAgree Documents or simply documents for short). In particular, iAgree semantics is based on CSPs.

Next, relying on iAgree semantics, we have developed an approach to automatically check and explain violations of iAgree validity criteria, organized in four core operations, and some additional analysis operations (C4 in figure §1.1). A reference implementation based on CSPs has been developed to support the execution of all the operations. Both groups of operations have been included in a catalogue of analysis operations ready to be used in SLA-driven solutions.

1.3.3 Contributions regarding SLA-driven Solutions Support.

The first contribution aimed at extending to well known SLA-driven solutions to become them WS–Agreement compliant (C5 in figure §1.1): SALMon and WSAG4J. SALMon is a monitoring platform which main goal has been to gather the Quality Of Service (QoS) of web services and check simple conditions included within terms of ad-hoc SLAs. We extended SALMon jointly with its creators [89] in order to support: (1) the analysis of WS–Agreement-compliant agreements with expressive SLOs, (2) to check and explain agreement violations, and (3) to put it publicly-available with an user-friendly interface. This extension was carried out by using our operations catalogue.

WSAG4J tool provides an implementation of the WS–Agreement protocol. We extended WSAG4J jointly with its creators [96] in order to endow it with analysis facilities in a such a way that templates, offers and agreements can be validated before, during and after the agreement creation. This extension was also carried out by using our operations catalogue. In addition, in this case we also were benefited of having a domain-independent language because it allowed to translate into iAgree without losing information, the XML-based Job Submission Description Language (JSDL)[5] used in WSAG4J for the examples.

The second contribution aimed at providing support for both human and software clients that are involved in the development of SLA-driven solutions (C6 in figure §1.1). In this case, we provide a development environment that comprises all tools developed in this dissertation that we coined IDEAS. The contribution is not the tool development
itself, but the concept of a novel Integrated Development Environment for SLA-driven Solutions (IDEAS), since to the best of our knowledge there is not exist any similar one. IDEAS provides a publicly-available user-friendly front-end which makes possible: (1) to edit documents assuring they are valid, (2) to analyse some properties appealing for final and technical users. Such an environment is also available to be used by software clients through a triple distribution model: as a Java library, as an OSGi\(^9\) service, and as a web service. IDEAS is a suite that has been partially validated in the development of advanced SLA-driven solutions to create and monitor agreements.

1.4 Thesis Context

This thesis has been developed in the context of the research group Applied Software Engineering (Ingeniería del Software Aplicada-ISA) of the Universidad de Sevilla. The work that has made this thesis development possible is in the context of the following research projects and networks:

- **WEB-FACTORIES.** Fábricas Software para Sistemas con Arquitectura Orientada a Servicios Web (CYCIT project). At the beginning of this research work, in the context of this project, we focused in the study of WS–Agreement specification. Specifically, we studied how to apply the time-awareness of web service matchmaking proposed by Martín-Díaz et al. [67], into WS–Agreement documents.

- **ISABEL.** Ingeniería de Sistemas Abiertos Basada en Líneas de productos. Proyecto de excelencia de la Junta de Andalucía. Most of the research work included in the current dissertation have been developed in the context of the ISABEL project. Specifically, the study of automated analysis of WS–Agreement documents to check and explain: document inconsistencies and the non-compliance between documents [71, 72], as well as the development of the first release of ADA (Agreement Document Analyzer), the core of IDEAS.

- **S-Cube.** the European Network of Excellence in Software Services and Systems, funded by the European Commission. Thanks to our participation in this network, we validated the usefulness of ADA to check and explain the unfulfillment of SLAs at runtime by means of SALMonADA service-based application [85].

- **SETI.** reSearching on intElligent Tools for the Internet of services (CYCIT project). In this project we performed the final enhancement of ADA and the prototype web application, into IDEAS. In this context, previous proposals were improved: (1) to explain SLA inconsistencies, by identifying new kinds of conflicts [90]; and (2) to check and explain the unfulfillment of SLAs [89].

\(^9\)www.osgi.org
1.5 Publications

Our research work has followed a clear path, allowing us to publish our contributions in mainstream workshops, conferences, and journals. In the following sections we point out the main contributions of each publication and its relevance for the current dissertation.

1.5.1 Publications Supporting this Dissertation

**International Journals**

- C. Müller, M. Resinas, A. Ruiz-Cortés. Automated analysis of conflicts in WS–Agreement. *IEEE Transactions on Services Computing*. [90]. This paper supports most of the content presented in Chapter §7 and part of the Chapter §8.

**International Journals accepted undergo major revisions**


**International Conferences**


- C. Müller, O. Martín-Díaz, A. Ruiz-Cortés, M. Resinas and P. Fernández. Improving Temporal–Awareness of WS–Agreement. *In Proceedings of the 5th International Conference on Service Oriented Computing*. [75]. This paper supports part of the content presented in Chapter §6.

- C. Müller, A. Ruiz-Cortés, M. Resinas. An Initial Approach to Explaining SLA Inconsistencies. *In Proceedings of the 6th International Conference on Service Oriented Computing*. [72]. This paper is the conference version of the aforementioned journal paper “Automated analysis of conflicts in WS–Agreement”.

- C. Müller, A. Ruiz-Cortés, M. Resinas. Explaining the Non–Compliance between Templates and Agreement Offers in WS–Agreement. *In Proceedings of the 7th International Conference on Service Oriented Computing*. [71]. This paper supports part of the content presented in Chapters §7 and §8.
International Workshops

- C. Müller, A. Ruiz-Cortés and P. Fernández. Temporal–Awareness in SLAs. Why should we be concerned?. *In Proceedings of the 1st Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop*. [76]. This paper supports part of the content presented in Chapter §6.


- C. Müller, M. Oriol, M. Rodríguez, X. Franch, J. Marco, M. Resinas, A. Ruiz-Cortés. SALMonADA: A platform for monitoring and explaining violations of WS-agreement-compliant documents. *In Proceedings of the 4th International Workshop on Principles of Engineering Service-Oriented Systems,(IEEE ICSE Workshops)*. [85]. This paper is the workshop version of the aforementioned journal paper “Comprehensive Explanation of SLA Violations at Runtime”.

National Conferences and Technical Reports

- C. Müller, O. Martín-Díaz, A. Ruiz-Cortés and J. M. García. Consistencia y conformidad en un contexto temporal. *In Proceedings of the workshop entitled “Métodos y Herramientas para el Desarrollo de Aplicaciones” in the 11th “Jornadas de Ingeniería del Software y Bases de Datos”*. [73]. This paper supports part of the content presented in Chapter §6 that was extended and revised in [74].

- C. Müller, O. Martín-Díaz, M. Resinas, P. Fernández and A. Ruiz-Cortés. A WS-Agreement Extension for Specifying Temporal Properties in SLAs. *In Proceedings of the 3rd “Jornadas Científico-Técnicas en Servicios Web y SOA”*. [74]. This paper supports part of the content presented in Chapter §6 that was extended and revised in [75].

- C. Müller, M. Resinas, and A. Ruiz-Cortés. An Initial Approach to Explaining SLA Inconsistencies. *In Proceedings of the 5th “Jornadas Científico-Técnicas en Servicios Web y SOA (Divulgación de Trabajos ya publicados)”*. [79]. This paper disseminates the paper [72] in the track of “Relevant Published Works” of a national conference.

- C. Müller, J. García Galán, A. Ruiz-Cortés, and M. Resinas. ADA: Agreement Documents Analyser. *In Proceedings of the 6th “Jornadas Científico-Técnicas en Servicios Web y SOA”*. [81]. This paper supports part of the content presented in Chapter §8 by a tool demonstration of our analysis tool.
1.5. PUBLICATIONS

• C. Müller, M. Resinas, and A. Ruiz-Cortés. Explaining the Non-Compliance between Templates and Agreement Offers in WS-Agreement*. In Proceedings of the 15th “Jornadas de Ingeniería del Software y Bases de Datos (Divulgación de Trabajos ya publicados)”. [83]. This paper disseminates the paper [71] in the track of “Relevant Published Works” of a national conference.

• C. Müller, M. Oriol, M. Rodríguez, X. Franch, J. Marco, M. Resinas, A. Ruiz-Cortés. SALMonADA: A Platform for Monitoring and Explaining Violations of WS–Agreement-Compliant Documents. In proceedings of the 8th “Jornadas de Ciencia e Ingeniería de Servicios (JCIS)”. [84]. This paper supports part of the validation included in Chapter §8 by a tool demonstration of our platform to monitor and analyse SLAs.

• C. Müller, A. Durán, M. Resinas, and A. Ruiz-Cortés. Experiences from building a WS–Agreement document analyzer tool. [80]. This technical report supports most of the content presented in Chapter §2 and part of the content presented in Chapter §6.

• C. Müller, S. Segura, and A. Ruiz-Cortés. A test suite for Agreement Document Analyser. [86]. This technical report supports the test suite designed to verify our reference implementation that is included in Appendix §G.

1.5.2 Further Publications

• D. Benavides, A. Ruiz-Cortés, M. Toro and C. Müller. Book Chapter entitled: “La esquizofrenia del docente e investigador en ingeniería del software”. In Book “Aspectos Filosóficos, Psicológicos y Metodológicos de la Informática”. It was previously presented in a national workshop “Métodos de Investigación y Fundamentos Filosóficos en Ingeniería del Software”, and extended to be published in the international workshop “Philosophical Foundations of Information Systems Engineering” [14]. [13]. In these publications we propose to use of a researching methodology followed in our research activities, that grouped concepts of “technology monitoring” and “competitive intelligence” into an action research iterative methodology.

• D. Benavides, A. Ruiz-Cortés, C. Muller and P. Trinidad. The triple schizophrenia of the software engineering researcher. In Proceedings of the 1st Workshop on Philosophical Foundations of Information Systems Engineering. [14]. This paper presents our action research methodology in an international workshop.

• C. Müller, M. Resinas and A. Ruiz-Cortés. Using Automated Analysis of Temporal–Aware SLAs in Logistics. In Proceedings of the 1st International Workshop on Service Oriented Computing in Logistics. [78]. The main contribution of this paper was to promote a debate on the usefulness of using the automated analysis of time-aware SLAs in logistics.
• C. Müller, J. A. Parejo, and A. Ruiz-Cortés. A First Approach to Model SLAs for Composite Services, using WS–Agreement. *In Proceedings of the 4th “Jornadas Científico-Técnicas en Servicios Web y SOA”*. [77]. This paper presents an approach to model SLAs for composite services using the schema proposed by WS–Agreement.

• P. Trinidad, M. Resinas, C. Müller, J. A. Parejo, A. Ruiz-Cortés. Aprendiendo a diseñar software usando juegos de mesa como enunciado de prácticas. *In Proceedings of the 17th “Jornadas de Enseñanza de la Informática”*. [134]. In this paper we present an academic methodology followed in a software-related subject.

### 1.6 Outline

**Part I: PREFACE.** It includes this introductory chapter, in which we have summarised the research context of the work developed necessary to understand the content of this dissertation, we have detailed the drawbacks of existing proposals, and we have summed up our contributions to overcome drawbacks identified. Finally, we have described the context in which the dissertation has been developed and the supporting publications.

**Part II: BACKGROUND INFORMATION.** It provides specific information about the elements involved in the scope of the research context of our work. Chapters §2 and §3 details the main concepts of the WS–Agreement recommendation, with the aim of clarifying the recommendation goals to the reader. Chapter §2 introduces the WS–Agreement structure to specify agreement documents, and in Chapter §3 the support for the processes of creating and monitoring WS–Agreement documents is detailed. Afterwards, Chapter §4 presents the elements involved in the analysis of agreement documents and the existing approaches included in the literature.

**Part III: OUR PROPOSAL.** This is the core of the dissertation and it is organised in five chapters. First of all, in Chapter §5 we define the problems identified regarding the specification and analysis of WS–Agreement documents from the results of the studies described in previous chapters, and we review the literature on the topic with regard to each and every problem identified. Then, we present our approaches to overcome the problems. In Chapter §6, we introduce iAgree, our proposed intermediate WS–Agreement configuration, by means of the comprised sublanguages and the validity criteria of iAgree documents. In Chapter §7 we detail the CSP-based formal semantics provided for iAgree, and the automated technique used to check the iAgree documents validity. Chapter §8 provides information about IDEAS, our Integrated Development Environment for SLA-driven Solutions including some validation scenarios of agreement creation and monitoring. In such scenarios, the support provided by iA-
gree automated validity criteria checking to develop SLA-driven solutions have been validated. These validation scenarios provide new analysis operations that are included within the ADA catalogue introduced in the same Chapter. And the Chapter ends with information about the iAgree studio developed to edit and analyse SLA-driven solutions.

Part IV: FINAL REMARKS. Chapter §9 concludes the dissertation with a summary of the contributions and a discussion of their usefulness, limitations and the future work identified.

Part V: APPENDICES. Several appendices have been attached to this dissertation to add supplemental material. Appendix §A contains the EBNF notation of the abstract syntax of iAgree. Appendix §B details the concrete iAgree syntax and its mapping to an XML-based serialised syntax. Appendix §C provides information about the CSP operations used to provide a formal semantics to iAgree and to automate the iAgree validity criteria. Appendix §D includes the former definitions for the automated validity criteria checking and the complete examples exposed in Figures of Section §7.3. Appendix §E includes the translations from other proposed WSACs to iAgree. Appendix §F includes the SLA-driven solution developed to support the SLA fulfillment. Finally, the complete test suite developed to verify the main ADA analysis operations is included in Appendix §G.
Part II

Background Information
WS–Agreement Documents Specification

I present to you the Relationship Agreement. A binding covenant that in its 31 pages enumerates, illuminates and codifies the responsibilities of Sheldon Lee Cooper (hereinafter referred to as the ‘Boyfriend’) and Amy Farrah Fowler (hereinafter referred to as the ‘Girlfriend’).

Sheldon Lee Cooper,
Fictional character on The Big Bang Theory

Your life works to the degree you keep your agreements.

Werner Erhard (1935—…),
American thinker, author of transformational models and applications for individuals, groups, and organizations

In this chapter, we detail the most relevant aspects of the WS–Agreement specification. After a brief introduction to the topic in Section §2.1, we describe in Section §2.2 the general structure proposed in WS–Agreement for agreement documents. Section §2.3 introduces the WS–Agreement configuration concept. In Section §2.4 we detail how a WS–Agreement document may include optional terms inside to define several agreement variants. In Section §2.5 the template compliance definition of WS–Agreement is presented. A study of other SLA specifications is included in Sections §2.6 and §2.7, providing details about the agreement terms and time awareness they support, respectively. Finally, a summary of the content closes the chapter in Section §2.8.
2.1 Introduction

This chapter is focused in the integration of organizations by means of services. In such a context, some organizations consume the services provided by a counter-party organization; in addition, this integration paradigm provide the setting for new specific challenges to be tackled. Concretely, consumer organisations usually require guarantees on the services they intend to consume and a kind of evidence on the quality offered by the provider organisations. Such information may be included in Service Level Agreements (SLAs), that establish the functional and quality aspects of Service-Based Applications (SBAs) [102], in the development of what we call SLA-driven solutions.

A variety of approaches can be found in the literature to describe SLAs by means of natural language intended to be used only by humans\(^1\); formal languages with the aim of analyzing some properties of the SLAs [17, 26, 116]; and XML documents in standardization efforts aimed at making the interoperability between consumers and providers easier [10, 37, 61, 97]. Among the approaches of the latter group we find the WS–Agreement [10] recommendation that includes one of the most widely used SLA specification approaches. This recommendation also provides a protocol for creating and monitoring agreements that is detailed in Chapter §3.

However, WS-Agreement is designed to establish a conceptual foundation and therefore, it does not develop a fully-fledged language to specify final documents but provides the placeholders that should be combined with domain-specific languages; this idea collides with previous SLA specifications such as WSLA [61] or ebXML [97]. WS–Agreement describes a domain-independent schema using an extensible XML language for specifying agreements, agreement offers, and templates to facilitate the discovery of compatible agreement parties. Such an schema can be extended up to eight different points with a sublanguage for each one. More specifically, two of the eight points need to be necessarily extended to achieve a fully-fledged language, namely the service description terms (SDT) that defines the service functionality, and the SLOs.

In this chapter we detail the structure and semantics of the XML–based schema proposed by WS–Agreement, the WS–Agreement configuration concept coined by us, how the recommendation supports optional terms inside WS–Agreement documents, and the template compliance definition provided by the specification. In addition, we also review the different scope, elements, and time awareness supported by other SLAs specifications.

---

\(^1\)SLAs of companies such as Amazon (aws.amazon.com/s3-sla/) or Google (www.google.com/apps/intl/en/terms/sla.html)
2.2 Documents Structure

The general structure of WS–Agreement documents is depicted in Figure §2.1 using a UML class diagram designed by us to clarify the XML structure proposed in the recommendation for the documents. The ellipsis in types and attributes represent the needs for a sublanguage to describe the corresponding elements. Thus, a WS–Agreement document is composed of an agreement identifier, an agreement context and agreement terms. Templates can also include agreement creation constraints to restrict the space of template–compliant offers that can be created from them (cf. Section §2.5). While describing the agreement documents constituents we refer to the template depicted in Figure §2.2.

The agreement context contains information about whether the service provider is the agreement initiator or responder. For instance, in the template of Figure §2.2 Amazon, who is the provider of the storage service, plays the role of responder because it receives agreement offers from other parties and provides a positive or negative response. In addition, it optionally provides information about the agreement parties endpoints, the agreement’s lifetime, references to the template from which an agreement offer is created, if this is the case. Furthermore, the context is open to any other relevant information, for instance, in the template of Figure §2.2 two metrics catalogues are included by reference (cf. Section §6.2.3) and the definition of validity periods (VPs) for the whole or a part of the agreement document (cf. Section §2.7).

The terms section of a WS–Agreement document describes both the characteristics of the services to be provided and the guarantees on such services. Note that terms can be composed just with a unique all or any nested compositor structure as detailed in Section §2.4. In WS–Agreement there are two kinds of terms, namely service terms (ST) and guarantee terms (GT).

2.2.1 Service Terms

The service terms describe those service features that are relevant for an agreement and are divided into service description terms, service properties, and service references.

Service description terms (SDTs) define the features of the service that will be delivered under an agreement. These features are such that do not make sense to monitor at service delivery time because they must provide an unambiguous description for the service that is subject of the agreement. Thus, they must have assigned specific values in agreement offers and SLAs, but not necessarily in templates.

In case of templates including service features without an assigned value, the other party may choose suitable values for them according to their domains and constraints.

Note that such a template is described following the human-readable iAgree concrete syntax detailed in Chapter §6. Appendix §B includes the same template serialised in XML following the structure proposed by WS–Agreement.
CHAPTER 2. WS–AGREEMENT DOCUMENTS SPECIFICATION

Figure 2.1: UML metamodel of the WS–Agreement schema
included in the template (cf. creation constraints at the end of the section). In case of service features with value assignments in templates, such as the DataDurability and RRS (Reduced Redundancy Storage) of Figure §2.2, they describe default values that can be changed by the other party according to the aforementioned domains and constraints. In the example, the assured data durability is of 99.999999999 % for the standard storage, that is RRS = False. Note that the DataDurability feature is not monitorable due to AmazonS3 is designed to provide a specific durability for the stored data. How these SDTs are organized and expressed must be defined in one or more sublanguages. For instance, in our example, we use the iAgree sublanguage (cf. Section §6.3) for SDTs to describe the AWS-S3 storage service by means of a set of attribute–value pairs, namely, the user region, the service interface used (SOAP or REST), the data durability, the required kind of redundancy (default or reduced), the storage size, an indicator if it is the first user storage project or not, and the price to pay for the storage, customer support plan, and for the whole service.

Service properties (SPs), define named, service–related sets of monitorable variables that can be used for the specification of guarantee terms. In contrast with the SDT features, the SPs must be considered for agreement monitoring because their values may change at service delivery time. All variables must be related to a SDT or a part of it and they may include a metric definition to specify the semantics and type of a variable with one or more sublanguages. In our example, the context includes two metrics catalogues iAgree.generalMetrics, and Amazon.metrics that describe the metrics and domains used in the document (cf. Section §6.2.3): a kind of service availability of the service that is called Monthly Uptime Percentage (MUP) (cf. Section §2.2.2 for more details); the size of transferred data; the service response time; the read and write requests performed by the user; and some support plan properties such as the online reporting, turn around time to solve problems and the phone support.

Finally, service references (SRs) provide endpoint references for the services under agreement and they also need one or more specific sublanguages to be defined. In the example it is included after the service name and immediately before the definition of the service description term.

---

3AmazonS3 provides a highly durable storage infrastructure designed for mission-critical and primary data storage. The service redundantly stores data in multiple facilities and on multiple devices within each facility. To increase durability, AmazonS3 synchronously stores the data across multiple facilities before returning SUCCESS. In addition, AmazonS3 calculates checksums on all network traffic to detect corruption of data packets when storing or retrieving data.

4Reduced Redundancy Storage (RRS) is a storage option within AmazonS3 that enables customers to reduce their costs by storing non-critical, reproducible data at lower levels of redundancy than AmazonS3’s standard storage. It provides a cost-effective, highly available solution for distributing or sharing content that is durably stored elsewhere, or for storing thumbnails, transcoded media, or other processed data that can be easily reproduced. The RRS option provides 400 times the durability of a typical disk drive, but does not replicate objects as many times as standard AmazonS3 storage.
### Template AmazonS3 version 1.0

**Provider** Amazon as Responder;

**Consumer** "Consumer with storage necessity";

**Global Period**: from 00:01 to 24:00 on SUN-SAT during 01/01/2012..12/31/2013 except from 23:00 to 24:00 on SUN;

**Defined Period**: FreeTrialQuota during 01/01/2012..12/31/2012;

**Metrics** [iAgree.generalMetrics, Amazon.metrics] //includes the domain metrics domains

**Agreement Terms**

- **Service** AWS-S3 available at. aws.amazon.com/s3 //Service reference in iAgree
- **Global description**: //Service description term in iAgree
  - Region;
  - Interface;
  - DataDurability = 99.999999999; //Probability of a successful data storage
  - RRS = False; //Reduced Redundancy Storage (RRS)
  - StorageSize; //StorageSize in TB
  - FirstProject; //Denotes if it is the first customer project
  - TotalPrice, StoragePrice, SupportPlanPrice;

**Monitorable Properties** //Service properties in iAgree

  **global:**
  - MUP; //Monthly Uptime Percentage, a kind of AmazonS3 service availability
  - TransferredGb;
  - ResponseTime;
  - ReadRequests, WriteRequests;
  - OnlineReportingSupport, PhoneSupport; //Customer support facilities
  - TurnAroundTime; //Minutes to solve problems

**Guarantee Terms**

- **G1**: **Provider guarantees** MUP >= 99.9; subject to a penalty of 1 ServiceCredit per 1 failed request; //simplified penalty
- **G2**: **Consumer guarantees** TransferredGb < StorageSize * 100
  - AND ReadRequests > WriteRequests;
- **G3**: **Provider guarantees** ResponseTime < 1000; //G2-G4 are inspired in the AmazonS3 SLA,
  - onlyIf (Interface = SOAP); //but it is not included in it
- **G4**: **Provider guarantees** ResponseTime < 700;
  - onlyIf (Interface = REST);

... G5-G9 omitted for spaces reasons ...

- **G10**: One or More between:
  - **G10.1**: **Provider guarantees** OnlineReportingSupport = true;
  - **G10.2**: **Provider guarantees** TurnAroundTime = 15;
  - **G10.3**: **Provider guarantees** PhoneSupport = true;
  - **G10.4**: **Provider guarantees** ExtendedSupport = true;

**Creation Constraints**:

- **C1**: Region belongs {US, EU_Ireland};
- **C2**: TotalPrice = 0;
  - on FreeTrialQuota;
  - onlyIf (FirstProject = true and StorageSize <= 5);
- **C3**: StoragePrice = 0.05 * StorageSize;
  - onlyIf RRS = true;
- **C4**: StoragePrice = 0.12 * StorageSize;
  - onlyIf RRS = false;
- **C5**: StorageSize <= 5000 TB;
- **C6**: TotalPrice = StoragePrice + SupportPlanPrice;
  - onlyIf (FirstProject = false or StorageSize > 5);
- **C7**: DataDurability >= 99.99;
  - onlyIf (RRS = true);
- **C8**: DataDurability >= 99.999999999;
  - onlyIf (RRS = false);

---

**Figure 2.2**: Template of AmazonS3 service scenario in iAgree (see Chapter §6)
2.2.2 Guarantee Terms

The guarantee terms describe the SLOs that an obligated party, usually the service provider, must fulfill as part of the agreement. The SLO of a guarantee term is an assertion defined over monitorable variables defined in the service properties section of the agreement document and over external factors such as date or time. Optionally, SLOs can be related to externally defined key performance indicators (KPIs), i.e. externally defined monitorable variables.

Each guarantee term can be guarded by an optional qualifying condition (QC) that expresses a precondition under which the guarantee holds. Thus, using QCs is the way WS–Agreement allows for defining conditional terms such as G3 and G4 terms of the example in which the provider assures diverse service depending on the interface used to consume the service, either SOAP or REST. Both QCs and SLOs can be expressed using any assertion language and so, users must specify sublanguages for them. In the example we use a predicate-oriented sublanguage that is detailed in Section §6.4. The structure of a guarantee term also includes a scope, i.e. a set of service operations to which the guarantee is applied; and a list of business values (BVL) to describe the relative importance between the terms, penalties, etc. For the sake of simplicity, we have included in the G1 term of Figure §2.2 a simplified version of the clause offered by Amazon to penalise itself by a lack of service. In the Figure clause, the customer are able to claim for a reward of a ServiceCredit per each failed request. However, in the SLA it is described that Amazon requires the customer to proof this violation by sending an email within ten business days after the end of the billing cycle in which the errors occurred. Proving the violation demands the customer to monitor and compute the MUP by subtracting from 100% the average of the error rates. Regarding the penalties, Rana et al. in [113] provide an approach to (1) identify classes of penalty clauses that can be associated with an SLA; (2) define how to specify penalties in an extension of WS–Agreement; and (3) specify to what extent penalty clauses can be enforced based on monitoring of an SLA. More examples of scope and business value elements are included in Section §6.4.

2.2.3 Creation Constraints

The creation constraints (CCs) can be included in templates in order to specify the mandatory presence of specific elements and their acceptable values, in offers created from such templates. To this end, the template may define specific items using XML Schema [140] to delimit the possible value assignments for the service features defined in the SDTs or, if the creation constraint involves several elements, they can be specified using any suitable constraints language that must be described by means of one or

---

5Service credits are obtained by applying a formula that considers both: (1) the amount paid for Amazon service within the month in which the lack of service takes place; and (2) the lack of service degree (cf. http://aws.amazon.com/s3-sla/ for more detail).
6Internal server errors divided by the requests during each five minute period
more sublanguages. Since items can be considered as a particular case of constraints, in i Agree we just consider a sublanguage for the constraints (cf. Section §6.6).

As the example of Figure §2.2 denotes, the creation constraints is where the service price policies must be specified in WS–Agreement documents. This is due to the price policies delimits the specific amount to pay for the service, that is included in the service description terms of both: agreement offers created from such a template, and in the final agreement signed by the parties. For instance, in the example, the TotalPrice to pay for the service results from adding the SupportPlanPrice and the StoragePrice that depends on if the redundancy storage has been required to be reduced or not. Furthermore, there is no price to pay during the FreeTrialQuota period. Therefore, an agreement offer created from such a template must observe these constraints and the demanded data durability depending on the chosen RRS, in the value assignment of service description terms.

2.3 WS–Agreement Configurations

The WS–Agreement recommendation neither provides default sublanguages, nor imposes additional constraints on the expressiveness of such sublanguages. Figure 2.3(a) shows the general structure of WS–Agreement documents from an abstract point of view including small white boxes that represent the sublanguages that users must define to describe the corresponding element of WS–Agreement documents. As such figure does not clarify which of the sublanguages are mandatory, optional, or require the definition of another sublanguage, we have included such information in the variability model of Figure 2.3(b).

As the number of different sublanguages to conform fully-fledged WS–Agreement compliant languages is unbounded, it is also unbounded the number of WS–Agreement Configurations, or WSACs for short, that denote to each and every possible combination of sublanguages. Despite authors that deal with WS–Agreement-based approaches do not provide rigorous definitions for the WSACs they use, we have inferred its implicit WSAC by analysing the kind of supported sublanguages of their approach.

Table §2.1 includes some of such implicit WSACs whose most relevant examples are WSAG4J framework\(^7\) and SWAPS, proposed by Waldrich et al. in [96], and Oldham et al. in [100], respectively. The first one uses a WSAC with grid-oriented sublanguages such as Job Submission Description Language (JSDL) [5] to describe the services, XSD-Schema for the service properties metrics and creation constraints, and Java Expression Language\(^8\) (JEXL) [41] for expressive SLOs and QCs that supports conditional terms. In turn, SWAPS [100] WSAC, uses WSDL-S/OWL for the SDTs, and the expression

\(^7\)WS–Agreement framework for Java developed by GRAAP members [http://packcs-e0.scai.fraunhofer.de/wsag4j/index.html](http://packcs-e0.scai.fraunhofer.de/wsag4j/index.html).

\(^8\)[http://commons.apache.org/jexl/](http://commons.apache.org/jexl/)
2.3. WS–Agreement Configurations

(a) WS–Agreement documents structure

(b) WS–Agreement mandatory and optional sublanguages

Figure 2.3: WS–Agreement structure and sublanguages
Implicit WSACs inferred from related approaches

<table>
<thead>
<tr>
<th>Approaches</th>
<th>WSAC Sublanguages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Context</td>
</tr>
<tr>
<td>WSAG4J [96]</td>
<td>✓</td>
</tr>
<tr>
<td>SWAPS [100]</td>
<td>✓</td>
</tr>
<tr>
<td>Palacios et al [104, 105]</td>
<td>✓</td>
</tr>
<tr>
<td>Mahbub et al [63]</td>
<td>✓</td>
</tr>
<tr>
<td>Comuzzi et al [26]</td>
<td>~</td>
</tr>
<tr>
<td>AgentScape [38]</td>
<td>✓</td>
</tr>
<tr>
<td>AssesSGrid [4]</td>
<td>✓</td>
</tr>
<tr>
<td>BEinGRID [33]</td>
<td>✓</td>
</tr>
<tr>
<td>Brein [3]</td>
<td>✓</td>
</tr>
<tr>
<td>VIOLA [40]</td>
<td>✓</td>
</tr>
<tr>
<td>JSS [136]</td>
<td>✓</td>
</tr>
</tbody>
</table>

Legend: ✓ support, ~ partial support

Table 2.1: Sublanguages supported by implicit WSACs of related approaches

The implicit WSAC inferred from Mahbub et al [63] approach describes the services, SLOs and QCs by means of expressive event calculus assertion (EC-Assertion) formulae (cf. the example of Table §4.1). However, it has a complex concrete syntax and they do not provide support for creation constraints and business value list.

The implicit WSAC obtained from the approach of Comuzzi et al [26] supports an expressive ad-hoc SLA specification model that is serialised to WS–Agreement. However, they do not detail such a serialisation process and studying its SLA model we detected some mismatching between such a model and WS–Agreement. For instance, they propose the use of an external price model that is not included in the SLA. This does not match with the use of creation constraints of WS–Agreement to delimit the allowed values for service properties such as the prices, between others.

The other implicit WSACs [3, 4, 33, 38, 40, 136] included in the Table §2.1 were detailed and studied for the WS–Agreement promoters in a technical report [35]. In such a technical report the support provided by each approach for the WS–Agreement elements is detailed.

The flexibility provided by these extension points is a great advantage of WS–Agreement since it makes it possible to use it in a virtually infinite number of different
2.4. VARIANTS

The terms in a WS–Agreement document can be grouped using three different terms compositors denoting that the comprised terms are either mandatory, optional, or alternative.

- mandatory terms compositor (All in Figure §2.1): every comprised term or compositor is mandatory. In other words, all of them must be fulfilled. WS–Agreement specification imposes that at the top level of the terms section, all terms must be inside a mandatory terms compositor. For the sake of simplicity we consider as implicit such a top level mandatory terms compositor. Thus, we do not use any keyword for it in the concrete syntax used in the document of Figure §2.2.

- optional terms compositor (OneOrMore in Figure §2.1): every comprised term or compositor is optional. In other words, a set of them, at least one, must be fulfilled.

- alternative terms compositor (ExactlyOne in Figure §2.1): every comprised term or compositor is alternative. In other words, only one of them must be fulfilled. For instance, the template of Figure §2.2 includes one of that compositors to allow any combination of features for the customer support plan.

Term compositors can be nested, thus enabling the specification of alternative branches with potentially complex nesting within the agreement terms. Choices expressed using compositors can be exercised by the party that makes the next step in the agreement creation process, i.e., by the agreement initiator if it is creating an offer from a template, by the agreement responder if it is creating an agreement from an offer, or by the service provider if it is delivering the service according to a previously created agreement. For instance, document of Fig. §2.4 offers the alternative of either choosing G1 or choosing G2 and one or more terms between G3 and G4. The AWS–S3 service description term and the monitorable service properties are included in each and every of the alternatives because they are comprised by the implicit top-level mandatory terms compositor.
CHAPTER 2. WS–AGREEMENT DOCUMENTS SPECIFICATION

Template: AmazonS3 version 1.1

Provider Amazon as Responder;
Consumer "Consumer with storage necessity";
Metrics [iAgree.generalMetrics, Amazon.metrics]

Agreement Terms
Service AWS-S3 available at. aws.amazon.com/s3

Global description:
Region;
Interface;
DataDurability = 99.999999999;
RRS = False;
StorageSize;
FirstProject;
TotalPrice, StoragePrice, SupportPlanPrice;

Monitorable Properties

global:
MUP;
TransferredGb;
ResponseTime;
ReadRequests, WriteRequests;
OnlineReportingSupport, PhoneSupport;
TurnAroundTime;

Guarantee Terms
Exactly One between:
G1: Provider guarantees ResponseTime < 1000;
onlyIf (Interface = SOAP);
All
G2: Provider guarantees ResponseTime < 700;
onlyIf (Interface = REST);
One Or More between:
G3: Provider guarantees OnlineReportingSupport = true;
G4: Provider guarantees TurnAroundTime = 15;

Creation Constraints //as in Figure 2.2

Figure 2.4: An example of nested term compositors, inspired in AmazonS3 scenario

Variant 1: AWS-S3, G1
Variant 2: AWS-S3, G2, G3
Variant 3: AWS-S3, G2, G4
Variant 4: AWS-S3, G2, G3, G4

Figure 2.5: Enumeration of all variants defined by the TCs in document of Fig. §2.4

Note that parties can exercise the choice but it is not mandatory to do so. In other
words, the term compositors can remain in an offer and even in a final agreement,
exact as they were defined in the former template.

We call variant\(^9\) to each of those sets of terms that can be chosen in one WS–Agreement
document. Consequently, term compositors can be seen as means to define the
variability of a WS–Agreement document in terms of the variants that can be chosen
by the party that makes the next step in the agreement creation process. Furthermore,
the use of variants supports a compact description of several WS–Agreement docu-
ments in a unique document. For instance, the document of Figure §2.4, is the compact

\(^9\)Oldham et al call it alternative set of guarantee terms in [100]
2.5 COMPLIANCE OF OFFERS WITH TEMPLATES

According to WS–Agreement\textsuperscript{10}, the purpose of templates is to give guidance on what forms of offer an agreement responder wishes to receive. As such, offers SHOULD in general comply with one of the templates advertised by the responder. However, the responder MAY accept offers which do not match any template, and the responder also MAY reject offers that do match for other policy reasons. And it is defined that an agreement offer is compliant with a template advertised by an agreement responder if and only if each term of service described in the Terms section of the agreement offer complies with the term constraints expressed in the CreationConstraints section of the agreement template. In addition, in the Context of the offer, the AgreementResponder value must match the value specified in the template; and the TemplateId must exactly match the name provided in the template document against which compliance is being checked.

Such a definition is not completely precise because terms of service could be interpreted either as the service terms, or as terms of ‘the’ service, considering the lack of the “the” article. The main difference between these two interpretations is the consideration of guarantee terms in the second compliance notion. The spirit of the WS–Agreement protocol focuses us to such a second compliance notion that can be defined as: there exist compliance between an offer and a template if and only if none of the offer terms contradicts the template creation constraints. Nevertheless, as far as we know, such a problem with the compliance definition has not been tackled in the existing literature. From now on we will denote that an offer O is compliant with a template T as areCompliant(O, T) or as $O \rightarrow T$. A precise compliance notion in the context of SLAs can be found in [116].

Following this compliance definition, documents of Figure §2.6 would be considered as non-compliant because of two reasons. Firstly, while the StoragePrice is established in the template creation constraints as 0.12 €/GB if not reduced redundancy (RR), the party who describes the agreement offer tries to pay 0.10 €/GB for such a non-redundancy storage service. And secondly, while the higher allowed value for the StorageSize is established as 5000 TB in the template creation constraints, in the agreement offer such a higher value is established as 10000 TB that is less restrictive.

2.6 OTHER SLA SPECIFICATIONS

Other SLA specifications have been proposed in the literature before and after the publication of the initial version of WS–Agreement recommendation in 2005. In this

\textsuperscript{10}Page 33 of the specification [10]
Why an Offer is not Compliant with a Template?

In this section we do not aim to perform a systematic review of such approaches, but to note the different scope and support for kinds of term and term elements they provide taking WS–Agreement specification as reference.

Quality Requirements Language (QRL) [115] (published in 2002). QRL is a fully-fledged and expressive language to describe quality requirements of products. The quality requirements and the products can be considered as the SLOs and SDTs of WS–Agreement documents, respectively. As WS–Agreement proposes the use of service properties and metrics to be used in the SLOs, QRL, similarly, describes a catalogue of attributes with their metrics to be used within the quality requirements. QRL supports the definition of some advanced features that inspired us in the development of iAgree (cf. Chapter §6) such as: (1) expressive validity periods for the quality requirements; (2) validity criteria for the quality requirements in order to assure the consistency of a requirement and the compliance between several requirements; (3) the utility of quality requirements; and (4) negotiation clauses to help in the agreement creation process between parties. However, QRL does not provide a mechanism to in-
clude nested compositions of quality requirements. On the contrary, WS–Agreement supports the specification of agreement variants by means of the terms compositors.

**SLA* specification [55] (published in 2010).** SLA* is a domain-independent syntax for machine-readable SLAs and SLA templates. Historically, SLA* was developed as a generalisation and refinement of the web-service specific XML standards: WS–Agreement, WSLA, and WSDL. Instead of web services, however, SLA* deals with services in general, and instead of XML, it is language independent. SLA* provides a specification of SLA(T) content at a fine-grained level of detail, which is both richly expressive and inherently extensible supporting controlled customisation to arbitrary domain-specific requirements. The model was developed as part of the FP7 ICT Integrated Project SLA@SOI, and has been applied to a range of industrial use-cases, including; ERP hosting, Enterprise IT, live-media streaming and health-care provision. SLA* authors defend in [55] that WS–Agreement is not suitable for human-based services such as the health-care use case, but, as far as we know, the elements included in such an use case can be described in WS–Agreement, somehow. SLA* defines a couple of elements that are not included in the WS–Agreement specifications but they may be included by using the sublanguages of WSACs. These elements are: (1) the concept of *operatives* to refer the parties of an agreement in SLA*, instead of consumer and provider established by WS–Agreement (it may be included in the context sublanguage); and (2) a syntactic facility denoting a set of service operations, instead of the scope used in WS–Agreement that does not allow to define such operation sets. However, as far as we know, SLA* does not supports several elements considered by WS–Agreement such as: (1) term compositors to include variants in a unique document; (2) custom business value expressions for preferences, because SLA* just provides the priority of a term that is similar to the relative importance included in WS–Agreement; and (3) creation constraints with a different purpose than the definition of price policies, for instance the following creation constraints are not supported by SLA*: The SLA must be signed by the parties a day before the starting time; or if the number of write request is less than 100 per hour the customer support plan does not include the phone assistance.

**Web Service Level Agreement (WSLA) [61] (published in 2002).** WSLA, developed by Heiko Ludwig et al. (IBM), provides an XML-based fully-fledged language for SLAs of web services, as well as a negotiation, monitoring and notification framework tightly coupled to the WSLA language. WSLA has inspired further specifications such as WS–Agreement and the conceptual design of SLA*. WSLA provides specific and expressive languages to describe the services properties and operations, SLO assertions and evaluation events to enable such SLOs; these elements can be considered as the SPs, SDTs, SLOs and QCs of WS–Agreement documents, respectively. In addition, WSLA supports expressive validity periods for the SLOs. However, WSLA does not provide the extensibility of WS–Agreement and SLA* specifications. Moreover, the terms cannot

---

11 The use case is detailed in [http://sla-at-soi.eu/use-cases/e-government/](http://sla-at-soi.eu/use-cases/e-government/)
be composed by terms compositors as in WS–Agreement. And furthermore, WSLA has been discontinued and no current implementation of it is provided.

Collaboration-Protocol Profile and Agreement Specification (CPPA) [97] (published in 2002). OASIS has proposed an electronic commerce collaboration standard language called ebXML [99]. It is based on previous works developed in projects as Unified Business Agreements and Contracts (UBAC), performed in 2002 by the United Nations Centre for Trade and Facilitation and Electronic Business (UN/CEFACT). The main objective of ebXML is to obtain a setup document to be used at runtime called ebXML Business Service Interface Configuration. Thus, ebXML is a complete approach which wraps all necessary aspects to define the information exchange in SOA systems. The starting point to obtain the setup document is to perform the ebXML business process (ebXMLBP) [98], which is based on business processes standard languages as BPMN. Then, it is necessary to study the collaborations between the interested parties by means of different OASIS specification languages. For the business collaboration context they propose to use the Collaboration-Protocol Profile (CPP) specification. CPP describes the agreement party capabilities, including information about: each party, communication protocol, security, document exchange protocol, retries, and other information necessary for an autonomous interaction between parties. Once described the CPP, it is mandatory to define information system terms and constraints to allow the electronic exchange. Such terms and constraints are specified by the ebXML agreement specification called Collaboration-Protocol and Agreement (CPA). CPP and CPA are included in the Collaboration-Protocol Profile and Agreement Specification (CPPA) [97]. Since the CPPA specification is focused in electronic services, they provide only a limited set of domain-specific quality of service constraints on the kind of messages, certificates, protocols, etc. Therefore, the differences in the scope of CPPA and WS–Agreement specifications make negligible a comparison between them.

Rule-Based Service Level Agreement Language (RBSLA) [109] (published in 2005). RBSLA is a declarative rule language which enhances the XML-serialised RuleML language with useful constructs to express SLA terminology. RBSLA support to express predicates, event condition action rules, and rule priorities, that describes concepts similar to the SLOs, QCs, and relative importance of BVL, of the WS–Agreement specification. In addition, RBSLA supports specific vocabularies defined externally (e.g. in RDFS/OWL ontologies, or Java class hierarchies) for elements such as metrics, pricing policies, rights and obligations. A great advantage is that off-the-shelf RuleML engines already exist for the processing of such SLAs in order to validate the SLA rules.

---

12 [http://ruleml.org/](http://ruleml.org/)
13 [http://www.w3.org/TR/rif-rdf-owl/](http://www.w3.org/TR/rif-rdf-owl/)
2.7. TIME AWARENESS IN SLAS

CC-Pi specification [19] (published in 2007). CC-Pi proposes a theoretical framework for mapping SLAs to service constraints. The CC-Pi model is, however, tightly-coupled to the negotiation and monitoring process. This leads to the lack of concepts such as agreement parties or service interfaces. Nevertheless, CC-Pi supports to describe variables and constraints that connect the involved variables, as the SPs and SLOs of WS–Agreement specification. CC-Pi defines a validity criteria to reach and validate agreements using Constraint Satisfaction Problems (CSPs) [135] that considers SLA requirements imposed by the involved parties.

2.7 TIME AWARENESS IN SLAS

An SLA must specify the lifetime and the validity of any kind of comprised terms or information. Such time awareness may be included in SLAs as time periods denoting the validity of SLAs or a part of them, and thus they are usually called as validity periods (VPs) [68, 115].

A practical way to establish a VP for a term or a set of them in an SLAs is by means of a tuple \( <T, VP> \), where \( T \) is a set of one or more terms. For instance, the AmazonS3 SLA includes a free trial quota that applies for the first AmazonS3 project of any user, with an storage limit of 5 GB, and expires twelve months after the project starting time. Such a free trial quota could be expressed as follows: \( <\text{FirstProject}=\text{true} \land \text{StorageSize}<5 \text{ GB} \land \text{cost}=0, \text{ For the first 12 months of the project}> \). A more expressive kind of interval is included in global period of the AmazonS3 example of Figure §2.2. Such a global period is affected by periodical maintenance tasks performed at the end of each Sunday\(^{14}\).

The VPs an SLAs should support [73, 75] can be disjoint and non-disjoint. Disjoints are time intervals wrapping time points from a starting to a final instant, but excluding specific time points. For instance a service that is offered just on working days wont include weekends. Non-disjoint are time intervals wrapping time points from a starting to a final instant without excluding time points. These time intervals were studied by Allen in [8], and to complete the two previously mentioned period types, we include the periodical feature if a time period is repeated regularly. The IETF RFC 3060 specification [127] was our inspiration to include the semantics of periodicity.

Figure §2.7 includes examples for all mentioned VPs: Global Validity Period is disjoint and non-periodical, Christmas Validity Period is non-disjoint and non-periodical, Working Validity Period, the Home Validity Period showed above are disjoint and periodical from monday to friday, and the Home Validity Period showed below is non-disjoint and periodical from saturday to sunday.

That way to describe the VPs can be used to analyse the time-awareness allowed

\(^{14}\)such maintenance tasks is not performed by Amazon, but it is included in the example to show the supported kind of VPs.
CHAPTER 2. WS–AGREEMENT DOCUMENTS SPECIFICATION

Figure 2.7: Examples of disjoint, non-disjoint, and periodical validity periods

by the existing SLA specifications:

The WS–Agreement specification [10] considers two locations to include time-awareness. An expiration time can be included to specify the last valid instant of the agreement. Note that such an expiration time, only allows to express a non-disjoint interval starting when the agreement is created: <all SLA terms, From the agreement creation to the expiration time>. Disjoint and/or global VPs are not allowed by WS–Agreement specification for VPs for the whole SLA, that we call global VP (GPs). The WS–Agreement specification recommends the use of QCs of conditional terms to specify VPs for specific terms, that we call local VPs (LPs), but an specific language to define such LPs is out of the WS–Agreement scope.

The study of time-awareness of Oasis UNCEFAT ebXML, must be done focused on its Collaboration-Protocol Profile and Agreement (CPPA) specification [97]. CPPA supports the definition of an agreement lifetime by means of an start and end points for the agreement. So, it allows a non-disjoint GPs starting from a different instant than the current. In addition, the specification states that after the end point, the agreement must be renegotiated by the parties. However, the LPs are not mentioned in the CPPA specification.

There are other SLAs and web services specification approaches with the following support for time-awareness:

Firstly, a group of approaches allowing non-periodical and non-disjoint GPs is comprised by the following works: An OASIS standard approach over UDDI repositories called UDDI Extension [7]; WS-QoS Ontology of Min Tian et al. [129], which proposes the specification of an expires element denoting a service description expiration; A set of semantic-web based service specifications [23, 24, 46, 59, 131]; And Lodi et al. [60], which discuss the design, implementation, and experimental evaluation of a middleware architecture for enabling SLA-driven clustering of QoS-aware application servers. Supporting application server technologies with dynamic resource management including a GP for the agreement.
Secondly, a group of approaches allowing periodical and non-disjoint LPs is comprised by the following works; Web Service Level Agreement (WSLA) - a precursor of WS–Agreement- [61]; Dimitris Gouscos et al. [47], which propose to model certain web service management attributes such as QoS and provision price, and discuss how this information can be accommodated within basic specification standards such as WSDL and exploited within the web service deployment and application lifecycle; WS-Policy Constraints of Sun Microsystems and OASIS [144] which extends the general WS-Policy definition framework of IBM [137] in order to express generic constraints on policies which can contain temporal information or even group several constraints with the same temporal information into a set.

Thirdly, a couple of approaches allow non-periodical and non-disjoint GPs for the whole service or agreement specification, and periodical and non-disjoint LPs. Specifically, SLA* specification [55] that supports: the GPs by the effectiveFrom and effective-Until elements; and the LPs through events associated to the agreement constrains. Akhil Sahai et al. propose Web Service Monitoring Language (WSML) [118] to specify the web services management and supervision; and Vladimir Tosic et al. propose the Management Applications of the Web Service Offering Language (WSOL) [130].

We find another couple of approaches that mention to allow in their future works non-periodical and non-disjoint global and/or LPs. Specifically, Glen Dobson et al. [34], which propose an unified quality of service ontology including the aforementioned kind of temporal aspects; and Jos de Bruijn et al. [30], which study the different implementations for the diverse family of Web Service Monitoring Languages (WSML).

We have also found other approaches that can not be grouped in the previous sets but they propose a kind of time-awareness on SLA. Dyachuk and Deters [36] approach is based on WSLA [61] and allows to specify a maximum waiting time for a specific component service of a composite web service in order to use an alternative service, if the time expires. This can be considered as a LPs for the composite web service specification. Tomarchio et al. [87], propose an extension for WS–Agreement that supports the VP for the whole agreement by an expiration time as established by WS–Agreement. In addition, they provide the specification of time intervals in which an agreement can be modified and renegotiated at monitoring time. Finally, Cardellini et al. [21] study the composite web service provisioning and they allow the specification of a GP for the whole composite web service.

Aforementioned works are summarised in Table §2.2.

2.8 SUMMARY

In this chapter we have described the main aspects of the WS–Agreement specification, its aims and semantics. Thus we provide: (1) a brief comparison with other
SLA specification such as QRL [115], SLA* [55], WSLA [61], CPPA [97], RBSLA [109], and CC-Pi [19]; (2) the structure of WS–Agreement documents and the semantics provided in the recommendation; (3) the discussion on the unbounded number of WS–Agreement configurations; (4) the possibility to describe several agreement documents in an unique document by means of variants; and finally (5) the template compliance definition. Furthermore, we have also included an study of the time awareness supported by a number of related SLA specifications.
Technology is dominated by two types of people: those who understand what they do not manage, and those who manage what they do not understand.

Archibald Putt,
author of Putt’s Law

In this chapter, we detail the most relevant aspects of the creation and monitoring protocol proposed in the WS–Agreement specification. After a brief introduction to the topic in Section §3.1, we describe in Section §3.2 the different phases in which SLAs may be during its lifecycle. While in Section §3.3 it is detailed the agreement creation protocol, Section §3.4 explains the facilities provided by WS–Agreement to monitor such created agreements. Section §3.5 informs about the existing tooling support for the creation and monitoring of WS–Agreement documents. Finally, a summary of the content closes the chapter in Section §3.6.
CHAPTER 3. WS–AGREEMENT PROTOCOL

3.1 INTRODUCTION

In this chapter we introduce the different phases that should be followed during the so-called SLA lifecycle to create and make use of SLAs, namely: preparation, information, negotiation, deployment, fulfillment, and termination. The WS–Agreement specification proposes a protocol to create and monitor agreements that considers from deployment to termination phases of such SLA lifecycle.

In order to clarify the agreement creation protocol proposed by WS–Agreement we provide the most common usage scenario and some additional considerations included in the recommendation. In addition, we mention the normative states in which agreements and guarantees may be during the monitoring phase in order to evaluate the SLA fulfillment. Finally, we provide information about the existing tooling support with special attention to WSAG4J, a framework that implements the WS–Agreement protocol.

3.2 SLA LIFECYCLE

During the last decade SLA management has become both: (1) an increasingly important issue in the IT infrastructure [88] [125], and (2) a fundamental aspect to achieve a quality-driven service consumption. In this context, in order to ease the management of SLAs, it is necessary to identify the different phases that should be followed during the so-called SLA lifecycle to create and make use of SLAs; this has been called in the literature as the contracting process [50].

Several proposals mention different phases of the SLA lifecycle. Thus, authors like Comuzzi [26], Radha [58] and Vonk [138] consider early SLA phases performed from the SLA preparation to the SLA deployment. Other proposals such as the WS–Agreement specification [10], pay more attention to such late SLA phases performed from the SLA deployment to the termination or expiration of the agreement, but performing a monitoring phase while the SLA is valid, in which the SLA fulfillment is controlled. Finally, authors like Koller [57] try to consider each and every SLA management phase in its approach. As these proposals provide a limited number of processes to manage the SLAs, we will focus on the general-purpose phases mentioned in the Fernández-Montes PhD [39], where a meta-model for trading architectures is presented. Such a service trading model supports the existing variability on: negotiation models, roles, interactions, protocols, etc. Thus, the phases we consider in this dissertation for the SLA lifecycle are the followings:

Preparation. It is a special stage that involves the creation of a document with the service offered by the provider and the analysis of its functional and non-functional requirements by the consumer (also called preferences).
3.3 CREATING SLAS WITH WS–AGREEMENT

Information. The goal of this phase, that is also called as discovery and selection, is to match service providers with potential consumers and vice versa. In other words, either consumers try to find a suitable provider for their desired services and service levels, or providers try to find service demands from potential consumers. In order to locate and select a suitable counterparty, the parties interchange public information during this phase that can be eventually aided by an intermediary party that provides an ordered list of potential counter parties [44].

Negotiation. Is the phase in which parties try to achieve an agreement on the service level for the service provisioning. Several offers and counter-offers are usually interchanged at this complex phase in which several strategies can be followed in the pursue to achieve a final agreement signed by the parties [114].

Deployment. In this phase both, the service provider and consumer set up a deployment plan to make it possible to follow all terms established in the agreement settled in the previous phase. The phase delimits when the service subject of the agreement can be started to be consumed or provided.

Fulfillment. This is the last phase in the contracting process and it involves the monitoring of the whole process in order to ensure that both parties observe the obligations established in the agreement, correctly. This phase requires specific monitoring tools and approaches [63, 101, 113].

Termination. This is the last phase of the SLA lifecycle and it delimits when the SLA expires and then, the service provisioning is stopped for the specific consumer. At this moment if the agreement includes termination clauses they must be considered.

3.3 CREATING SLAS WITH WS–AGREEMENT

The agreement creation protocol supported by the WS–Agreement recommendation proposes the use of agreement offers and templates to perform the corresponding phases of the SLA lifecycle mentioned in the previous Section §3.2.

The most common usage scenario of the agreement creation protocol starts with the preparation of an agreement template. A template is a partially completed offer that specifies customizable aspects of the documents, and rules that must be followed in creating an agreement, which are called agreement creation constraints (cf. Section §2.2). Figure §2.2 depicts a template prepared by Amazon for its AmazonS3 service.

As depicted in Figure §3.1, once the template is published, it can be found by any
potential customer demanding the storage service at the *information* phase of the life-cycle. Such a potential customer prepares an agreement offer compliant with the published template and initiates the creation process by sending it to the provider.

Finally, the provider either accepts or rejects the agreement offer. If rejected, new offers may be send to the provider in this *negotiation* phase, but if accepted, such an offer is *deployed* as the agreement that regulates the service provision. As can be seen, the customer initiates the interaction and therefore it plays the role *initiator* in the agreement creation process, whereas the provider plays the role *responder*.

Although the previous is the usual usage scenario, two additional considerations are included in WS–Agreement. First, the publication of a template is optional and thus, any party may send an agreement offer to other party without any template published. However, the acceptance of an agreement offer is more likely if it is defined based on a previously published template. Second, although the service provider usually acts as responder, WS–Agreement also allows consumers to play the role of responders by publishing templates with the service they intend to consume and some desired guarantees. In these cases, providers would initiate the SLA creation process by sending agreement offers including the service they provide and their capabilities.

![Diagram](image.png)

*Figure 3.1: A typical WS–Agreement interaction to create agreements*

### 3.4 Monitoring SLAs with WS–Agreement

One of the advantages provided by the SLA-driven solutions for the consumers is that providers are obligated to assure a certain service level by contract. In order to
control if the SLOs are being fulfilled or not by the parties, the SLA must be monitored by using suitable monitoring techniques at the *fulfillment* phase of the lifecycle. Such SLA fulfillment information would be used to perform the *termination* phase of the lifecycle that can include a compensation action (such as applying penalties or rewards according to the fulfillment information or adapting the SLAs for future service consumptions if too much violations were detected).

SLAs must provide the information of how SLOs and SPs must be monitored. Such an information is included in WSLA [61] documents by means of the *measurement directives* element. In turn, WS–Agreement provides a *metrics* for each service property that may include such a kind of measurement information, and the *scope* to denote which SLO must be monitored when a service operation is being providing. When an SLO is violated, the penalties and rewards included within the *business value list* elements of the same guarantee term apply (cf. Section §2.2.2).

WS–Agreement does not provide a technique to monitor the service subject of an agreement in order to check the fulfillment of WS–Agreement documents, but it considers several states of the SLAs and GTs. As depicted in Figure §3.2\(^1\), an SLA in WS–Agreement can be in one of the following states: Pending, PendingAndTerminating, Observed, ObservedAndTerminating, Rejected, Complete and Terminated; and as depicted in Figure §3.3, the GTs can be in one of the following states: NotDetermined, Fulfilled, Violated. Such states are thoroughly described in the recommendation [10] and we do not intend to explain them in this dissertation but to remark that an SLA or a GT would change its state by the use of an external monitoring technique that provided information about the current SLA and GT fulfillment.

The monitor component included in such external monitoring technique should consider some aspects related with the monitoring information that SLAs, including:

- The monitor must be aware of such a monitoring information in order to monitor specific SLOs and metrics when the scoped service operations are executed at runtime. In addition, if the guarantee terms have local periods specified they must be considered as the periods in which the terms must be monitored.

- The monitor must include enough monitoring logic to obtain the term state. Such a logic depends on the complexity of monitoring and measuring the SLOs and SPs. For instance, the monitor must compute the MUP of the AmazonS3 scenario, explained in Section §2.2.2, by subtracting from 100% the average of the *error rates*, i.e. internal server errors divided by the requests during each five minute period of a monthly billing cycle.

- Customers (or an intermediary third-party) are the typical responsible for monitoring in order to demonstrate SLA violations. For instance, a low MUP in the AmazonS3 scenario must be monitored and demonstrated by the customer, as in

---

\(^1\)The information included in Figures §3.2 and §3.3 is obtained from pages 34 and 37 of WS–Agreement specification [10], respectively.
other commercial services such as Google Cloud Storage\textsuperscript{2}, or RackSpace\textsuperscript{3}. In the AmazonS3 case, the customer must provide within ten business days after the end of the billing cycle in which the errors occurred, both: (1) the time instants in which it suffers the lack of service; and (2) a log including the requests and error messages provided by Amazon.

In the following Section §3.5 we include some information about some approaches to monitor WS–Agreement documents.

\textsuperscript{2}https://cloud.google.com/products/cloud-storage
\textsuperscript{3}http://www.rackspace.com/
3.5 **WS–AGREEMENT TOOLING SUPPORT**

The WS–Agreement specification has been successfully applied in the Grid computing arena within grids services in projects such as Assess Grid, a project funded by the European Commission [4], BEinGRID SLA negotiation component [33], UNICORE-VIOLA, of CoreGRID and Viola [40], or decentralized, crossmiddleware grid job submission service (JSS) [136], between others. Furthermore, The GRAAP Working group of the Open Grid Forum, leads by researchers of the Fraunhofer SCAI, developed WSAG4J, a framework supporting the WS–Agreement protocol and considering all phases of the SLAs lifecycle. In the following sections we provide details of WSAG4J framework and other approaches that uses the WS–Agreement specification for services that do not offer computational jobs.

3.5.1 **WSAG4J framework**

The WSAG4J framework included in the Figure §3.4 is designed as a generic SLA layer that can be used in a wide set of scenarios and it comprises the following modules:

![Figure 3.4: WSAG4J architecture provided by the authors at the project homepage](http://packcs-e0.scai.fraunhofer.de/wsag4j/server/architecture.html)

*API Module* contains interface definitions and implementations that are shared by

---

4 these modules are described at [http://packcs-e0.scai.fraunhofer.de/wsag4j/server/architecture.html](http://packcs-e0.scai.fraunhofer.de/wsag4j/server/architecture.html)
the different modules of the framework. **Client Module** is an implementation of the client API defined in the API module. It is implemented for accessing the WSAG4J web service stack. **SLA Engine Module** is the core of the WSAG4J framework. It provides a generic implementation of a WS–Agreement based SLA engine. It implements the standard functionality for processing agreement offers, creating agreements, monitoring the agreements runtime states, and evaluating and accounting agreement guarantees. Agreement acceptance policies and business logic for instantiating and monitoring SLA aware services can easily plugged in. **Web Service Module** implements the WS–Agreement port types and delegates the calls to the WSAG4J engine, if necessary. The web service module is implemented on the base of the Apache Muse framework. This framework provides the WSRF container required by the WS–Agreement specification. **Server Distribution** comprises the WSAG4J server including the web service stack, the SLA engine and all required configurations. It is a packaged web application archive which can easily be deployed in a wide set of application servers.

WSAG4J framework provides full support to grid services by considering documents the Job Submission Description Language (JSDL) as sublanguage for the service description terms. An example is included in the XML Document 3.1. The authors provide more examples available online. 

```xml
<wsag:Template xmlns:wsag="http://schemas.../ws-agreement">
  <wsag:TemplateName>UNICORE6-COMPUTE</wsag:TemplateName>
  <wsag:TemplateName>UNICORE6</wsag:TemplateName>
  <eng:WSAG4JSession eng...> <eng:SessionID>12115dfe963-419678bc139598cf</eng:SessionID>
</eng:WSAG4JSession>
</wsag:Context>
<wsag:Terms>
  <wsag:ServiceDescriptionTerm wsag:Name="APPLICATION_STD_1" ... "UNICORE6">
    <jsdl:JobDefinition xmlns:jsdl="http:...">
      <jsdl:JobDescription>
        <jsdl:Application>
          <jsdl:ApplicationName>WISDOM-PACK</jsdl:ApplicationName>
          <jsdl:ApplicationVersion>1.0</jsdl:ApplicationVersion>
          <jsdl-posix:Environment name="InputFilename">/input.dat</jsdl-posix:Environment>
          <jsdl-posix:Environment name="OutputFilename">/output.dat</jsdl-posix:Environment>
        </jsdl:Application>
      </jsdl:JobDescription>
    </jsdl:JobDefinition>
  </wsag:ServiceDescriptionTerm>
</wsag:Terms>
</wsag:Template>
```

---

5http://packcs-e0.scai.fraunhofer.de/wsag4j/server/guarantee_evaluation_example.html.
It is important to remark that WSAG4J implements\(^6\) the WS–Agreement, a domain-independent specification with a limited definition for the validity criteria of the WS–Agreement documents. Thus, as we detail at following, WSAG4J supports: checking if an offer is compliant with a template, and some guarantee states validations.

WSAG4J supports a dynamic agreement offer validation based on template creation constraints defined as XSD-based constraints\(^7\) as such included in Document 3.1. Such an offer validation follows the template compliance definition included in Section §2.5.

In addition, WSAG4J provides automatic guarantee evaluation based on guarantee states. The framework does not provide neither monitoring techniques, nor techniques to validate the violation of a guarantee term. Nevertheless, it supports to plug in a set of monitoring handlers\(^8\) that provide the domain specific logic to update the agreement service term states. This would allow to query the system if a specific GT is in the violated state or not.

The WSAG4J programming model, that is thoroughly described online\(^9\), allows an user to extend the WSAG4J framework with new SLAs, by specifying service instantiation techniques and service monitoring strategies. Extending the WSAG4J framework with a new SLA basically requires three steps: (1) definition of a SLA template, (2) implementation of a service instantiation and termination strategy, and (3) implementation of a service monitoring strategy.

### 3.5.2 Other Tools supporting WS–Agreement

The application of WS–Agreement to other kind of services that do not provide computational jobs is supported by the recommendation. Thus, several approaches can be found that provide some kind of tooling support either to create [26, 100] or monitor [27, 63, 103] WS–Agreement documents specified with their own WSACs.

---

\(^6\)A thorough description is included in [http://packcs-e0.scai.fraunhofer.de/wsag4j/index.html](http://packcs-e0.scai.fraunhofer.de/wsag4j/index.html).

\(^7\)Some examples of the supported item constraints are provided at [http://packcs-e0.scai.fraunhofer.de/wsag4j/server/creation_constraints_example.html](http://packcs-e0.scai.fraunhofer.de/wsag4j/server/creation_constraints_example.html).

\(^8\)In [http://packcs-e0.scai.fraunhofer.de/wsag4j/server/server_monitoring_example.html](http://packcs-e0.scai.fraunhofer.de/wsag4j/server/server_monitoring_example.html) some gaps are showed in the source code to put the domain dependent monitoring code in monitoring handlers.

\(^9\)at [http://packcs-e0.scai.fraunhofer.de/wsag4j/server/programming_model.html](http://packcs-e0.scai.fraunhofer.de/wsag4j/server/programming_model.html)
However, as we detail below they do not provide a full support for the WS–Agreement protocol.

The approaches that provides support for the SLA creation do not make use of the WS–Agreement template-offers protocol. On the one hand, Comuzzi et al. [26] propose the use of an ad-hoc SLA model that assure the consistency and compliance between consumers and providers requirements. When the parties reach an agreement in such ad-hoc SLA model, it is serialised to WS–Agreement specification but the complete serialisation process is not detailed by the authors. On the other hand, Oldham et al. [100] provide a technique to check the compliance between provider SLAs and consumer SLAs, and not between templates and agreement offers as WS–Agreement protocol establishes. In addition, they provide a technique that ranks a set of consumer SLAs by how the meet the requirements included in a provider SLA.

The approaches dealing with monitoring of WS–Agreement documents [63, 103] do not provide support for the states mentioned in WS–Agreement specification. Nevertheless, they provide techniques to monitor the services in order to check violations of SLAs specified with their own WSACs and thus, their proposed techniques could be used as the aforementioned monitoring handlers to complete the monitoring facilities of WSAG4J framework. Specifically, the approach of Palacios et al. in [103] use testing of WS–Agreement documents to detect potential violations for the terms before deploying the SLA. And the approach of Mahbub et al. in [63] is able to detect violations in the events that take place during the execution of web service compositions. Such events are included within terms of SLAs specified with WS–Agreement. In the following Sections §4.2.3 and §5.5.2 we provide more information about these approaches.

### 3.6 Summary

In this chapter we have introduced the different phases in which SLAs may be during its lifecycle with the aim of describing how these phases are considered in the agreement creation and monitoring protocol proposed by WS–Agreement. Regarding the SLA creation, we have provided the most common scenario of the WS–Agreement protocol and some additional considerations included in the specification. In turn, regarding the SLA monitoring we have cited the different states that agreements and guarantees may have during the WS–Agreement protocol. Such states ease the evaluation of agreements and guarantees at monitoring in order to detect violations. However, domain-specific monitoring techniques are not provided by WS–Agreement specification due to its domain-independent purpose. Such a domain-independence of the specification boost the appearance of a plethora of tools with a diverse support for the creation and monitoring of WS–Agreement documents. At the end of the chapter we have provided information about most important of these tooling support approaches.
The analysis of SLAs provides information about the validity of some SLA properties during its lifecycle. In this chapter we describe how the analysis of SLAs is dealt within the literature. After a brief introduction to the topic in Section §4.1, in Section §4.2 of this chapter, we introduce the foundations of SLAs analysis, that is, the issues to take into consideration when developing a method for analysing the SLAs. Next, we study the related approaches, assessing them on the basis of the foundations identified. The results of the evaluation are presented in Section §4.3. Finally, a summary of the content closes the chapter in Section §4.4.
4.1 **Introduction**

SLA analysis helps ensure the proper regulation of services that are subject of the SLAs. As far as the agreement documents are described, the terms and other kind of informations such as context information or creation constraints in WS–Agreement documents, must be analysed in order to ensure that the service consumption and the parties obligations are specified properly.

Automating this analysis may bring many benefits. At preparation phase of the SLA lifecycle, it allows investigating properties related to the information included in the agreement documents, and thus detect potential consistency problems that can be overcome beforehand by modifying the agreement document. For instance, the price policies of SLAs usually include formulae to compute the cost that requires the use of a spreadsheet or an ad-hoc calculator as such provided on-line by Amazon\(^1\) due to the complexity of computing the cost (cf. Figure §4.1). At information and negotiation phases, an automated analysis of agreement documents may help to find compliant counter-parties to make the process of reaching an agreement easier. At deployment phase, it ensures that the deployed SLAs are consistent to avoid further penalties. And finally, at fulfillment phase, it helps to detect SLA violations. Therefore, the automated analysis of the SLAs can be defined as the computer-aided extraction of useful information from SLAs.

![Figure 4.1: Online simple monthly calculator for AmazonS3 storage service](http://calculator.s3.amazonaws.com/calc5.html)

However, automatically analysing agreement documents implies taking into ac-
count the following issues that are explained in the section in detail.

1. It is often useful to define the analysis of models in terms of analysis operations that take a set of parameters as input, and return a result as output. It has been successfully done in other fields such as feature models [132] or Business Processes [20]. Thus, we first must come up with the set of analysis operations that we aim at supporting.

2. Another issue to consider is when the analysis is going to be performed, that is, in which phase of the SLA lifecycle we plan to execute the analysis operations.

3. Finally, we need to select or develop a technique or formalism to enable the automation of the analysis.

4.2 FOUNDATIONS

In the following, we define the basic issues to consider when facing the analysis of agreement documents, according to which we will compare the current approaches.

4.2.1 Analysis Operations

The analysis can be defined in terms of analysis operations that take a set of parameters as input, and return a result as output. We next describe the four analysis operations we have found in literature from a study we have conducted on the current approaches dealing with WS–Agreement documents analysis. Please notice that we focus on WS–Agreement documents as inputs as the reference SLA model used in the dissertation, but there are several approaches that use other SLA models as mentioned in Section §4.3.

Consistency Checking

A WS–Agreement document is consistent if it does not includes contradictions between one or more of its terms and or creation constraints in case of templates. This operation takes a WS–Agreement document as input, and returns whether it is consistent or not.

\[ isConsistent(WS-Agreement\ document) : Boolean \]

Figure §4.2 shows an inconsistent agreement document inspired in the AmazonS3 scenario. The inconsistency arises from a contradiction between the two terms at the bottom of the document.
CHAPTER 4. ANALYSIS OF SLAS

Compliance Checking

As defined in Section §2.5, an WS–Agreement offer is compliant with a template if the service terms of the agreement offer fulfill the template creation constraints. This operation takes two WS–Agreement documents, a template and an agreement offer as input, and returns whether they are compliant or not.

\[
\text{areCompliant}(\text{WS-Agreement Offer}, \text{WS-Agreement Template}) : \text{Boolean}
\]

Figure §4.3 shows a non-compliance between a template inspired by the AmazonS3 scenario and an agreement offer of a potential customer. The non-compliance of the example takes place due to the intention of the customer to pay less for a Not Reduced Redundancy storage service of AmazonS3.

Fulfillment Checking

A deployed WS–Agreement SLA is fulfilled if the comprised terms are not violated by the monitored data obtained at runtime. This operation takes three inputs, the monitored data from the real service consumption at runtime, the WS–Agreement SLA that
regulates the service consumption, and a monitoring period (MP) that in the simplest case is the SLA global VP; and returns whether the SLA is being fulfilled or not.

\[ isFulfilled(WS-Agreement\ SLA, \text{MonitoredData}, MP) : \text{Boolean} \]

Figure §4.4 shows an SLA violation due to the monitored number of read and write requests.

![Figure 4.4: The operation detects an SLA violation between the inputs](image)

**Unfulfillment Explanation**

The unfulfillment of a WS–Agreement SLA is considered as explained if the origin of the unfulfillment is pointed out. The unique approach we have found that provides explanation for SLA violations is based in an event-calculus technique [63, 124]. Such approaches store the service requests and responses as events. Thus, this operation takes as inputs both, the stored events and the WS–Agreement SLA that regulates the service consumption; and returns the violated terms as explanation.

\[ whyIsNotFulfilled(WS-Agreement\ SLA, \text{StoredEvents}) : \text{Terms} \]

Table §4.1 shows the violation of an event-calculus formula by two events: a request event with the \( t_1 \) timestamp and a response event with the \( t_2 \) timestamp. Such a formula is included in a sample of [63] and represents that the \( \text{ResponseTime} \) of the \( \text{getRate} \) operation, must be less than 100 ms. Figure §4.5 shows an example of this kind of violations explanation in the Amazon scenario.

**4.2.2 SLA lifecycle Phase considered in the Analysis**

Another issue to take into consideration in the analysis of SLAs is when we are carrying out the analysis, that is, at which phase of the SLA lifecycle (cf. Section §3.2). Furthermore, as the kind of WS–Agreement document involved in the analysis, as the analysis operations that result useful may vary depending on when the operation is performed. As detailed below, in each phase of the SLA lifecycle, the analysis has some implications.
ResponseTime < 100 in getRate operation using event-calculus

<table>
<thead>
<tr>
<th>Event-calculus Formula lines</th>
<th>Truth Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{forall } t_1 : \text{time} \exists t_2 : \text{time} \text{Happens(i}: \text{getRate(ID,country2,country1),} \ t_1,R(t_1,t_1)) \cap \text{Happens(i}: \text{getRate(ID),} \ t_2,R(t_1,t_2)) \cap \text{oc: self: sub(t_2,t_1)&lt;100}</td>
<td>-</td>
</tr>
<tr>
<td>\text{Happens(i}: \text{getRate(ID,country2,country1),} \ t_1,R(t_1,t_1))</td>
<td>\text{True}</td>
</tr>
<tr>
<td>\text{Happens(i}: \text{getRate(ID),} \ t_2,R(t_1,t_2))</td>
<td>\text{True}</td>
</tr>
<tr>
<td>\text{oc: self: sub(t_2,t_1)&lt;100}</td>
<td>\text{False}</td>
</tr>
</tbody>
</table>

\text{t_2 = 24657}  
\text{t_1 = 24500}

Table 4.1: ResponseTime < 100 in getRate operation using event-calculus [63]

Figure 4.5: The operation explains an SLA violation between the inputs

Analysis at the Preparation Phase

Analysing WS–Agreement documents at the very beginning of the lifecycle, when 
the document is being modelled, is crucial. At this moment, the parties usually 
describe templates including: (1) the provided or demanded service functionality; (2) 
the provided or demanded service level by means of defining some guarantees; and (3) 
some creation constraints that the other party will be obligated to fulfill in order to 
reach an agreement. As such information should be described according to the available 
resources in case of providers (or the needs of consumers), this preparation phase 
may become an iterative process in which some contradictions may appear between 
the guarantee terms and creation constraints. Thus, the consistency of the template 
should be checked when new information is included in the template at this phase. 
Working at this preparation phase, every potential problem detected can be overcome 
before negotiating or deploying the SLA.

Analysis at the Information and Negotiation Phases

Analysing WS–Agreement documents at the information and negotiation phase, when 
templates and agreement offers are interchanged between the parties is really
useful to avoid consistency and compliance problems. The consideration of an inconsistent agreement offer as potential agreement between the parties may carry problems at the next steps of the negotiation or even at the deployment phase of the SLA reached between the parties. Therefore, the following actions should be performed at information and negotiation phases:

On the one hand, the party who intent to prepare an offer from a previously published template, should not take the template consistency for granted. Thus, the template consistency must be checked before preparing the agreement offer to be sent to the template owner. When the agreement offer is prepared it must be checked for both: (1) consistency, to avoid potential errors introduced at the agreement offer description; and (2) compliance with the template in which it is based, to avoid sending a non-compliant offer that would have more probability to be rejected.

On the other hand, several agreement offers are received by the template owner and it should not take the offers consistency and compliance for granted. Thus, the offer consistency and compliance with the template must be checked before accepting or rejecting the agreement offers.

**Analysis at the Deployment and Fulfillment Phases**

Although the WS–Agreement documents had been analysed during the previous phases, analysing the WS–Agreement SLA before deploying it is mandatory to avoid an inconsistent agreement to be signed by the parties. If an inconsistent SLA is deployed, further problems related to the inconsistent information may appear because the parties expectations won’t be satisfied.

In case the deployed SLA is consistent, the SLA fulfillment must be monitored during the whole SLA lifetime. Such a fulfillment monitoring implies to have available monitoring techniques that collect information at runtime while the service is being consuming, in order to know the real quality of service provided. Once such provided quality of service is know, the fulfillment checking analysis operation can be executed to ensure that SLA terms are not being violated. Note that the fulfillment analysis should be performed periodically as a quality assurance test, or even just when new information about the provided quality of service is gathered.

In case a violation is detected, unfulfillment can be explained in order to know with information was violated and to react as a consequence. This reaction could imply to perform some actions that are out of the scope of the current dissertation such as: (1) to apply penalties or rewards, for instance in [112] the potential violation situations of terms and penalties are classified; or (2) to perform a proactive adaptation of the SLAs as proposed in [18, 45, 106, 119].
4.2.3 Analysis Technique

Depending on the decisions made with regard to the other elements such as the analysis operation to support, or the phase of SLA lifecycle, we will have more or less freedom to select the type of automated analysis technique(s) to use. Let us summarize some mechanisms for automated analysis identified in the literature (dealing with WS–Agreement and other SLA specification models):

- **Constraint Satisfaction Problems (CSP)** [135] are used to model the SLAs as variables with an allowed domain, and constraints on such variables. By using such a CSP modelling, some approaches provide an operational semantics to analyse the consistency, compliance and other analysis operations.

- State-search and model-checking is used to provide operational semantics to SLA models that can be checked for consistency, e.g. by using Maude-based tools [25].

- **SWRL** [139], a semantic web rule language combining OWL and RuleML, is used to define an operational semantics for the compliance between SLAs modelled by using semantic languages such as OWL-S [128] or WSDL-S [141].

- Description logics is used to describe web services requests and offers, that can be translated to agreement offers and templates, respectively. And such description logics models can be used to provide an operational semantics for the compliance between documents.

- **Event-Calculus (EC)** [121], which is a first-order temporal logic language, is used to describe assertions inside WS–Agreement documents and to define an operational semantics for the fulfillment checking by using inference rules of first-order logic. Such rules are used as inside guarantee terms, as to define the expected system behaviour in order to support the detection of: (1) violations of terms by the recorded behaviour of a system; and (2) violations of service guarantee terms by the expected behaviour of the system.

- **Timed automata** [9] are used to encode the violations of requirements included in the SLAs. Then, to identify a violation it is sufficient to pass all the occurring events to the automaton and verify if a final state is reached.

- **UML and OCL** are used to model SLAs and to provide semantics to the compliance between SLAs.

- **Ad-hoc algorithms** are used by several authors to: (1) check the template compliance of an agreement offer in WSAG4J framework [96]; (2) create SLAs without inconsistencies [26]; (3) audit whether the execution of a web service is compliant or not with an SLA [49]; (4) check the fulfillment of variables included in the SLAs in [22, 101].
4.3 CURRENT APPROACHES

In the following, we briefly summarize some approaches dealing with the analysis of the WS–Agreement and other SLA specifications, and classify them according to the foundations introduced in Section §4.2.

Ruiz et al. in [116] present a CSP-based approach to automate the consistency and conformance checking in the context of the web service procurement and using the QRL language mentioned in Section §2.6. In addition, the work also provides the optimal offer for a web service demand by using a weighted composition of utility functions. The use of QRL allows this work to describe expressive requirements that can be mapped to most of the WS–Agreement elements. For instance, the QCs could be mapped to an implication in which the antecedent is the QC and the consequent is the SLO. However, QRL lacks of a way to compose several requirements as TCs do with terms in WS–Agreement. Nevertheless, some important conclusions can be made from the CSP-based proposed in this work, and we highlight at followings two of them: (1) any web service specification can be interpreted in terms of a CSP by considering the service parameters, its domains, and the set of demanded and offered service requirements as the CSP variables, its domains, and the CSP constraints, respectively; and (2) a symmetric technique supports offers and demands written in the same language and that language includes other logical operators apart from the equality operator. Such a symmetry notion was adopted, later, in the European Network of Excellence in Software Services and Systems (S-Cube)² [70].

Braga et al. in [17] present a quality of service (QoS) Tool that transforms QoS contract descriptions into modular structural operational semantics (MSOS) specifications. MSOS specifications are executable and analyzable in the Maude MSOS Tool, which uses efficient rewriting to execute, search and model checking MSOS specifications to check the consistency of the model. The QoS contract descriptions are a simple SLA model that includes properties, its allowed values, and they support the description of QoS requirements using logical operators and not only the equality operator. However, some WS–Agreement elements cannot be expressed such as the QCs, TCs, CCs, or business value list.

Oldham et al. in [100] propose the use of OWL ontologies to describe WS–Agreement SLAs but neither templates nor agreement offers. They present an SWRL-rules-based approach [139] to check the compliance between provider SLAs and consumer SLAs by the ABLE rule language and the IBM’s ABLE engine [15]. As Ruiz et al. in [116], they assure the symmetry (in both provider and consumer) and they also provide the best provider SLA for a given consumer SLA. However, the compliance is checked between SLAs and not between templates and agreement offers as WS–Agreement establishes. As a consequence, the kind of compliance they support checks provider and consumer SLA terms (instead of template CCs against offer terms as the mentioned

²http://www.s-cube-network.eu/
in the specification). Furthermore, they only support the ExactlyOne TC that allows alternative sets of terms, and not the OneOrMore TC that allows optional sets of terms.

Di Noia et al. in [94, 95] propose the use of DL to describe web services offers and demands by including axioms inside TBoxes, that represent the allowed values for the service properties. Such DL-based web service offers and demands can be checked for compliance by following an ad-hoc algorithm that make use of some concepts such as satisfiability and subsumption, both related to the TBoxes axioms. In addition, they also provide the best offer for a set of demands, as proposed by Ruiz et al. and Oldham et al. in [100, 116], respectively. The advantage of the Di Noia’s work is they provide explanations when a service offers does not match with a service demand. As we comment for Ruiz’s work, the offers and demands can be considered as templates and agreement offers of WS–Agreement. However, Di Noia et al. do not provide support for WS–Agreement elements such as, QCs, TCs, and CCs.

Mahbub et al. in [63, 124] propose an event-calculus technique that provides not only detection, but also explanations of violations modelled as events in WS–Agreement SLAs. However, such events, which are used in the specification of the SDTs and GTs, are restricted to those which can be observed during the execution of the composition process of a service-based system (SBS) and not for a single service. This set of events is determined by a static analysis of the BPEL composition process of the SBSs. The proposed technique monitors different types of deviations from GTs including: (1) violations of terms by the recorded behaviour of a system and (2) violations of GTs by the expected behaviour of the system. Therefore, the proposed technique requires to: record the system behaviour, and/or specify the expected system behaviour. Despite Mahbub’s work supports expressive EC-Assertions for the GTs, they are not human-understandable and some WS–Agreement elements are not supported such as, the CCs, or the business value list to define penalties or preferences.

Raimondi et al. in [111] propose a timed automata-based approach to create monitors, automatically from SLA conditions that are able to detect SLA violations. The SLA model they support is SLAng [29], that is defined on an abstract modelling level in terms of meta-models in Essential Meta Object Facility (EMOF) notation. It applies the Object Constraint Language (OCL) to specify quality constraints. However, several WS–Agreement elements are not supported such as QCs, TCs, and business value lists.

Skene et al. in [123] provide a precise semantics for SLAng [29] SLAs that support to check the compliance between SLAs, that can be considered as the compliance between templates and agreement offers. The provided compliance notion is similar to the conformance defined in QML [42] and QRL [115]. As in Raimondi’s work, they support SLAng, and thus, the proposed technique does not support QCs, TCs, and business value lists of WS–Agreement documents.

Wäldrich et al. in the WSAG4J framework [96] provides an implementation of the template compliance checking established in WS–Agreement as mentioned in Section §3.5.1.
Comuzzi et al. in [26] propose an ad-hoc technique to create consistent WS–Agreement SLAs. They do not use the templates and agreement offers to create such an SLA, but they propose the use of ad-hoc consumer requirements and provider offers in order to create consistent SLAs. Nevertheless, the WS–Agreement specification is used to serialise the final SLA achieved. As they do not use templates, WS–Agreement elements such as the CCs are not supported, but they do not support neither QC, nor TCs.

Chen et al. propose in [22], a monitoring technique that support checking the fulfillment of an ad-hoc SLA model. The novelty of the proposed monitoring technique is that rather than monitoring runtime service metrics or soliciting consumers to provide feedback, they adopt mechanisms to enforce the provider to obligatorily monitor and report its runtime status to the manager. Deceptive providers must be inhibited from intentionally reporting fake information. Aiming at this problem, they propose an aspect oriented style to dynamically weave monitoring logic into the service provider. Although the ad-hoc SLA model supports expressive assertions for the SLOs, they do not provide support for WS–Agreement elements such as the QC, TC, and business value list.

Oriol et al. propose in [101] a monitoring technique that support checking the fulfillment of an ad-hoc SLA model. The technique is able to monitor the services at runtime in order to achieve real measures of properties such as the response time, availability, and several aggregate metrics. Such a technique provides: (1) support for SOAP and REST services; (2) extensibility to monitor new metrics; and (3) a combination of passive monitoring and on-line testing [117].

Palacios et al. propose in [104] a testing monitoring technique aiming to react at the preparation phase of the lifecycle against a potential SLA unfulfillment that may take place at the fulfillment phase. Such a technique supports expressive WS–Agreement documents and generates a set of test cases considering the SLA terms that are checked in order to detect the potential SLA violations. The same authors in [105] analyse how to monitor guarantee terms of WS–Agreement documents.

In Table §4.2, we include a summary of these approaches including the supported SLA model, the analysis operation they provide, the SLA lifecycle phase in which they provide the analysis, and the used technique. In such a table, “✓” and “∼” denote full and partial support for an analysis operation or the WS–Agreement specification.

4.4 SUMMARY

In this chapter, we have introduced the foundations to be considered in the analysis of SLAs, and we have evaluated the existing approaches dealing with SLAs analysis with respect to those foundations, to be named: (1) the analysis operations supported; (2) the SLA lifecycle phase(s) at which the analysis is carried out; and (3) the formalism
or technique used to perform the analysis. The result of this study will be used to understand the existing weak points or problems in SLAs analysis (cf. Section §5.3), and they will thus constitute goals to be achieved with the solutions proposed in this doctoral thesis.
### Analysis Techniques Provided in the Literature

<table>
<thead>
<tr>
<th>Approaches</th>
<th>SLA Model</th>
<th>Checking</th>
<th>Explanation</th>
<th>Lifecycle Phase</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Cons.</td>
<td>Compl.</td>
<td>Un/fulfil.</td>
<td></td>
</tr>
<tr>
<td>Braga [17]</td>
<td>ad-hoc</td>
<td>✓</td>
<td></td>
<td></td>
<td>Model-check</td>
</tr>
<tr>
<td>Comuzzi et al [26]</td>
<td>~ WS–Agreement</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>ad-hoc</td>
</tr>
<tr>
<td>A. Ruiz [116]</td>
<td>QRL</td>
<td>✗</td>
<td></td>
<td></td>
<td>CSP</td>
</tr>
<tr>
<td>Oldham [100]</td>
<td>~ WS–Agreement</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>SWRL rules</td>
</tr>
<tr>
<td>Di Noia [94, 95]</td>
<td>DL-model</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>ad-hoc</td>
</tr>
<tr>
<td>Raimondi [111]</td>
<td>SLAng</td>
<td>✓</td>
<td></td>
<td></td>
<td>timed-autom.</td>
</tr>
<tr>
<td>Skene [123]</td>
<td>SLAng</td>
<td>✓</td>
<td></td>
<td></td>
<td>ad-hoc</td>
</tr>
<tr>
<td>Chen [22]</td>
<td>ad-hoc</td>
<td>✓</td>
<td></td>
<td></td>
<td>ad-hoc</td>
</tr>
<tr>
<td>WSA4G [96]</td>
<td>WS–Agreement</td>
<td>✓</td>
<td></td>
<td></td>
<td>ad-hoc</td>
</tr>
<tr>
<td>Mahbub [63, 124]</td>
<td>~ WS–Agreement</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>EC</td>
</tr>
<tr>
<td>Palacios et al [104, 105]</td>
<td>WS–Agreement</td>
<td>✓</td>
<td></td>
<td></td>
<td>ad-hoc</td>
</tr>
<tr>
<td>SALMon [101]</td>
<td>ad-hoc</td>
<td>✓</td>
<td></td>
<td></td>
<td>ad-hoc</td>
</tr>
</tbody>
</table>

Legend: ✓ support, Cons.: Consistency, non-/Compl: non-/Compliance, Un/fulfil: Un/fulfillment, ~-support: partial support, P: Preparation, I: Information, N: Negotiation, D: Deployment, F: Fulfillment

Table 4.2: Support provided by the existing analysis techniques
Part III

Our Proposal
Some men have thousands of reasons why they cannot do what they want to, when all they need is one reason why they can.

Willis R. Whitney (1868–1958), American chemist, founder of the research laboratory of the General Electric Company

In this chapter, we motivate our contributions. After a brief introduction to the chapter goals in Section §5.1, we describe the problems found in the literature in the approaches dealing with SLA specification, automated analysis, and lifecycle support in Sections §5.2, §5.3, and §5.4, respectively. In Section §5.5 we analyse the support provided for such proposals. An overview of our proposed solution is included in Section §5.6. Finally, a summary of the content closes the chapter in Section §5.7.
CHAPTER 5. MOTIVATION

5.1 INTRODUCTION

This dissertation aims to improve the current support to develop SLA-driven solutions focused in WS–Agreement recommendation. The goal of this chapter is threefold. First, to clarify the problems that we found in the literature regarding: (1) the SLA specification; (2) the SLA analysis techniques; and (3) the SLA lifecycle support. Second, to motivate the need for an enhanced analysis to develop SLA-driven solutions by considering real-world WS–Agreement documents and by using analysis operations not supported by current approaches. And third, to provide an overview of the solution proposed in our work.

The contributions of this dissertation result from: (1) an analysis of the WSACs used in the literature, the proposed analysis techniques and supporting tools; and (2) the application of the knowledge gained in previous experiences with the automated analysis of web services matchmaking [68, 115], feature models [12, 132], and BPMN [1, 20].

5.2 PROBLEMS RELATED TO THE SLA SPECIFICATION

One of the first problems that must be faced in order to provide support for SLA-driven solutions that use WS–Agreement documents is to decide how WS–Agreement documents should be described. Despite WSLA [61], one of the SLA specifications that inspired WS–Agreement, proposed a fully–fledged agreement language with specific syntaxes for the documents that were found appealing for several researchers [6, 93, 122], WS–Agreement does not provide such syntaxes. Instead, as commented in Sections §2.2 and §2.3, it defines a schema that can and must be completed with an WSAC, that is a set of sublanguages to specify several parts of the agreement document such as service description terms or service level objectives. Many different languages such as JSDL [5] (Job Submission Description Language) and the WSLA [56] expression language, have been used for this task since WS–Agreement does not impose additional constraints on their expressiveness.

Wäldrich et al. state in [11] that this extensibility is a great advantage of WS–Agreement since it make it possible to use the specification in an unbounded number of different ways. However, they also realise that most projects implementing the specification\(^1\) use only a fraction of WS–Agreement and they expose the hypothesis of it results from the problem that WS–Agreement does not provide sufficient instructional information on how to use it.

In this dissertation we do not intend to improve the WS–Agreement specification text providing more instructional information, but to tackle the drawbacks found in the implicit WSACs proposed in the literature (included in Table §2.1 of Section §2.3).

\(^1\)They studied the projects included in [35]
Such drawbacks were found while modelling real-world SLAs of Amazon, Google, PayPal, Renderfarms and others (up to 8)\(^2\) and they comprise the lack of: (1) support for some WS–Agreement elements, (2) support for high and extensible expressiveness, (3) domain-independence, (4) time awareness, (5) human-readability, and (6) validity criteria for the documents.

### 5.2.1 Lack of Full-WS–Agreement Support

Most of the implicit WSACs proposed in the literature do not provide a precise semantics for term compositors (TCs) and the scope of terms as explained in the following sections.

**Lack of Support for Variants**

Only three [40, 96, 100] of the ten studied WSACs provide a limited support for TCs that allow the description the agreement variants mentioned in Section §2.4. However, the variants are needed in real-world SLAs to describe alternative, and optional terms sets such as the customer support plan facilities offered by Amazon that are depicted in terms G10.1-G10.4 of Figure §5.1 that is an excerpt of the template of Figure §2.2. Such terms must be optional and not alternative in order to support the consumers to select an unbounded number of them. The customer support plan price would be calculated according to the selected variant.

![Template](amazon_s3_version_1.0.png)

**Figure 5.1:** Optional terms sets described by using an optional terms compositor

On the contrary, in the example of Figure §5.2 two predefined customer support plans comprising specific set of alternative terms are described by means of an *ExactlyOne* term compositor. While one of the predefined customer support plan comprises the online reporting support and a turn around time to solve problems of 15 minutes, the other customer support plan comprises the phone support and an extended support.

Note that the terms compositors provides in the example several variants that represent the different customer support plans provided by Amazon. How these variants must be considered in the agreement creation and monitoring processes must be revised and a precise semantics should be provided.

\(^2\)Included in a SLA repository [http://193.147.175.241/ada/downloads/sla_repository](http://193.147.175.241/ada/downloads/sla_repository)
Figure 5.2: Alternative terms sets described by using an alternative terms compositor

Lack of Support for Terms Scopes

The implicit WSACs proposed in the literature do not provide a precise semantics for the scope of terms that, as mentioned in Section §2.2 is established by WS–Agreement as the service operations affected by a guarantee term (GT). We consider that such an information is needed to determine which terms may be violated when a service operation is being consuming at runtime. In addition, the service properties related by the SLO of a GT whose scope is an specific service operation are those properties that must be monitored when such a service operation is being consumed at runtime. For instance, the excerpt of Figure §5.3 includes three GTs that denotes the need to monitor the ResponseTime property when the getData operation were executed at runtime, and the MUP property when the storeData operation were executed at runtime.

Figure 5.3: Excerpt of an AmazonS3 template with scoped terms
5.2. Problems Related to the SLA Specification

5.2.2 Lack of Support for Suitable Predicates

Most of the implicit WSACs proposed in the literature do not support expressive enough and human-readable arithmetic-logic predicates for WS–Agreement elements such as SLOs, QCs and CCs. Only three [63, 96, 100] of the ten studied WSACs support expressive predicates for SLOs and QCs but not for CCs. However, while modelling the price policies of the AmazonS3 scenario we found that it is needed expressive creation constraints such as: \( \text{RRS} = \text{true} \) implies \( \text{StoragePrice} = 0.05 \times \text{StorageSize} \). In addition, the predicates used in [63] are not human-readable because they are thought to be handled by experts in event-calculus assertion formulae as such depicted in Table §4.1 of Section §4.2.1. The problem with this predicates is not only the language used for them, but also the relation between the predicates of SLOs, QCs, and CCs and with other sublanguages of the WSACs to describe the service properties and the metrics in catalogues or in the context. The relation between such sublanguages must be considered in the development of techniques to validate and debug such predicates (see the problem mentioned in Section §5.2.6). Moreover, the set of expressive predicate languages used in the WSACs proposed in the literature are limited by ontologies [100] or XML-based schemas [96] and the problem is that new kind of predicates could be required in future and the languages will have to consider them. Thus, we need not only expressive predicates, but also extensible to be adapted in future.

5.2.3 Lack of Domain-independent WSACs

Most of the implicit WSACs proposed in the literature are domain-dependent because most of them are projects developed to ease the grid computing, i.e. they provide suitable sublanguages for grid services but not necessarily to other domains. Thus, either they do not provide support for describing agreements on other domains, or they must be adapted to do it. For instance, the implicit WSAC proposed in [96] establishes the JSDL to describe grid services that is not suitable to describe the AmazonS3 service because JSDL focuses on the number of CPUs, processing time, and such a kind of service properties for the grid computing services. Another example can be found in the implicit WSAC proposed by Mahbub et al. in [63], in which an event-calculus assertion language is used for the SLOs and QCs. Such a language is specially suitable for events that occurs between the component services of composite web services. However, it results complex to apply it for events in single services such as the AmazonS3 storage service. We conclude that a suitable WSAC should comprise domain-independent sublanguages supporting to define mappings from other domain-dependent sublanguages as the model transformation approach that has been recently presented in [52].

5.2.4 Lack of Time-aware WSACs

The implicit WSACs proposed in the literature only supports the description of a validity period (VP) for the whole WS–Agreement documents, as established in the
recommendation. However, while modelling the AmazonS3 scenario we found the necessity of describing VPs affecting not the whole, but part of WS–Agreement documents. For instance, despite the AmazonS3 service price is usually established depending on the service properties selected by the consumers, Amazon provides a free trial quota period for a year for new consumers would change the price policy as follows: TotalPrice = 0 during the first year. After such a period, the service price would be applied as usual. Furthermore, while modelling other SLAs we realised about the necessity for disjoint and periodical VPs as such mentioned in Section §2.7.

5.2.5 Lack of Human-readable WSACs

Since WS–Agreement provides an XML-based schema, the implicit WSACs proposed in the literature to complete such an schema are comprised of XML-based languages. For instance, the aforementioned JSDL to describe services in [96], and the WSLA predicate tags to describe the SLOs in [100]. Ideally, a WSAC should provide end-user languages in order to approach to the natural language used in commercial SLAs of companies as Amazon or Google. But a first step is to provide at least human-readable languages avoiding the use of XML tags.

5.2.6 Lack of Validity Criteria

The implicit WSACs proposed in the literature do not provide enough validity criteria for WS–Agreement documents because WS–Agreement specification just provide a limited semantics for the consistency of a WS–Agreement document, and for the template compliance of an agreement offer. The consistency property of WS–Agreement documents provided by existing approaches only considers inconsistencies that affects the whole WS–Agreement document (cf. Section §4.2.1). In turn, the template compliance of an agreement offer only validates the offer terms as established by the specification (cf. Section §5.5.1). A suitable validity criteria must be developed including the assurance of more properties of the WS–Agreement documents such as: (1) the consistency of single WS–Agreement documents taking into account inconsistencies that affects part of the document; and (2) a stronger compliance notion between agreement offers and templates considering all terms of both documents to be compliant with the template creation constraints, and not only the offer terms. Such a validity criteria would allow to check and explain the semantic errors that a WS–Agreement document may include.
5.3 Problems Related to the Automated Analysis

As detailed in Chapter §4, several works can be found in the literature proposing analysis techniques to extract useful information from SLAs. The most commonly proposed techniques aim to check the SLA consistency, the compliance between agreement documents, and even to check the SLA fulfillment and provide some explanation when an SLA is violated at monitoring time. However, they present some of the following problems that can be found in emerging techniques.

5.3.1 Lack of a Fully-functional Reference Implementation

They do not provide a fully-functional reference implementation due to the reduced facilities to validate agreement documents. For instance, (1) they do not provide explanations or diagnostics to identify the inconsistencies, and (2) they provide false positive at checking the consistency due to they do not consider new types of conflicts, different from the inconsistencies, that arise when using QCs and TCs. These new types of conflicts appear in optional parts of the SLA without invalidating the whole document and we refer them as optional conflicts in this section by simplicity.

5.3.2 Lack of a Reuse-oriented Design

The proposed techniques lack of a reuse-oriented design to share the existing know-how. The proposed analysis techniques have been developed from scratch instead of using a combination of simpler problems with known solutions. For instance, the SLA fulfillment checking can be formulated in terms of the previously developed techniques to check the compliance or consistency of WS–Agreement documents (cf. Section §8.2.2). Thus, the isFulfilled operation mentioned in Section §4.2.1 could be solved in terms of the areCompliant operation of Section §4.2.1 as follows:

\[
(isFulfilled(WS-Agreement SLA, MonitoredData, MP) : Boolean) =
\]

\[
= (areCompliant(createOfferFromData(MonitoredData),
\]

\[
createTemplateFromSLA(WS-Agreement SLA)) : Boolean; during a MP)
\]

Where createOfferFromData and createTemplateFromSLA are operations that create spurious documents from the monitored data and the SLA, respectively, in order to apply the compliance checking.

\[3\text{In the following chapters they were coined as conditionally inconsistent terms, dead terms, and partial inconsistencies.}\]
5.3.3 Lack of Publicly-available User-friendly Interfaces

The existing analysis techniques lack of publicly-available user-friendly interfaces to proof their capabilities.

5.4 Problems Related to the SLA-driven Solutions Support

The existing approaches do not provide an appropriate support for the development of SLA-driven solutions due to the following drawbacks.

5.4.1 Lack of Validation at Agreement Creation and Monitoring

The approaches do not provide mechanisms to validate the documents in the agreement creation and monitoring processes. That means to validate the consistency and compliance of the documents: (1) while the documents are being prepared, (2) during the agreement is being negotiated, and (3) when the agreement is deployed to regulate the service consumption. Moreover, the existing proposals do not provide explanations for possible semantic problems that the documents may include.

5.4.2 Lack of an Integrated Development Environment

The existing approaches lack of an integrated development environment providing edition, creation, monitoring, and analysis facilities into the same tool. Such a lack can be a consequence of the lack of interoperability between the existing SLA analysis tools.

Although the extension points provided by the specification are a great advantage of WS–Agreement, as usual, this advantage comes with a price for WS–Agreement users: the difficulty of creating an interoperable tooling support since such tools should deal with the great variety of sublanguages that can be used in different WS–Agreement documents. This problem gets even harder for tools with analysis facilities since the tools should understand not only the sublanguages syntaxes, but also their semantics. In fact, the application of WS–Agreement specification to SLA-driven solutions that do not deal with grid services has not achieved the maturity level of SLAs in Grid computing arena in terms of tooling support and commercial applications. We consider as a possible cause the unbounded nature of services regulated in SLA-driven solutions (composite or single services, storage services, cloud services, translation services, etc) that are specified with many different specification languages, against the bounded nature of grid computing services that are usually defined with JSDL.

We have also realised that several approaches use WS–Agreement with a different
5.5 Analysis of Current Solutions

In this section we analyse the solutions provided in the literature for the aforementioned problems. Note that the same proposals that deal with the automated analysis provide somehow a support for SLA-driven solutions at any phase of the SLA lifecycle. Therefore, we analyse in the same Section §5.5.2 the solutions provided for the drawbacks mentioned in Sections §5.3 and §5.4.

5.5.1 Solutions Dealing with SLA Specification

Table §5.1 collects the implicit WSACs studied in Section §2.3, indicating for each of them whether they support: (1) the required sublanguage for describing the different elements of WS–Agreement documents; and (2) the problems mentioned in Section §5.2. We use a small tick ‘✓’ if the sublanguage or the problem is completely supported, and the symbol ‘∼’ denotes a partial support for such a sublanguage or problem.

The table reveals that none of them supports neither each and every sublanguage, nor a solution for all the aforementioned problems. WSAG4J [96], SWAPS [100], and the implicit WSAC used by Palacios et al. in [104, 105], are the approaches that provide more support for the aforementioned problems. WSAG4J uses a WSAC with grid-oriented sublanguages such as JSDL [5] to describe the services, XSD-Schema for the service properties metrics and creation constraints, and Java Expression Language (JEXL) [41] to describe suitable predicates for SLOs and QCs. They also support terms compositors to define agreement variants. However, such a WSAC of WSAG4J lacks of: (1) expressive CCs described by using suitable predicates as such mentioned in §5.2.2; (2) a precise semantics for the terms scopes, and an extension to describe validity periods applied to any agreement element.

In turn, the WSAC called SWAPS [100], proposes the use of WSDL-S [141] and OWL [143] to describe the services and their properties, the predicate tags of WSLA [61] are used for expressive SLOs and QCs. In addition, OWL-time [142] is used to describe some temporal properties. For instance, it is provided the validity period: dayOfWeek equals weekday, for an SLO limiting the response time to 14 seconds or less. SWAPS supports terms scopes to define affected service operations by the guarantee terms. However, it lacks of a sublanguage to specify: extended information inside the context, business values, and predicate-based creation constraints. Furthermore, SWAPS only provides a precise semantics for the ExactlyOne term compositor that they call alternative set of guarantees. However, the variants are needed in real-world SLAs to
### Table 5.1: Support for Sublanguages and Problems of the implicit WSACs proposed.

The implicit WSAC used in the work of Palacios et al. in [104, 105] uses ad-hoc sublanguages for all supported WS–Agreement elements. They support expressive predicates within the SLOs and QCs. However, since they dealt with SLAs, they do not consider CCs of templates. In addition some WS–Agreement elements are not supported in all of their works, such as the penalties, variants, and the expiration time to describe a global period. For instance, the variants are not considered in the approach that analyse the GTs to identify test requirements. Moreover, the unique validity criteria they provide is the fulfillment checking at preparation time by means of testing techniques.

The proposed WSACs do not provide neither end-user nor human-readable sublanguages avoiding the use of XML elements. Although providing end-user syntaxes
for the WSAC sublanguages would be really appealing for non technical users, we consider interesting to provide human-readable syntaxes, as a first step. In further research (cf. Section §9.2) we will study how to apply linguistic patterns to make more understandable the WSAC sublanguages for the end-users. Similar approaches have been successfully adopted to define quality requirements in [115], process performance indicators [31] of business process management systems.

None of the proposed WSACs provide enough validity criteria for the WS–Agreement documents as mentioned in Section §5.2.6. From the approaches that provide some validity criteria, in [26, 63, 96] it is only provided support for inconsistencies that affect the whole agreement documents (cf. Section §4.2.1), and not to a part of them; and in [96, 100] it is only supported the basic compliance definition provided in WS–Agreement and detailed in Section §2.5. However, we consider that such basic compliance definition, summarised with discontinuous arrows in Figure 5.4(a), is not enough and a new stronger definition is required.

The reason for such a strong compliance definition, whose extension is summarised with continuous arrows in Figure 5.4(a), is that the basic compliance definition does not state anything about the terms of the template. In other words, the party that creates the agreement offer may ignore the terms specified in the template. The problem with this basic definition is that the template creator can specify terms in the template, but the party that creates an agreement offer based on such template cannot do anything with them because the definition of compliance does not provide any semantics with regard to them. Thus, it is unknown for the party that creates the agreement offer whether the terms of the template specify default values, or preferred values, or mandatory values, or any other meaning.

As result of our experience developing SLA-driven solutions we have identified three non-compliance situations that must be checked within the new strong compliance definition, namely:

1. When the guarantee terms of an agreement offer (O) contradict the template (T) guarantee terms. We proposed in [71] an extended definition of compliance that considered such a non-compliance situation. This non-compliance may occur in the real world if the party that prepares the agreement offer modifies some of the template terms. For instance, while in the agreement offer of Figure 5.4(b) the initiator party demands: ReadRequests < WriteRequests, the responder establishes in the template the opposite assertion: ReadRequests > WriteRequests.

2. When the guarantee terms of an agreement offer are more restrictive than template guarantee terms. As in the previous case, this non-compliance may occur in the real world if the party that prepares the agreement offer modifies some of the template terms. For instance, while the template of Figure 5.4(b) published in a guarantee term MUP >= 99.9, the agreement initiator required in the offer a more restrictive guarantee term detailing MUP >= 99.99. Note that the basic compliance notion defined in WS–Agreement the specification is observed in the
example. However, probably the offer is not appealing for the template publisher because it is obligated to assure a higher MUP that the initially provided. Thus, it could be said that the received offer is not compliant with the template published expectations.

3. When the guarantee terms of a template are less restrictive than its own creation constraints. This non-compliance occurs when the template guarantee terms are non-compliant with the template creation constraints. For instance, the template of Figure 5.4(b) includes the guarantee term $\text{StorageSize} \leq 6000 \ \text{TB}$, that is less restrictive than a creation constraint. Note that although the $\text{StorageSize} \leq 6000 \ \text{TB}$ term does not contradict the $\text{StorageSize} \leq 5000 \ \text{TB}$ CC, the template creator should consider it as unacceptable because the own template terms are not compliant with their creation constraints.

5.5.2 Solutions Dealing with Automated Analysis and SLA-driven Solutions Support

The existing analysis techniques mentioned in Section §4.3 are summarised in Table §5.2. In such a table it is depicted that none of the approaches supports checking the optional conflicts. Note that an approach that only check inconsistencies that affect the whole agreement may incorrectly consider an SLA as conflict-free if such an SLA only has optional conflicts. Thus, new analysis techniques must be developed considering the optional conflicts to improve the analysis of WS–Agreement documents. Furthermore, the explanation of conflicts must be provided not only at monitoring time for SLA violations (see unfulfillment explanation the column in Table §5.2 and Section §4.2.1), but also at any phase of the SLAs lifecycle. Thus, the explanations of each and every kind of conflict that may be included in agreement documents would provide useful information for the parties from the preparation to the termination phase of the lifecycle. Note that empty columns in the conflicts explanation section of the Table §5.2 denote this lack of support. Moreover, the explanation provided by Di Noia et al. in [94, 95] for non-compliance scenarios is focused in web services matchmaking and not in SLAs. And furthermore, the explanation provided for SLA violations in [63, 124] and [54] can and must be enhanced as we detail at following in the section.

In addition, it is remarkable that the existing analysis approaches do not provide: (1) reusability due to they have been developed from scratch, (2) a fully-functional reference implementation to validate WS–Agreement documents, and (3) extensibility to develop new analysis techniques.

Inconsistencies Explanation

When the consistency checking operation returns that the WS–Agreement document includes inconsistencies, usually it is not easy to discover the origins of such inconsistency. Since, the inconsistencies are contradiction between agreement terms, and creation constraints, in case of analysing templates (cf. Section §4.2.1), returning
5.5. ANALYSIS OF CURRENT SOLUTIONS

(a) Basic and strong template compliance definitions

(b) An example of strong non-compliance between documents

Figure 5.4: Compliance between templates and offers
such contradictory terms and/or creation constraints, (referred as agreement elements for the sake of simplicity), would be a useful explanation for the document inconsistency. This kind of explanation is really appealing for the template designer to solve the problem at the preparation phase of the lifecycle, but also for the parties at negotiation phase because the consistency of the documents to be interchanged with the other party can be assured.

An inconsistency can be considered as explained if the contradictory agreement elements are pointed out as the origin for such inconsistency. This operation takes a WS–Agreement document, and returns a minimal subset of contradictory agreement elements.

\[
\text{whyExistInconsistencies}(\text{WS-Agreement document}) : \text{AgElements}
\]

For instance, in the AmazonS3 template of Figure §5.5 the highlighted term and creation constraint would be returned as inconsistency explanation to the template

### Analysis Techniques Provided in the Literature

<table>
<thead>
<tr>
<th>Approaches</th>
<th>SLA Model</th>
<th>Checking</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Braga [17]</td>
<td>ad-hoc</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Comuzzi et al [26]</td>
<td>~ WS–Agreement</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>A. Ruiz [116]</td>
<td>QRL</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Oldham [100]</td>
<td>~ WS–Agreement</td>
<td>~</td>
<td>✓</td>
</tr>
<tr>
<td>Di Noia [94, 95]</td>
<td>DL-model</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Raimondi [111]</td>
<td>SLAng</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Skene [123]</td>
<td>SLAng</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Chen [22]</td>
<td>ad-hoc</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>WSAG4 [96]</td>
<td>WS–Agreement</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mahbub [63, 124]</td>
<td>~ WS–Agreement</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Palacios et al [105]</td>
<td>WS–Agreement</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SALMon [101]</td>
<td>ad-hoc</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Legend: ✓ support Cons.: Consistency, non-/Compl: non-/Compliance, Un/fulfil: Un/fulfillment, Cons++: Consistency considering optional conflicts.
Optional Conflicts Detection and Explanation

The inconsistencies are contradictions between agreement elements that wholly invalidate an agreement document. But there are other kinds of conflicts that just affect to specific agreement terms or variants. We consider these conflicts are harder to detect because they usually involve relationships amongst several properties. Thus, an explanation is really useful for any party in charge of the SLA management at any phase of the SLAs lifecycle.

Figure §5.6 includes three examples of this kinds of conflicts. Specifically, Figure 5.6(a) includes a qualifying condition that can not be fulfilled by the creation constraint due to the lack of REST interface. However, just such term is invalidated and not the whole template. This kind of conflict is tackled by Palacios et al. in [105] in their testing technique and they call it inapplicable terms. In turn, Figure 5.6(c) has an SLO that when it is enabled, i.e. when the REST interface is chosen, it is in contradiction with other SLO. In this case the contradiction just affect to such terms and only when the qualifying condition is enabled. And Figure 5.6(b) includes an optional term with the following contradictory qualifying condition: (Only if Region = EEUU AND Europe). Such a conflict is an usual mistake when a SLA designer translate a requirement from natural language such as (to enable the phone support for EEUU and Europe regions), to a
formal expression and the designer miss to change the natural language AND to a logic XOR. This kind of conflict in an optional term just affects to such agreement variants that include such an optional term and not the whole document.

(a) Conflicting QC of a conditional term
(b) Conflicting optional terms
(c) Conflicting SLO of a conditional term

Figure 5.6: Explanation of conflicts that do not wholly invalidate documents

In general, the detection of optional conflicts is an operation that takes a WS–Agreement document, and returns whether it has such conflicts or not.

\[ \text{existOptionalConflicts}(\text{WS-Agreement document}) : \text{Boolean} \]

In turn, the explanation of these kinds of conflicts is an operation that takes a WS–Agreement document, and returns a minimal subset of conflicting agreement elements.

\[ \text{whyExistOptionalConflicts}(\text{WS-Agreement document}) : \text{AgElements} \]

Non-Compliance Detection and Explanation

In the decision making to accept or reject an agreement offer, it is very appealing for the agreement responders to know both: (1) if the offer is compliant with the corresponding template, or not, and, in the latter case, (2) which terms of each agreement document are in conflict. We consider the strong template compliance definition mentioned in Section §5.5.1.
In general, the detection of non-compliance is an operation that takes two WS–Agreement documents, a template and an agreement offer, and returns whether they are compliant or not.

\[ \text{areCompliant}(\text{WS-Agreement Offer}, \text{WS-Agreement Template}) : Boolean \]

This definition must be completed with the compliance between the guarantee terms of the template and the creation constraints of the same template as mentioned in Section §5.5.1. Thus, the \text{areCompliant} operation must be overloaded to check the elements of a given template.

\[ \text{areCompliant}(\text{Template.Terms, Template.CCs}) : Boolean \]

In turn, the explanation of non-compliance between templates and agreement offers is an operation that takes two WS–Agreement documents, a template and an agreement offer, and returns a minimal subset of terms from the agreement offer that contradict a minimal subset of agreement elements from the template.

\[ \text{whyAreNotCompliant}(\text{WS-Agreement Offer, WS-Agreement Template}) : \]

\[ < \text{TemplateAgElements, OfferTerms} > [\ast] \]

Such an explanation is useful to consider renegotiation or adaptation tasks, or even to accept a non compliant offer because the template owner consider it as acceptable. Figure §5.7 shows an agreement offer and the template in which it is based. The agreement offer is not compliant with the template because the offer designer wanted to pay less for a not reduced redundancy storage service. Knowing such situation, the template owner may decide if it accepts or rejects the offer. However, if the template owner accept the offer without such an explanation, pricing problems will appears at service consumption time, and possibly a lawsuit as consequence.

Finally, the explanation of non-compliance between the guarantee terms and creation constraints of the same template returns a minimal subset of guarantee terms that are less restrictive than a minimal subset of creation constraints.

\[ \text{whyAreNotCompliant}(\text{WS-Agreement Template}) : < \text{Template.GTs, Template.CCs} > [\ast] \]

An example is provided in Figure 5.4(b).
Figure 5.7: Explanation of non-compliance between WS–Agreement documents

**Improving the SLA Unfulfillment Explanation**

We have found different monitoring tools supporting WS–Agreement documents [27, 63, 103, 124]. Specifically, in [103] it is proposed a technique to test the SLAs in order to adapt the potential problems. Such SLA adaptation may avoid problems in the further SLA deployment. The approach of [27] supports the adaptation of SLAs defined on service-based applications (SBAs) comprising several services. Such an adaptation is needed when a service included in the SBA is replaced to a new one. Finally, [63, 124] and other approach supporting an ad-hoc SLA specification [54], provide a kind of explanation for SLA violations that, in our opinion is not enough for the following reasons.

On the one hand, in [54] the explanation for an SLA violation is provided as the violation of specific service properties acting as key performance indicators. For instance, if the response time is violated, it is pointed out as violating property independently of the number of terms in which such a property appears. Thus, it is not provided which specific term was violated. On the other hand, the explanation provided in [63, 124] is complementary to the previous one because the causes of a violation are explained with the violated terms solely. However, such an approach lacks of a precise explana-
tion when the violated term has more than one service property involved. For instance, AmazonS3 storage service guarantees a certain MUP reinforced by a clause to reward their consumers against a lack of service, but only if the consumer demonstrates such a lack of service (cf. Section §2.2.2). Thus, a term in which AmazonS3 SLA stated that the MUP is more than 99.9 % would be enough as explanation for a violation. Nevertheless, if the term were \((MUP > 99.9 \% \text{ AND ResponseTime} < 100 \text{ ms})\), we would need to know if the violation was caused by the MUP or the response time because only a low MUP is rewarded.

We consider that a more general operation to explain the SLA violations at a specific monitoring period (MP) must be proposed returning as explanation both, the violated SLA terms and the specific violating monitored data.

\[
\text{whyIsNotFulfilled}(\text{WS–Agreement SLA, MonitoredData, MP}) : \]

\[
< \text{ViolatedTerms}, \text{ViolatingMonitoredData} > [\ast]
\]

**Improving the Interoperability between Analysis Proposals**

We have found several approaches with a different support to create WS–Agreement documents [26, 96, 100] that could be enhanced with the analysis operations motivated in the current section. Between them, [26, 100] use WS–Agreement with a different purpose than such defined in the specification. Specifically, Comuzzi et al. in [26] do not use the templates and agreement offers to create an SLA, but they propose the use of ad-hoc consumer requirements and provider offers in order to create consistent SLAs. Nevertheless, the WS–Agreement specification is used for the final SLA achieved. In turn, Oldham et al. in [100] propose a partner selection between WS–Agreement documents in which a provider SLA are matched with consumer SLAs, instead of using templates and agreement offers that are the documents proposed by WS–Agreement for such a purpose. WSAG4J [96] is the most WS–Agreement-compliant tool and it supports some analysis facilities already (cf. Section §3.5).

We consider that the functionality provided by these approaches to create SLAs with WS–Agreement may complement each other in order to have a improved analysis support. Thus, an improved and ideal process considering these approaches would be as follows:

1. At the preparation phase a provider could store a template by using WSAG4J in order to put it available.

2. At information phase a potential customer may get such a public template and check two aspects: if the template matches its requirements by using the technique developed in [26], and if the template matches different agreement offers by using the technique developed in [100].
3. At negotiation phase the created agreement offer would be sent to the provider through WSAG4J framework. Afterwards, the provider may check its compliance by using the technique developed in [100] and to decide if it is an acceptable offer.

4. Finally, at deployment phase the agreement offer would be deployed as the SLA through WSAG4J.

The interoperability needed to perform such an ideal process is not provided due to the aforementioned different use of the WS–Agreement specification, and the different WSACs used for each approach. In addition, although such an ideal process would have a better analysis support, it still lacks of some of the motivated facilities such as the explanation of conflicts that are really useful to solve contradictions between the terms. Therefore, we consider interesting to take WSAG4J, as the most WS–Agreement-compliant tool and to incorporate the analysis operations developed in the current dissertation to improve its analysis support.

5.6 Overview of Our Solution

This doctoral thesis is aimed at enhancing the current support for the development of SLA-driven solutions. Specifically, we focus on the specification of WS–Agreement documents and its analysis. In the following, we outline our overall solution to overcome the shortages found in current approaches regarding the problems defined in Sections §5.2, §5.3 and §5.4. It is depicted in Figure §5.8, where the contributions related to SLA specification are placed down, in the left side, those dealing with analysis problems occupy the right side, and those dealing with the SLA-driven solutions support are showed at the top of the Figure. The modules within each side are separated into layers, so that modules in one layer rely on or use the functionality provided by the modules that are just below them. Let us sum up each contribution of this thesis according to the modules shown in the figure.

C1. iAgree. In this dissertation we do not intend to improve the WS–Agreement specification providing more instructional information in response to the comment made by Waldrich et al. state in [11]. But we are convinced that defining an intermediate WSAC will help to develop SLA-based solutions. We propose iAgree, an Intermediate ws-AGREEement configuration to specify WS–Agreement templates, offers and agreements. iAgree provides: (1) support to all the elements mentioned in WS–Agreement specification, including QC, TC, terms scope, rewards and penalties (highly WS–Agreement-compliant); (2) highly expressive and extensible SLOs, QCs, and CCs thanks to the use of an underlying CSP solver able to deal with any kind of constraints; (3) domain-independence, by both, the extensibility of its language, and the use of externally defined catalogues of metrics whose application domains and operators are
just limited by the underlying solver; (4) validity periods for the whole document or specific terms; and (5) a human-readable syntax that avoids the use of XML tags.

C2. iAgree Validity Criteria. We have identified a set of validity criteria for iAgree documents, which allows us to check up to 4 kind of semantic errors: inconsistencies, dead terms, conditionally inconsistent terms, and non-compliant terms. As far as we know, we are pioneers at defining semantic errors for WSACs.

C3. iAgree Formal Semantics. We have defined a formal semantics for iAgree documents that is based on Constraints Satisfaction Problems (CSPs). This provides the iAgree constructions with precise meaning, and eases the automated extraction of information from them.

C4. iAgree Document Analyser. We have defined an iAgree Document Analyser (ADA) that comprises a catalogue of analysis operations. A small set of these operations, the core operations have been developed on the basis of the iAgree formal semantics to check basic iAgree properties such as the consistency, compliance and why a document is inconsistent. Such core analysis operations can be used and combined to create compound operations. For instance, several compound operations have been developed by combining the core operations in order to automate the checking and explanation of inconsistencies that do not wholly invalidate iAgree documents. Advanced SLA-driven solutions, as such included in Chapter §8, can be developed as a combination of the core operations.

C5. SLA Lifecycle Support. We provide support for the validation of iAgree documents in the agreement creation and monitoring processes. Specifically, we have extended WSAG4J [96] and SALMon [101] approaches with an enhanced analysis support that allows to validate the iAgree documents (1) while the documents are being prepared, (2) during the agreement is being negotiated, and (3) when the agreement is deployed to regulate the service consumption. Moreover, we provide explanations for possible semantic problems that the documents may include.

C6. IDEAS. We provide support for both human and software clients that are involved in the development of SLA-driven solutions. In this case, we provide a development environment that comprises all tools developed in this dissertation that we coined IDEAS. The contribution is not the tool development itself, but the concept of a novel Integrated Development Environment for SLA-driven Solutions (IDEAS), since to the best of our knowledge there not exist any similar one. IDEAS provides a publicly-available user-friendly front-end which makes possible: (1) to edit documents assuring they are valid, (2) to analyse some properties appealing for final and technical users at
any phase of the SLA lifecycle. Such an environment is also available to be used by software clients through a triple distribution model: as a Java library, as an OSGi\textsuperscript{4} service, and as a web service.

![Figure 5.8: Overview of the contributions of this doctoral thesis](image)

### 5.7 Summary

In this chapter we have presented the main problems that motivate the contributions of this dissertation. We have analysed the drawbacks of the WSACs used in the literature and we have observed that none of them provide a full support to specify real-world agreements. We have also motivated the need for an enhanced analysis of WS–Agreement documents by considering operations to check and explain the semantic problems that a WS–Agreement document may include at any phase of the SLA lifecycle. Finally, we have provided an overview of the solution provided in the dissertation.

\[\text{\textsuperscript{4}www.osgi.org}\]
In this chapter, we detail iAgree, an Intermediate ws-AGREEment configuration which provides a fully-fledged language for WS–Agreement documents. After a brief introduction to iAgree in Section §6.1, we describe the iAgree sublanguages in Sections §6.2–§6.6. Section §6.2 exposes how the relevant agreement context information is specified in iAgree, including the definition of expressive validity periods to be used in the document. Section §6.3 details the sublanguages for describing the service features and properties. Section §6.4 includes the sublanguages for SLOs, qualifying conditions, and other optional information to be included in the guarantee terms. How the service and guarantee terms can be composed is related in Section §6.5. Section §6.6 details the sublanguage for specifying agreement creation constraints in templates. In Section §6.7 the validity criteria of iAgree documents is provided. Finally, a summary of the content closes the chapter in Section §6.8.
CHAPTER 6. IAGREE. AN INTERMEDIATE WS-AGREEMENT CONFIGURATION

6.1 INTRODUCTION

Despite providing a fully-fledged language for WS–Agreement documents is not a goal of the recommendation, we find that the specific sublanguages included in previous SLA specifications such as WSLA [61] have been appealing for researchers when a fully-fledged language for the SLAs was required [6, 93, 122]. In this dissertation a fully-fledged language for WS–Agreement documents is provided by means of iAgree, an Intermediate ws-AGREEment configuration, that complements the WS–Agreement schema with a sublanguage for each extension point of the specification (cf. Section §2.3). Figure §6.1 depicts that iAgree provides sublanguages for: (1) agreement context information, (2) service terms, (3) guarantee terms, and (4) creation constraints. Note that the unique WS–Agreement element in the Figure that is linked with two elements in iAgree is the metric sublanguage. This is due to we define the metrics for service properties and attributes of SDTs in iAgree documents by means of external metrics catalogues whose reference is included within the agreement context.

iAgree is provided with features that are not included in other WSACs (cf. Section §5.5.1) such as: (1) it is fully WS–Agreement-compliant, providing support for conditional terms and variants; (2) it is domain-independent and expressive enough to support arithmetic-logic predicates to describe SLOs, QCs, and CCs; (3) it is time aware, supporting validity periods for the whole, or a part of the documents; and (4) it has defined validity criteria to allow the analysis of iAgree documents.

iAgree documents are described using a human-readable notation as the template of Figure §6.2 depicts. Note that, despite the fact that iAgree sublanguages semantics is detailed later in the chapter, the iAgree document of the figure is more human-readable than the XML-based languages used in the WSAC proposed in the literature. Nevertheless, in order to serialise the document, iAgree provides a mapping to XML that is used for our automated analysis technique (cf. Appendix §B).

An intuitive definition of the iAgree sublanguages semantics is provided in the following sections, in the order of appearance in iAgree documents. Abstract and concrete iAgree syntaxes are included in Extended Backus-Naur Form (EBNF) in Appendices §A and §B, respectively.
Figure 6.1: iAgree sublanguages completing the WS–Agreement schema
Template: AmazonS3 version 1.3 //excerpt of 1.0 with validity periods
Provider Amazon as Responder;
Consumer Me;
Global Period: during 01/01/2012..12/31/2013 except from 23:00 to 00:00 on SUN;
Defined Period:
  FreeTrialQuota: during 01/01/2012..12/31/2012;
Metrics [iAgree.generalMetrics, Amazon.metrics]

Agreement Terms
Service AWS-S3 available at: .aws.amazon.com/s3;
Operations: getData,
Global Description:
  Region;
  DataDurability = 99.999999999;
  RRS = False;
  StorageSize;
  FirstProject;
  TotalPrice, StoragePrice, SupportPlanPrice;
Description for getData:
  Interface
  Monitorable Properties
    global:
      MUP;
    for getData
      ResponseTime;

Guarantee Terms
G1: Provider guarantees MUP >= 99.9;
on GlobalPeriod;
...

G3: Provider guarantees ResponseTime < 1000;
on GlobalPeriod
  onlyIf (Interface = SOAP);
  upon getData

G4: Provider guarantees ResponseTime < 700;
on GlobalPeriod;
  onlyIf (Interface = REST);
  upon getData
...

Creation Constraints:
C1: Region belongs {US, EU_Ireland};
C2: TotalPrice = 0
  on FreeTrialQuota;
  onlyIf (FirstProject = true and StorageSize <= 5);
...

Figure 6.2: Excerpt of AmazonS3 service scenario template in iAgree
6.2 AGREEMENT CONTEXT INFORMATION

iAgree supports the description of the mandatory context information of WS–Agreement specification such as some information about the parties involved in the agreement creation, and the kind of document. In addition, iAgree also allows defining some optional information in the agreement context such as the validity periods and metrics to be used in the document. Figure §6.3 shows an schema of the context information supported by iAgree (cf. the complete EBNF in Appendix §B) that is explained in detail in the following sections.

![Figure 6.3: Schema of the context information supported by iAgree](image)

### 6.2.1 WS–Agreement Protocol Information

The context information included in WS–Agreement is related to the offer-template agreement creation protocol proposed in the specification. In this sense, it is needed to identify the document type, its name, the agreement expiration time, and information about the parties involves in the agreement creation parties such as the initiator or responder role they plays. Note that despite the kind of document and its name is out of the context in the specification, we explain their syntaxes in this section because they are needed for the WS–Agreement protocol.

The iAgree syntax describes the document name and identifier after specifying the type of WS–Agreement document by using `Template`, `Agreement`, or `AgreementOffer` keywords (cf. Figure §6.4). In case the document is an agreement offer, the former template in which the offer is based may be specified as depicted in the first line of Figure §6.5. In addition, the agreement context in WS–Agreement documents must describe the parties involved in the agreement. Specifically, it must be specified their names and roles of initiator or responder. iAgree describes the parties names after `Provider` and `Consumer` keywords, and the `Responder` keyword denotes which of the parties may accept or reject the agreement offers. Note that the initiator party is the party without a specified role. For instance, in the template of Figure §6.4, the provider is the responder and the consumer is the initiator of the agreement and thus, it is in charge of preparing agreement offers. iAgree also supports to describe the date in which the agreement expires by means of the `Expiration Time` clause in the context as depicted in Figures §6.4 and §6.5.
CHAPTER 6. IAGREE. AN INTERMEDIATE WS-AGREEMENT CONFIGURATION

Template AmazonS3 version 1.3
Provider Amazon as Responder;
Consumer Me;
Expiration Time: 12/31/2013;
...

Figure 6.4: Protocol information of the AmazonS3 template in iAgree

AgreementOffer MyOffer for Template AmazonS3 version 1.3
Provider Amazon as Responder;
Consumer Me;
Expiration Time: 12/31/2013;

Figure 6.5: Protocol information of an offer to be sent to Amazon in iAgree

6.2.2 Validity Periods

IAgree supports VPs more expressive than the aforementioned expiration time. The expiration time just provides an VP that starts at the agreement deployment time and ends at an specific end time. In order to support more kinds of VPs, we proposed in [75] an abstract syntax for specifying VPs in WS–Agreement documents that is included in Figure §6.6 by means of an UML class diagram. We consider expressive VPs, which can be either joint/non-disjoint, periodic/non-periodic time intervals.

The Global Period clause in the context supports the specification of an expressive lifetime for the whole agreement, referred to as global period (GP) in this dissertation. The VPs in iAgree support these kinds of expressive time intervals. For instance, a disjoint and periodical VP is included in the GP of Figure §6.7 that specifies: the comprised hours, week days and dates by means of the from, on and during keywords, respectively. In addition it excludes a time interval by means of the except keyword. A simpler example can be found in the iAgree syntax of Figure §6.6 that has assigned the following GP: from 01/01/2012 to 12/31/2013, except from 23:00 to 00:00 on every SUN.

The iAgree context section supports to describe VPs that could be referred from specific terms and creation constraints (cf. Section §6.4) by means of the Defined Pe-
6.2. AGREEMENT CONTEXT INFORMATION

period clause. We call this VPs as local periods (LPs) in this dissertation. In order to improve the iAgree document readability and to make the definition of VPs easier, we provide two syntactic facilities:

1. VPs can be defined by exclusion of other periods, as depicted in the GP of Figure §6.7. Another example would be the definition of a nonFreeTrialQuota period that could be defined as GlobalPeriod except FreeTrialQuota.

2. VPs can be defined from a relative current time point to the future or to the past. For instance, a period that denotes the last 24 hours would be specified in iAgree by means of (from current to a day before) (cf. examples of Section §E.3).

Template AmazonS3 version 1.3
Provider Amazon as Responder;
Consumer Me;
Expiration Time: 12/31/2013;
Global Period: from 00:01..24:00 on SUN-SAT during 01/01/2012..12/31/2013 except from 23:00 to 24:00 on SUN;
Defined Period:
FreeTrialQuota: during 01/01/2012..12/31/2012;
...

Figure 6.7: VPs included in the context information of AmazonS3 in iAgree

6.2.3 Metrics Catalogues

Finally, we have also extended the context information of WS–Agreement documents in iAgree with the declaration of the metrics used for to describe the service and its properties, within a section started by the Metrics keyword. The metrics are usually described in external metrics catalogues that can be domain-dependent or not.

In order to provide a complete definition for a metric the following elements are required: (1) an identifier, (2) a description of its meaning in natural language, (3) the allowed values domain, and (4) its measure unit. Figures §6.9 and §6.10 shows two metrics catalogues. The former includes domain-independent iAgree general metrics and the latter details the domain-dependent metrics of AmazonS3. IAgree may import such metrics catalogues by including its location through a Uniform Resource Identifier (URI) (see iAgree.generalMetrics, and Amazon.metrics in Figure §6.8) after the Metrics keyword. Nevertheless, iAgree allows to define specific metrics for a document as the ErrorRate in Figure §6.8, by specifying its identifier and allowed values.

The metrics in the iAgree metrics catalogues can define allowed values in five different domains:

- Integers: Their elements are integer numbers of an specific range of values. For instance, an allowed range for the ReadRequests would be [0..1000]. If a range is not specified it is considered [MIN_INTEGER..MAX_INTEGER], by default.
CHAPTER 6. IAGREE. AN INTERMEDIATE WS-AGREEMENT CONFIGURATION

- Floats: Their elements are float numbers of an specific range of values. For instance, an allowed range for the MUP defined as the probability of a lack of service, would be \([0..100]\). If a range is not specified it is considered \([\text{MIN}_\text{FLOAT}..\text{MAX}_\text{FLOAT}]\), by default.

- Booleans: Their elements only may take two values: true or false. For instance, the customer request on the phone support.

- Enumerations: Their elements are values denoted by identifiers. For instance, the interface used to consume the service can be either SOAP or REST.

- Sets: When more than one literal are necessary to establish a metric value, it is needed a set domain. For instance, the AmazonS3 service is offered in more than a region or country (cf. Region metric in Figure §6.10).

```
Template AmazonS3 version 1.3
  Provider Amazon as Responder;
  Consumer Me;
  Metrics [iAgree.generalMetrics, Amazon.metrics]
  ErrorRate: float [0..100];
...
```

Figure 6.8: Metrics included in the context information of AmazonS3 in iAgree

```
catalogue iAgree.generalMetrics {
  metrics {
    Availability {
      description: "Probability to the requested service to be delivered";
      domain: float [0..100] %;
    }
    ResponseTime {
      description: "Time, in milliseconds, in which a request is responded";
      domain: float [0..\text{MAX}_\text{FLOAT}] ms;
    }
  }
}
```

Figure 6.9: General iAgree metrics catalogue
catalogue Amazon.metrics {
metrics {
    Region {
        description: "Regions and countries in which the AmazonS3 service can be consumed";
        domain: set {US, west-US, EU,Ireland, Pacific_Asia, Southamerica, GovCloud_US}; }
    MLP {
        description: "Monthly Uptime Percentage, that is the results of subtracting from 100% the average of the error rates. The error rates are internal server errors divided by the requests during each five minute period";
        domain: float [0..100] %; }
    Interface {
        description: "Service interface used by the consumer to request the service";
        domain: enum {SOAP, REST}; }
    StorageSize {
        description: "Size of the data storage required by the consumer in TB";
        domain: float [0..5000] TB; }
    DataDurability {
        description: "Probability of the data to be successfully stored and retrieved";
        domain: float [0..100] %; }
    RRS {
        description: "Reduced Redundancy Storage offered by AmazonS3 for a cheaper service";
        domain: boolean; }
    FirstProject {
        description: "Denotes whether the current project is the first customer project or not";
        domain: boolean; }
    TotalPrice {
        description: "Amount of money to pay for the service in a monthly billing cycle";
        domain: float [0..MAX_FLOAT] euro; }
    StoragePrice {
        description: "Partial amount of money to pay for the required data storage";
        domain: float [0..MAX_FLOAT] euro; }
    SupportPlanPrice {
        description: "Partial amount of money to pay for the required customer support plan";
        domain: float [0..MAX_FLOAT] euro; }
    TransferredGb {
        description: "Number of GB transferred at service consumption time";
        domain: float [0..MAX_FLOAT] GB; }
    ReadRequests {
        description: "Counter of read requests performed by the AmazonS3 customer";
        domain: integer [0..MAX_INT]; }
    WriteRequests {
        description: "Counter of write requests performed by the AmazonS3 customer";
        domain: integer [0..MAX_INT]; }
    OnlineReportingSupport {
        description: "Denotes whether the customer requires the online reporting support feature to solve possible problems";
        domain: boolean; }
    PhoneSupport {
        description: "Denotes whether the customer requires the phone support feature to solve possible problems";
        domain: boolean; }
    TurnAroundTime {
        description: "Counter in minutes taken by Amazon customer service to solve possible problems";
        domain: integer [0..MAX_INT]; }
}
6.3 SERVICE TERMS

The iAgree sublanguage for service terms supports the specification of service description terms (SDTs), service properties (SPs), and service references (SRs) (cf. Section §2.2), following the schema included in Figure §6.11 (cf. the complete EBNF in Appendix §B).

```
Template/AgreementOffer/Agreement AmazonS3 version 1.3 [for Template... -in offers-]

   ... Metrics [catalogue1, ..., catalogueN]     //metrics catalogues in context

   AgreementTerms
      Service name available at serviceReference //service name and reference
      Operations: operation1, ..., operationM;     //service operation list
      Global Description:
         attribute1 [value1];                      //section for service offer items
            ...
         attributeP [valueP];
      Description for operation1:
         attribute1 [value1];                      //section for offer items of specific operations
            ...
         attributeQ [valueQ];
            ...
      Description for operationM:
            ...

   Monitorable Properties
      global:                                       //section for service properties
         serviceProperty1;
            ...
         servicePropertyR;
      for operation1:
         serviceProperty1;
            ...
         servicePropertyS;
      for operationM:
         serviceProperty1;
            ...
         servicePropertyT;
            ...
```

Figure 6.11: Schema of the service terms supported by iAgree

The SDTs are described in iAgree by means of named sets of offer items that describe the service features and it may include a value assignment. Figure §6.12 includes the iAgree abstract syntax to describe SDTs and a basic example of the iAgree concrete syntax. Such a syntax describes the offer items by using attribute–value pairs and providing the attribute metrics in the catalogue Amazon.metrics.

The offer items can refer, either to the whole service or to an specific service operation. If they refer to the whole service they are grouped by using the Global Description clause as depicted in the example of Figure §6.12. If offer items refer to a specific service operation they are grouped by using the Description for clause, followed by the operation(s) name(s). In these cases, a list of operation names must be included in the document by using the Operations keyword. For instance, in the template of Fig-
Figure 6.12: Abstract syntax and example of SDTs in iAgree

... the offer items that refers to the whole service are grouped within the Global Description; and the offer items that refers either with getdata or storedata service operations, are grouped by the operation name.

Note that when a value appears in the offer item of a template it denotes the by default value for such an offer item but it is not imposed the value and it may be changed in further agreement offers. For instance, the 99.999999999 % and false values assigned in Figure §6.13 for DataDurability and RRS offer items respectively, denotes that the data durability offered by Amazon by default, if the customers do not require the cheaper reduced redundancy storage (cf. Section §2.2.1).

It is remarkable that if an XML element can be flattened in terms of attribute-value pairs and its domain has been defined in an XML schema, then it can be easily mapped into this sublanguage. Koller et al. in [57] support this idea by using attribute-value pairs to describe services. We consider that the advantages of using a general sub-language for SDTs is more relevant than the potential expressiveness of more specific (and possibly more complex) sublanguages like JSDL or WSDL for example. Applying this general approach, we can automatically process any SLA document independently of the specific nature of the services to be agreed upon, considering that the transformation from any XML-based sublanguage to our proposed format is straightforward. Appendix §E includes some mappings from other WSACs to iAgree, including the mapping of an SDT described using JSDL to iAgree.

In order to reference a service, we just propose a simple SR sublanguage by including in the service description, the URI in which such a service is available, as showed in Figures §6.12 and §6.13.

---

1They call it as SLA parameters
CHAPTER 6. IAGREE. AN INTERMEDIATE WS-AGREEMENT CONFIGURATION

Template AmazonS3 version 1.3 //with service operations
...
Metrics [iAgree.generalMetrics, Amazon.metrics]

AgreementTerms
Service AWS-S3 available at: aws.amazon.com/s3
Operations: getData, storeData;
Global Description:
Region;
DataDurability = 99.999999999;
RRS = false;
StorageSize;
FirstProject;
TotalPrice;
Description for getData:
Interface;
Description for storeData:
Interface;

Monitorable Properties
global:
AverageResponseTime;
for getData:
ResponseTime;
for storeData:
ResponseTime;

Figure 6.13: Service Terms of AmazonS3 template in iAgree

Since the SPs are sets of service properties whose value may change while the service is being consuming, we denote it by the Monitorable Properties clause. Similarly to the offer items, if the SPs refer either to the whole service or to specific operations, they can be grouped by using the global or for keywords, respectively (cf. Monitorable Properties section in Figure §6.13).

6.4 GUARANTEE TERMS

Once specified the context and service terms, it is possible to define the guarantee terms (GTs) that specify the parties obligations on the service. Figure §6.14 shows an schema of the iAgree syntax for GTs (cf. the complete EBNF in Appendix §B) that is explained in the section.

The mandatory elements of a GT are: its identifier, the SLO and the obligated party that guarantees such an SLO as denoted in Figure §6.14. In iAgree the obligated party is denoted by the Consumer guarantees or Provider guarantees clauses, followed by the guaranteed SLO. In order to define the SLOs, iAgree allows assertions for service properties and literals using algebraic operators and mathematical operations. Nevertheless, the kind of supported operations depends on the specific CSP solver used in the reference implementation (cf. Section §8.3). The template of Figure §6.15 includes several SLOs with its obligated party. For instance, G2 just provide the mandatory information of a guarantee term: (1) an SLO limiting the transferred data and the the
write requests; and (2) the obligated party that is the consumer.

The same assertion sublanguage may be used to specify an optional qualifying condition (QC) that enables the SLO. As G3, G4, and G5 of the template of Figure §6.15 denotes, the QC is described by using the onlyIf keyword. For instance, if the interface used is SOAP, the response time provided by Amazon is less than if it were REST because G3 term is enabled and G4 disabled.

As mentioned in Section §2.2, there may be guarantee terms whose scope is either the whole service or a set of service operations and iAgree supports it by using the upon keyword followed by a set of service operation names. For instance, when getData service operation were consumed, G3 and G4 terms whose scope is such an operation would be taken into account. Note that if a guarantee term, such as G1 and G2, does not specify a scope, its scope is considered to be the whole service.

The business value list elements mentioned in the WS–Agreement specification are supported by iAgree as follows:

- An order relation of the importance between the terms of the iAgree document can be specified by using the relative importance clause. For instance, G1 is more important than the other guarantee terms of the template of Figure §6.15.

- A penalty or reward can be defined by using the subject to a penalty/reward of clauses followed by the penalty or reward amount and the assessment interval by using the per keyword. For instance, the G1 term of Figure §6.15 penalises Amazon for a lack of service, and the G2 term of the same Figure rewards the consumer for the right use of the service.

- The utility of a GTs can be specified with a numeric value by using the utility keyword. Optionally, such an utility may depends on the selection of specific alternative terms included in different variants. For instance, the G3 term of Figure §6.16 that assures the online reporting support of Amazon varies its utility depending on the selection of the G5 term that assures the phone support because the phone support may be more useful for customers.
• Any other kind of business value or utility function can be expressed in iAgree by using the **business value** clause. In [75] we used such a custom business value element to specify utility functions for GTs of WS–Agreement documents.

The local period of a guarantee term can be specified in Agree by using the **on** keyword. Such a period can be defined as mentioned in Section §6.2 (see the period of G4 term of Figure §6.15) or by a reference to a predefined period in the Defined Period section of the agreement context (see the period of G2 term of the same Figure), or even a merge between a predefined period and another period (see the period of G5 term of the same Figure). Note that if a guarantee term does not specify a local period, as G1 and G3 of the same Figure, it is considered it is valid during the global period that defines the agreement lifetime.

### 6.5 Agreement Variants

iAgree supports the structure of term compositors proposed in a WS–Agreement document (cf. Section §2.4). Although iAgree supports to compose both, service and guarantee terms, the most common terms to be composed are the guarantee terms.

As depicted in the guarantee term section of Figure §6.16 the terms compositors are specified in iAgree by using the **All**, **One Or More between**, and **Exactly One between** clauses, to describe mandatory, optional, and alternative terms compositors, respectively. Unfortuantely, the WS–Agreement specification does not detail the possible impact of such terms compositors in the agreement creation and monitoring processes, and moreover, a precise semantics for the terms compositors is not provided. The unique reference in the WS–Agreement specification is: *Term compositors are structural elements of an agreement offer and the agreement. Choices expressed using compositors MUST be exercised by the service provider to satisfy the described requirements through some concrete delivery of service.* At following we summarise both, the semantics we provide in iAgree to such terms compositors, and our point of view about how they affect to the agreement creation and monitoring.

The consequence of using terms compositors in iAgree templates is that the offered terms set is not unique because if optional or alternative terms compositors are included, more than two different terms of sets can be included in compliant offers. For instance, four variants can be obtained\(^2\) from the template of Figure §6.16 as mentioned in Section §2.4.

The inclusion of more than one variant within templates provide the advantage to the responder parties of having a higher probability to meet the initiator expectations on the service and its guarantees, but as disadvantage such variants make harder the template analysis. Regarding the agreement offers, the inclusion of several variants

\(^2\)Section §7.5 details how these variants are obtained.
within them is not a problem, hence, it provides more information about the initiator expectations. As an advantage, having such a kind of information may help to achieve an agreement between the parties according to the initiator preferences. However, if the agreement is created from an agreement offer that includes variants, all of them should be considered in the agreement monitoring phase. This exposed notions are merely a first step in the achieving of a precise semantics for the variants in the agreement creation and monitoring processes.
CHAPTER 6. IAGREE. AN INTERMEDIATE WS-AGREEMENT CONFIGURATION

```
Template AmazonS3 version 1.4 //1.0 with scoped terms and Business value list elements
Provider Amazon as Responder;
Consumer “Consumer with storage necessity”;
//Global Period, Defined Period, and Metrics as in previous examples

AgreementTerms
Service AWS-S3 available at: .aws.amazon.com/s3
Operations: getData, storeData;
Global Description:
  Region;
  DataDurability = 99.999999999;
  RRS = false;
  StorageSize;
  FirstProject;
  TotalPrice;
Description for getData:
  Interface;
Description for storeData:
  Interface;

Monitorable Properties
global:
  AverageResponseTime;
  AverageMUP;
  ReadRequests;
  WriteRequests;
for getData:
  ResponseTime;
for storeData:
  ResponseTime;
  MUP: Percent;

Guarantee Terms
G1: Provider guarantees MUP >= 99.9;
  relative importance 0.4;
  subject to a penalty of 1 ServiceCredit per 1 failed request;
G2: Consumer guarantees TransferredGb < StorageSize * 100
    AND ReadRequests > WriteRequests;
    on FreeTrialQuota;
    relative importance 0.1;
    subject to a reward of 1 ServiceCredit per 100 right requests;
G3: Provider guarantees ResponseTime < 1000;
  only If (Interface = SOAP);
  upon getData;
  relative importance 0.2;
G4: Provider guarantees ResponseTime < 700;
  on from 00:01..12:00 during 01/01/2013..12/31/2013
    except from 23:00 to 24:00 on SUN;
  only If (Interface = REST);
  upon getData;
  relative importance 0.2;
G5: Provider guarantees MUP > 99.999;
  on GlobalPeriod except during 01/01/2013..12/31/2013;
  only If (RRS = true);
  upon storeData;
  relative importance 0.1;
```

Figure 6.15: Guarantee Terms of AmazonS3 template in iAgree
Figure 6.16: Agreement variants inspired in the AmazonS3 scenario, in *iAgree*
6.6 Agreement Creation Constraints

The agreement creation constraints (CCs) are assertions imposed by the template creator that must be observed by the other party while defining a compliant agreement offer. The probability to reach an agreement between the parties increases if the agreement offer is compliant with the template. WS–Agreement specification describe two types of CCs, namely items that delimits the value of a specific offer item, and constraints that are used when the assertion involve more than one offer item (cf. Section §2.2). Although items can be considered as a particular case of constraints, iAgree provides support for both CCs. Figure §6.17 shows an schema of the iAgree syntax for CCs (cf. the complete EBNF in Appendix §B) that is explained in the section.

```
Template AmazonS3 version ... 
Creation Constraints:
  CId: CCexpression; //CC id and expression
  [on localPeriod;] //local validity period
  [onlyIf precondition;] //precondition
  CId: SP/SDTattribute belongs domain; //syntactic facility to constraint the domains
```

Figure 6.17: Schema of the creation constraints supported by iAgree

As far as we know, in the literature there is a lack of best practices guide or methodology for describing neither good WS–Agreement templates in general, nor the properties that must be fulfilled between GTs and CCs of a same template. Figure §6.18 denotes our perspective of the relation between GTs and CCs. As depicted in the Figure, the CCs establish the less restrictive conditions that must be satisfied in the phase of the SLA lifecycle in order to reach an agreement. The GTs may impose more restrictive conditions depending on the difficulty or the cost of assuring specific values for SPs or SDT attributes. For instance, at the left side of Figure §6.18, the less restrictive values for DataDurability, ResponseTime, and Price\(^3\) are expressed in CCs. Since the values for such properties and SDT attributes depends on the values of RRS, Interface, and StorageSize respectively, more restrictive GTs and CCs are included in the template. CCs for the non monitorable DataDurability and Price attributes; and a GT for the monitorable ResponseTime property.

The creation constraints are described in iAgree with the same assertion language for SLOs and QCs mentioned in the previous Section §6.4 (see C2–C8 constraints of Figure §6.19). Furthermore, as depicted in the C1 constraint of Figure §6.19, a syntactic facility is included to constraint the domain of a specific SDT attribute or service property, by using the belongs keyword. Such a syntactic facility provides support for items constraints in iAgree.

Optionally, the CCs can be guarded by a condition, that is specified as QCs by using

\(^3\)Note that we have used in the example a simpler version of the AmazonS3 price policy used in the dissertation. We have not considered in the example the price dependence on RRS and FirstProject SDT attributes.
6.7 Validity Criteria

As detailed in Section §5.2.6, the WSACs used in the literature barely include some validity criteria for the documents. In pursuing to determine the validity of an iAgree document we have proposed a novel classification that includes all the conflicts found in iAgree documents with and without validity periods.

The conflicts included in the classification can be found in iAgree documents, independently of the specific WSAC used, and thus, we consider the proposed classification as WSAC-independent. The conflicts can be classified attending to two different criteria: (1) the conflict type (inconsistencies, dead terms, and conditionally inconsistent terms); and (2) the conflict scope (partial and total), that depends on its influence to the whole documents or a set of the included variants. In turn, the extension for the template compliance of WS–Agreement considers checking the compliance between terms of agreement offers and templates, as motivated in Section §5.5.2.
CHAPTER 6. IAGREE. AN INTERMEDIATE WS-AGREEMENT CONFIGURATION

Template AmazonS3 version 1.0  //excerpt of 1.0
Provider Amazon as Responder;
Consumer “Consumer with storage necessity”;
Global Period: from 00:01..24:00 on SUN-SAT during 01/01/2012..12/31/2013
except from 23:00 to 24:00 on SUN;
Defined Period:
FreeTrialQuota: during 01/01/2012..12/31/2012;
Metrics [iAgree.generalMetrics, Amazon.metrics]

AgreementTerms
Service AWS-S3 available at: aws.amazon.com/s3
Operations: getData, storeData;
Global description:
Region;
Interface:
DataDurability = 99.999999999;
RRS = False;
StorageSize;
FirstProject;
TotalPrice, StoragePrice, SupportPlanPrice;
...

Creation Constraints:
C1: Region belongs {US, EU, Ireland};
C2: TotalPrice = 0;  //Free service
on FreeTrialQuota;
onlyf (FirstProject = true and StorageSize <= 5 and WriteRequests <= 2000
and ReadRequests <= 200000);
C3: StoragePrice = 0.05 * StorageSize;
onlyf RRS = true;
C4: StoragePrice = 0.12 * StorageSize;
onlyf RRS = false;
C5: StorageSize <= 5000 TB;
C6: TotalPrice = StoragePrice + SupportPlanPrice;
onlyf (FirstProject = false or StorageSize > 5);
C7: DataDurability >= 99.99;
onlyf (RRS = true);
C8: DataDurability >= 99.999999999;
onlyf (RRS = false);

Figure 6.19: Creation Constraints of AmazonS3 template in iAgree

6.7.1 Conflicts in iAgree Documents

Most of the literature related to conflicts within SLAs focus exclusively on inconsistencies applied to the whole document. However, the use of qualifying conditions in guarantee terms leads to other types of conflicts, namely dead terms and conditionally inconsistent terms. Moreover, according to the experience of users validating our proposal (cf. Chapter §8), these conflicts are harder to detect because they usually involve relationships amongst several properties. This is more significant if there is a human in the process of creating or managing the WS–Agreement document. Next, we detail these three types of conflicts.

Inconsistencies. A contradiction between the terms, parts of the terms, or the creation constraints, and all of these amongst themselves (e.g., StorageSize < 5 and StorageSize = 5 in the same expression) constitutes an inconsistency of the iAgree
6.7. VALIDITY CRITERIA

document. This means that it is impossible to find a satisfactory assignment to the variables that appear in those terms or creation constraints. The consequence is that the whole document (or one or more variants) is invalidated because it will never be fulfilled regardless of the way the service is provided.

For instance, document of Fig. §6.20 includes an inconsistency between the SLO of G6 and the creation constraint C9 because they state contradictory expressions. This contradiction may occur in the real world if the provider tries to obtain a minimum benefit by imposing a minimum StorageSize in the G6 term, due to the storage service price depends on such size (see the cost creation constraints C3, C4, and C6 of Figure §2.2). However, the template owner skips the C9 creation constraint by mistake.

![Figure 6.20: Document with an inconsistency between a GT and a creation constraint](image1)

**Dead term.** This conflict is caused when the condition of a conditional term never holds in a document or one or more variants. In other words, a dead term is a guarantee term whose qualifying condition has a contradiction with itself or one or more terms and/or creation constraints of the document making the term dead because its SLO can never be applied since its precondition never holds.

For instance, document of Figure §6.21, includes a dead term, (G7), because its qualifying condition can never be satisfied (by the SLO of the G2 guarantee term). This contradiction may occur in the real world if the template owner is the provider and he/she only considers their guaranteed terms (G7) while editing, skipping terms guaranteed by the other party (G2), by mistake.

![Figure 6.21: Document with a dead GT caused by another GT](image2)
**Conditionally inconsistent term.** This conflict is caused when a conditional term makes the document inconsistent when its condition holds, which contradicts usual expectations. In other words, a conditionally inconsistent term is a guarantee term with the following characteristics: (1) it is not inconsistent, and (2) when its qualifying condition holds, then its SLO does not hold because of a contradiction within itself, with the qualifying condition or with other terms or creation constraints of the iAgree document. Consequently, the conditionally inconsistent term is one that when the qualifying condition holds, the SLO and, hence, the guarantee term, does not hold.

For instance, the G12 term of Fig. §6.22 is conditionally inconsistent because when its qualifying condition enables the SLO, such SLO is contradictory with the G6 SLO. This contradiction may occur in the real world if the template owner tries to obtain a minimum benefit by imposing a minimum storage size (as in the G6 term), since the storage service cost depends on such size (see creation constraints of Fig. §2.2). However, in a further revision of the template, the provider decides to reduce the storage size for all regions less US due to technical problems. The provider skips that G6 term applies to every region and therefore, the conditionally inconsistent G12 does not allow reaching agreements for any region less US.

![Figure 6.22: Document with a conditionally inconsistent GT](image-url)

**6.7.2 Conflicts in Time-aware iAgree Documents**

The conflicts mentioned in the previous section apply for time-aware documents, but they must be slightly revised by considering some additional conditions which are imposed by the validity periods. Note that, since the constituent agreement elements of WS–Agreement documents may have assigned not overlapped local periods (i.e., local periods without sharing any timepoint), those agreement elements considered as conflicting in the previous sections (cf. Figures §6.20-§6.22), might not be actually so whenever their local periods do not overlap. For instance, document of Figure §6.23 includes the inconsistency of Figure §6.20 between the SLO of G5 and the creation constraint C7. However, these agreement elements are consistent in Figure §6.23 due to their local periods do not overlap. To the contrary, the SLO of G3 and the creation constraint C8 of the same Figure §6.23 are inconsistent because they state contradictory expressions in overlapped local periods.

The conclusion that arises is that the conflicts analysis of time-aware iAgree docu-
6.7. VALIDITY CRITERIA

ments requires an study the overlapping between local periods. Note that such analysis of overlapping between validity periods is a non-trivial and error-prone task when performed manually, and some automated techniques have been proposed in [68].

<table>
<thead>
<tr>
<th>Template AmazonS3 version 1.4 //with validity periods</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>Global Period: from 00:01..24:00 on SUN-SAT during 01/01/2012..12/31/2013 except from 00:00 to 00:01 on SUN;</td>
</tr>
<tr>
<td>Defined Period:</td>
</tr>
<tr>
<td>RUSH: from 00:01..24:00 on SUN-FRI during 01/01/2012..12/31/2013;</td>
</tr>
<tr>
<td>HOME: GlobalPeriod except RUSH;</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>G3: Provider guarantees MUP &gt;= 99.9;</td>
</tr>
<tr>
<td>on GlobalPeriod;</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>G5: Consumer guarantees StorageSize &gt; 30;</td>
</tr>
<tr>
<td>on HOME;</td>
</tr>
<tr>
<td>Creation Constraints:</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>C7: StorageSize &lt;= 30 //in GB</td>
</tr>
<tr>
<td>on RUSH;</td>
</tr>
<tr>
<td>C8: MUP &lt; 99.99;</td>
</tr>
<tr>
<td>on RUSH;</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Figure 6.23: Document with an inconsistent GT and CC whose VPs overlap

In addition, time-aware documents may include temporal conflicts originated by conflicting validity periods definitions. Several factors may lead to conflicting validity periods in iAgree documents, and it is quite difficult to supervise all of them in order to conclude whether a period is wrong or not. Temporal conflicts are mostly based on the concept of covering of a period [67], which is defined as follows: a period is totally covered by one or more subperiods iff the union of all timepoints composing the subperiods equals the former period. Otherwise, we get the following situations:

- **Undercovering** if there are timepoints during the former period that are not considered in any subperiod.

- **Overcovering** if there are timepoints during the subperiods that are not considered in the former period.

We identify up to four types of temporal conflicts in iAgree documents.

**Undercovering of the Global Period** This conflict appears whenever there are timepoints during the global period that are not considered in any local period. In other words, there are some periods in which the agreement is considered as active but not the service itself, or any guarantee about the service provisioning. As an example, the document in Fig. §6.24 has an undercovering of the global period UNDER_PERIOD because the guarantee on the service MUP is not detailed in 2013.
CHAPTER 6. IAGREE. AN INTERMEDIATE WS-AGREEMENT CONFIGURATION

Overcovering of the Global Period  This conflict appears whenever there are local periods in the document that are not included in the global period. In other words, there are some periods in which the agreement is not considered to be active but it is active the service itself, or some guarantees about the service provisioning. As an example, the document in Fig. §6.25 has an overcovering of the global period because, though the global period does not consider from 23:00 to 24:00 each SUN, the agreement provides the service MUP for such hours.

Out-ranged Exceptions  An out-ranged exception is due to a period which is defined by excluding from a period A another period B, such that B is not contained in or equals to A, and thus the resulting period is A. As an example, the period defined as GlobalPeriod except 01/01/2014 is an out-ranged exception considering the global period defined in previous Figures §6.24 and §6.25.

We consider out-range exceptions as a temporal conflict because if a user is trying to define a period by exclusion, the removed period should have a timepoint, at least, in common with the period from which it is removed.

Full-ranged Exceptions  A full-ranged exception is due to a period which is defined by excluding from a period A another period B, such that B contains or equals to A, and thus the resulting period is empty. As an example, the period defined as GlobalPeriod except during 01/01/2012..12/31/2013 is a full-ranged exception considering the global period defined in previous Figures §6.24 and §6.25.

We consider full-ranged exceptions as a temporal conflict because the result of an
except keyword must include a timepoint, at least, and not a period with a null dura-
tion.

### 6.7.3 Scope of conflicts

As stated in Section §6.4, term compositors enable the definition of variants within an iAgree document. Therefore, depending on the term or terms that are involved in a conflict, it may affect all variants of an iAgree document or just some of them. Note that we consider that the compliance conflict between iAgree documents commented in Section §5.5.1 affects to all variants of such documents because a further work on compliance degrees must be developed in future (cf. chapter §9).

**Total conflicts.** A conflict is considered as total when it affects all variants of a given document. For instance, assuming that there are no term compositors in the examples of the previous section §6.7.1, the scope of all those conflicts is total.

**Partial conflicts.** A conflict is considered as partial when it affects a subset of the variants of a given document. For instance, document of Figure §6.26 has a set of terms composed by the alternative terms compositor. It defines two variants, namely: \{G3, G9.1, C1\} and \{G3, G9.2, C1\}. The second variant includes a dead term because the QC of G9.2 never holds since it contradicts the creation constraint. However, in the first variant there is no conflict because it does not contain guarantee term G9.2. Therefore the scope of such a conflict is partial since it does not affect all document variants.

```
Template AmazonS3 version 1.8
... 
G3: Provider guarantees MUP >= 99.9 %;
... 
G9: Exactly One between
G9.1: Provider guarantees ResponseTime <= 1 sec;
  onlyIf (Region = US);
G9.2: Provider guarantees ResponseTime <= 2 sec;
  onlyIf (Region = EU);
... 
Creation Constraints:
C1: Region belongs \{US, EU, Ireland\};
...
```

Figure 6.26: Template with a partial dead term (G9.2) by the creation constraint

Another example of partial conflicts can be found in document of Figure §6.27. It has a set of optional guarantee terms (G10.1–G10.4) by the optional terms compositor. However, G10.1 and G10.4 are contradictory. Therefore, every variant that contain both GTs is inconsistent. However, as they are optional terms, there may be other variants in which they were not included together. Consequently, in this document there is an inconsistency conflict with a partial scope.
6.7.4 Template Compliance

The template compliance of an iAgree offer is established according to the strong template compliance notion motivated in Section §5.5.2. That is, a compliant offer must observe not only the creation constraints, but also the terms included in the template. This leads to two non-compliance situations between the documents.

Firstly, a contradiction between a set of terms or creation constraints of a template, and a set of terms of an agreement offer constitutes a non-compliance between them. This means that it is impossible to find a satisfactory assignment to the variables that appear in both: the agreement offer terms, and the template terms or creation constraints. As a consequence the agreement offer does not fulfill the template considerations. Secondly, in order to be compliant, the conditions imposed in the agreement offer must be equal or less restrictive than such stated in the template, but never more restrictive.

For instance, while document of Figure §6.28 depicts a compliant offer with the template of Figure §6.30, Figure §6.29 includes a non-compliant offer with the same template by the following non-compliant elements: (1) a mistake in the StoragePrice that does not match with the C4 creation constraint; (2) a contradiction between the G2 terms from both documents; and (3) the G3 of the offer is more restrictive than the G1 term of the template. These problems may occur in the real world if the consumer tries to: (1) pay less for the service; (2) impose some preferred conditions that contradicts the provider conditions; and (3) impose more restrictive conditions to the provider, respectively. In Section §5.5.1 another example is provided with these three kinds of non-compliance situations.

In addition, terms and creation constraints of a given template must be checked for fulfillment. That is, the guarantee terms of the template must be equal or more restrictive than the creation constraints included in the same template. Therefore, the whole compliance notion in iAgree can be summarised as the compliance of template guarantee terms (T.GTs) with the offer terms (O.Ts), and the compliance of such offer terms with the template creation constraints (T.CCs). That is denoted by the following formulae.
6.7. VALIDITY CRITERIA

**Figure 6.28: Agreement offer compliant with template of Figure §6.30**

```
AgreementOffer  AmazonS3CompliantOffer version 1.1 for Template AmazonS3 version 1.0
...
AgreementTerms
Service AWS-S3 available at. aws.amazon.com/s3
Global description
  Region = US;
  Interface = SOAP;
  DataDurability = 99.999999999;
  RRS = False;
  StorageSize = 500 GB;
  FirstProject = False;
  TotalPrice = ...
  StoragePrice = 60; //500 * 0.12
  SupportPlanPrice = ...
...
G2: Consumer guarantees TransferredGb < StorageSize * 100
    AND ReadRequests > WriteRequests;
...
```

**Figure 6.29: Agreement offer non-compliant with template of Figure §6.30**

```
AgreementOffer  AmazonS3NonCompliantOffer version 1.2 for Template AmazonS3 version 1.0
...
AgreementTerms
Service AWS-S3 available at. aws.amazon.com/s3
Global description
  Region = US;
  Interface = SOAP;
  DataDurability = 99.999999999;
  RRS = False; //Reduced Redundancy Storage
  StorageSize = 500 GB;
  FirstProject = False;
  TotalPrice = ...
  StoragePrice = 40; //non-compliance cause 1
  SupportPlanPrice = ...
...
G2: Consumer guarantees TransferredGb < StorageSize * 100
    AND ReadRequests < WriteRequests //non-compliance cause 2;
G3: Provider guarantees MLP >= 99.99;
...
```

---

---

6.7.5 SLA Unfulfillment

A contradiction between terms from an agreement described with iAgree, and specific monitored data obtained at the fulfillment phase of the lifecycle, constitutes an agreement unfulfillment (a.k.a. violation). Consequently, to detect an unfulfillment, a technique to monitor the values that service properties take at service consumption time is needed (in our joint work called SALMonADA [89] detailed in Section §8.2.2 we use the approach proposed in [101]). This conflict can be considered a particular case of the non-compliance conflict at the fulfillment phase, since the monitored data
Guarantee Terms

G1: Provider guarantees MUP > 99.9; subject to a penalty of 1 ServiceCredit per 1 failed request; //simplified penalty

G2: Consumer guarantees TransferredGb < StorageSize * 100
AND ReadRequests > WriteRequests;

G3: Provider guarantees ResponseTime < 1000;
onlyIf (Interface = SOAP);

G4: Provider guarantees ResponseTime < 700;
onlyIf (Interface = REST);

... G5-G9 omitted for spaces reasons ...

G10: One or More between:
G10.1: Provider guarantees OnlineReportingSupport = true;
G10.2: Provider guarantees TurnAroundTime = 15;
G10.3: Provider guarantees PhoneSupport = true;
G10.4: Provider guarantees ExtendedSupport = true;

Creation Constraints:
C1: Region belongs {US, EU_Ireland};
C2: TotalPrice = 0;
on FreeTrialQuota;
onlyIf (FirstProject = true and StorageSize <= 5);
C3: StoragePrice = 0.05 * StorageSize;
onlyIf RRS = true;
C4: StoragePrice = 0.12 * StorageSize;
onlyIf RRS = false;
C5: StorageSize <= 5000 TB;
C6: TotalPrice = StoragePrice + SupportPlanPrice;
onlyIf (FirstProject = false or StorageSize > 5);
C7: DataDurability >= 99.99;
onlyIf (RRS = true);
C8: DataDurability >= 99.99999999;
onlyIf (RRS = false);

Figure 6.30: Excerpt of the Template of AmazonS3 service scenario of Figure §2.2
are equivalent to an offer without guarantee terms, and the SLA is equivalent to a template without CCs. The unique difference is that it is needed a monitoring period that is extracted from the VPs of the terms to monitor. Note that the monitored data for service properties such as response time or MUP must be achieved when specific service properties that are affected for such properties are being consumed. For instance, the agreement of Figure 6.31 comprises terms defined upon specific service operations namely, \( G_3, G_4 \) for \( get\text{Data} \) and \( G_5, G_6 \) for \( store\text{Data} \). Thus, if a value of 600 ms is monitored for the \( Response\text{Time} \) property while the service is being consuming, it would be an acceptable value for the \( get\text{Data} \) service operation but not for the \( store\text{Data} \) service operation.

### 6.8 Summary

In this chapter we have introduced iAgree, our proposed intermediate WSAC. The iAgree design results from a number of discussions and research works performed during the last years that can be summarised as follows. In the very beginning of our research work [73], we studied the kind of VPs that an SLA should support. Later we worked on applying such VPs to WS–Agreement documents and an initial version of the iAgree sublanguage for VPs, was published in an international conference [75]. Within the same research work we studied how to include utility functions for guarantee terms in the WS–Agreement structure, that is one of the elements of the iAgree sublanguage for business value list related in current chapter.

The next step was to define the validity criteria of WS–Agreement documents by trying to assure its consistency [72] and compliance [71]. While developing such a validity criteria we realised about the necessity of providing the syntax and semantics of specific sublanguages to describe the services, guarantee terms, and creation constraints. The proposed syntax were closely related to the XML schema of WS–Agreement and the semantics were provided by means of semantic mappings to CSP elements. Although we had achieved a set of sublanguages comprising a fully-fledged language for WS–Agreement documents, we presented them as implementation issues.

Afterwards, we found the necessity of having a concrete syntax for WS–Agreement documents avoiding the use of XML schema and we developed an initial version of the human-readable iAgree notation that was called \textit{wsag4people} [80]. While improving the previous validity criteria, to be published in a journal [90], with part of the conflict taxonomy mentioned in the current chapter, we realised about the needs for providing an intermediate WSAC. Furthermore, some WS–Agreement promoters commented in [11] the poor instructional information provided by the WS–Agreement specification. We tackled such a lack of information by developing the iAgree syntax and validity

---

4The abstract and concrete syntax are thoroughly described in EBNF in Appendices §A and §B, respectively.
Agreement AmazonS3
  Provider Amazon as Responder;
  Consumer "Consumer with storage necessity";
  //Global Period, Defined Period, and Metrics as in previous examples

AgreementTerms
  Service AWS-S3 available at. aws.amazon.com/s3
  Operations: getData, storeData;
  Global Description:
    Region;
    DataDurability = 99.99999999;
    RRS = false;
    StorageSize;
    FirstProject;
    TotalPrice;
  Description for getData:
    Interface;
  Description for storeData:
    Interface;

Monitorable Properties
  global:
    AverageResponseTime;
    AverageMUP;
    ReadRequests, WriteRequests;
  for getData:
    ResponseTime;
  for storeData:
    ResponseTime;
    MUP: Percent;

Guarantee Terms
  G3: Provider guarantees ResponseTime < 1000;
      onlyIf (Interface = SOAP);
      upon getData;
  ...
  G4: Provider guarantees ResponseTime < 700;
      onlyIf (Interface = REST);
      upon getData;
  ...
  G5: Provider guarantees MUP > 99.999 AND ResponseTime < 500;
      onlyIf (RRS = true);
      upon storeData;
  ...
  G6: Consumer guarantees ReadRequests > WriteRequests;
      upon storeData;

Figure 6.31: AmazonS3 SLA with GT affecting different service operations
criteria detailed in this chapter.
Automated Analysis of iAgree Documents

Explanation separates us from astonishment, which is the only gateway to the incomprehensible.

Eugene Ionesco (1909–1994),
Romanian and French playwright and dramatist, author of Decouvertes (1969)

The goal of this chapter is to introduce our approach to automate the analysis of iAgree Documents. We start in Section §7.1 outlining this approach. In Section §7.2 we justify the formalization of iAgree semantics and describe the procedure we have followed to map it in terms of Constraint Satisfaction Problems (CSPs). Section §7.3 presents a CSP-based implementation of a minimum set of core operations. Next, in Section §7.4 we present a reference implementation of the operations related to validity criteria checking based on the core operations. The next Section §7.5 describes how to normalize iAgree documents. Finally, in Section §7.6 we sum up the context of the chapter.
7.1 Introduction

Typical analysis operations such as checking whether a document contains any errors become a tedious and time-consuming task. Our approach to automate this process is depicted in Figure §7.1. The underlying idea is to leverage off-the-shelf CSP reasoners to extract information from the CSP that represents iAgree documents. Thus, analysis operations on iAgree documents are defined on two normalization operations and four core operations: existInconsistencies, whyExistInconsistencies, areCompliant and whyAreNotCompliant. Each core operation can be expressed in terms of standard CSP-based reasoning operations. This scheme requires to define a mapping between iAgree and CSP denoted as map(D) in Figure §7.1.

To the best of our knowledge, there not exists a list of analysis operations, but in the case this list would exist, it should be an open-ended list. This fact led us not only to keep the number of core operations as reduced as possible, but also to keep the documents as simple as possible. In this sense, two aspects should be carefully considered, namely temporality and terms compositors. In this thesis we have decided not to consider them and focus on documents without local valid periods nor XOR/OR compositors, we refer to them as normalized documents. Based on our experience, we conjecture that inferring the results of applying a given operation on non-normalized documents from the results obtained amongst a number of normalized documents can be carried out with very simple algorithms. Furthermore, the computation complexity of these algorithms is lower than the equivalent CSP-based solution [69]. Remaining analysis operations may be organized in very different ways: lifecycle phases such as fulfillment, negotiation, deployment; application domains with special requirements such as logistics with frame agreements, cloud platforms, etc.

7.2 iAgree Formal Semantics

7.2.1 Primary goal

The primary objective of formalizing iAgree is to establish a sound basis for a sophisticated automated support. Therefore, according to the formalization principles defined by Hofstede and Proper [51], the style and target domain should be chosen accordingly (Primary Goal Principle). In our case, we follow a transformational style and we propose Constraint Satisfaction Problems (CSP) [135] as target domain. A brief introduction to CSPs can be found in Appendix §C.2.

Alternative semantic domains have been used to formalize related proposals such as Event Calculus by Mahbub and Spanoudakis [63] and Description Logics by Old-

\footnote{Actually areCompliant can be defined in terms of existInconsistencies, but we include it for the sake of legibility. It can be considered sugaring syntax.}
Figure 7.1: Outline of our approach to automate the analysis of iAgree documents

ham et al. [100]. In our case, we have chosen CSPs because of two major reasons. First, SLOs, SDTs, QCs and CCs can be considered as a set of constraints (or a single n-ary constraint) on a set of metrics, and optionally some utility functions can be used to model business value list and preferences. This way of approaching iAgree documents fits nicely into the way CSPs and CSOP (Constraint Satisfaction Optimization Problems) express their concepts and, hence, they provide a very natural way to model the problem. Thus, such modeling strategy follows the Semantics Priority Principle while keeping close to the iAgree conceptual model. Furthermore, this avoid unnecessary representational choices as suggested by the Conceptualization Principle. As a consequence, we can define iAgree semantics and, then, extended it to newer iAgree versions with more features without modifying the essence of iAgree Core semantics.

The second reason for choosing CSP is that they have already been successfully used as logical formalism for SLAs automated analysis. In fact, the semantics adopted by S-Cube, the European Network of Excellence in Software Services and Systems, to check consistency and compliance in SLAs is defined in terms of CSPs [70] strongly inspired in Ruiz-Cortés et al. [116] who pioneered interpreting these checkings as CSPs. Furthermore, there is a plethora of off-the-shelf CSPs reasoners that can be used to automatically solve CSPs efficiently and, hence, to automatically infer information from them.
7.2.2 Mapping iAgree documents

An iAgree document $D$ describes a service by means of a set of attribute–value pairs and defines some service guarantees for a set of service properties whose domain has been previously defined. The semantics of such iAgree documents can be modelled in a CSP by means of the semantic mapping we denote as $map(D)$. The basic idea is to map the offer items of the SDTs and service properties into CSP variables, whereas GTs and CCs are mapped into CSP constraints. By doing so, the set of CSP solutions represents the range of values that may take the offer items and service properties in the next step of the agreement creation process, i.e., when creating an offer from a template. This mapping is summarised in Table §7.1 and detailed as follows.

The offer items of SDTs are mapped into variables, domains and, if they are part of an offer, into constraints. Specifically, for each SDT attribute–value pair three elements are mapped to the CSP. Firstly, the attribute is added into the set of variables of the CSP. Secondly, its domain, which is obtained from a function $domain_D(\text{attribute})$ that takes the attribute values from the metric catalogue, is added into the set of domains of the CSP. For instance, the document of Figure §6.2 specifies that the offer item $\text{FirstProject}$ is included in the $\text{Amazon.metric}$ catalogue. Therefore, as can be seen in the catalogue included in Section §6.2.3 it is stated that the $\text{FirstProject}$ domain must be interpreted as the domain $\text{boolean}$, that is included in the CSP of Figure §7.2.

If the offer item has associated a service operation as scope (see “Description for op” in Table §7.1), a qualified attribute is created. For instance, the offer item $\text{Interface}$ of the same example has associated the scope $\text{getData}$, and hence, it is included in the CSP of Figure §7.2 as $\text{getData.Interface}$. Note that we have ignored in the CSP $\text{RRS}$, $\text{DataDurability}$, $\text{StoragePrice}$, and $\text{SupportPlanPrice}$ offer items of the SDT because they are not used in the agreement elements of Figure §6.2. Thirdly, if the document is an offer, a constraint is added into the set of constraints in order to assign the specified value to the variable. If the document is a template, SDTs represent default values and, hence, they are not mapped into CSP constraints.

Service properties are mapped into CSP variables and their respective domains. The domain is obtained from the function $domain_M(\text{propertyName})$. For instance, the document of Figure §6.2 specifies that service property $\text{MUP}$ is as included in the $\text{Amazon.metric}$ catalogue. Therefore, as can be seen in the catalogue included in Section §6.2.3 it is stated that the $\text{MUP}$ domain must be interpreted as the domain: real value between 0 and 100, i.e: $domain_M(\text{MUP}) = \{x \in \mathbb{R} : x \geq 0 \land x \leq 100\}$. These variables are used in the guarantee terms. As in the SDT offer items, if the service property has associated a service operation as scope (see “for op” in Table §7.1), a qualified property is created. For instance, the property $\text{ResponseTime}$ of the same example has associated the scope $\text{getData}$, and hence, it is included in the CSP of Figure §7.2 as $\text{getData.ResponseTime}$. 
Table 7.1: Mapping from iAgree to CSP
Guarantee terms are always mapped as CSP constraints. If there is a QC, the GT is mapped as an implication between qualifying condition and the SLO to represent the fact that the SLO can be applied only if the QC holds. For instance the \((\text{Interface} = \text{SOAP} \Rightarrow \text{ResponseTime} < 1000)\) constraint mapped in Figure §7.2 from G3 of Figure §6.2. Otherwise, only the SLO expression is added into the set of constraints of the CSP. For instance the \((\text{MUP} >= 99.9)\) constraint mapped in Figure §7.2 from G1 of Figure §6.2.

Creation Constraints are added into the constraints without modification. For instance, see the constraint \((\text{Region}=[\text{US, EU, Ireland}])\) in Figure §7.2.

\[
V = \{ \text{Region, getData.Interface, StorageSize, FirstProject, TotalPrice, MUP, getData.ResponseTime} \}
\]

\[
D = \{ \text{set \{US, ... , GovCloud_US\}, set \{SOAP, REST\}, float, boolean, float, float [0..100], float } \}
\]

\[
C = \{ (\text{MUP} >= 99.9), (\text{getData.Interface} = \text{SOAP}) \Rightarrow \text{getData.ResponseTime} < 1000, \\
(\text{getData.Interface} = \text{REST}) \Rightarrow \text{getData.ResponseTime} < 700 \\
\text{Region}=[\text{US, EU, Ireland}], (\text{FirstProject AND StorageSize} <= 5) \Rightarrow \text{TotalPrice} = 0 \}
\]

Figure 7.2: Example of a CSP mapping from Template of Fig. §6.2

The mapping function \(\text{map}(D)\) assumes that \(D\) is a normalized document, hence, we consider that all terms are composed by a mandatory terms compositor and active at the same validity period. In addition, the unique context information that is mapped to the CSP in order to validate the compliance between offers and templates is the information regarding the party that plays the role of responder, and the template identifier in which an agreement offer is based.

### 7.3 AUTOMATED ANALYSIS OF CORE OPERATIONS

#### 7.3.1 Geometrical interpretation of a CSP

In current section we describe our approach to check iAgree validity criteria by using a CSP-based technique to detect and explain the conflicts and non-compliance situations. Note that the examples included in the section contains CSPs mapped from examples provided in Section §6.7. In addition, some of the Figures include a Venn diagram as such depicted in Figure §7.3 that represents graphically both, the CSP solution space and the CSP constraints with a short example of the CSP shown in Figure §7.2.

#### 7.3.2 Checking and Explaining Inconsistencies

A contradiction between terms, parts of terms, or creation constraints, and all of these amongst themselves constitute an inconsistency in an iAgree document. This
7.3. AUTOMATED ANALYSIS OF CORE OPERATIONS

Solutions
Space of $\gamma$

$\gamma = (V, D, C)$

- $C_1$: $\text{MUP} \geq 99.9$
- $C_2$: $(\text{Interface}=\text{SOAP}) \Rightarrow (\text{ResponseTime}<1000)$
- $C_3$: $(\text{Interface}=\text{REST}) \Rightarrow (\text{ResponseTime}<700)$
- ... $C_N$: ...

$\gamma = (V, D, C)$ (MUP=80, Interface=REST, ResponseTime=800)

Figure 7.3: Graphical representation for CSPs

means that it is impossible to find a satisfactory assignment to the variables that appear in those terms or creation constraints. This can be expressed as:

$$\text{existInconsistencies}(D) \iff \text{solve}(\text{map}(D)) = \emptyset \quad \text{(7.1)}$$

In the case of the template in Figure §6.20 (cf. Page 107), applying this checking leads to the following non-satisfiable constraint\(^2\) (see Figure 7.4(a)):

$$\text{solve}(\{\text{StorageSize}, \text{MUP}\}, \{\ldots, \ldots\}, \{\text{StorageSize} > 30, \text{StorageSize} \leq 30, \text{MUP} = 99.9\} ) = \emptyset$$

Figure 7.4: Checking and explaining inconsistencies of Template of Figure §6.20

In order to explain why a document is not consistent we can do it by tracing back the constraints obtained by $\text{whyExistInconsistencies}$ as follows:

\(^2\ldots, \ldots\) denotes we are not interested in show them explicitly for the sake of legibility.
\[ whyExistInconsistencies(D) = explain(map(D)) \] (7.2)

In our example, see Figure 7.4(b), two constraints (two terms of the iAgree document) explain the inconsistency.

\[
\text{explain(\{StorageSize, MUP\}, \{\ldots\}, \{\text{StorageSize}>30, \text{StorageSize} \leq 30, \text{MUP} \geq 99.9\}) =}
= \{\text{StorageSize}>30, \text{StorageSize} \leq 30\}
\]

Notice that in order to make it possible to provide an explanation for the inconsistencies and the other conflicts exposed in this chapter, the relation between the CSP constraints and the terms or creation constraints of the iAgree document \(D\) must be stored during the mapping. Thus, the minimal set of constraints that make it impossible to find a solution for the CSP can be traced back to the agreement elements that were mapped into such constraints in order to be returned as the conflict explanation.

### 7.3.3 Checking and Explaining Non-Compliant terms

Checking the compliance between iAgree documents requires to check both the compliance of template guarantee terms (T.GTs) with the offer terms (O.Terms), and the compliance of such offer terms with the template creation constraints (T.CCs). Thus, in order to incorporate the simpler operations in the set of core operations, we decided to incorporate an operation to check the compliance not for the document as a whole, but for terms, guarantee terms and creation constraint. In particular we propose:

\[
\text{areCompliant(T.GTs, O.Terms) } \iff \text{implies(map(T.GTs), map(O.Terms))}
\]

\[
\text{areCompliant(T.CCs, O.Terms) } \iff \text{implies(map(T.CCs), map(O.Terms))}
\]

Where \(\text{implies}(A, B)\) corresponds to the so-called implication restriction by Marriott and Stuckey [64], which amounts to

\[
\text{implies}(A, B) \iff \text{solve}(A \land \neg B) = \emptyset
\]

Figure 7.5(a) shows an example where an offer is not compliant with a template (non-compliance analysis between guarantee terms and creation constraints of the offer are not showed for the sake of simplicity). Notice that the offer offers some constraints which are not considered by the template. Figure 7.5(b) shows these constraints.

In order to explain why there is not exists compliance, we can do it by tracing back the constraints obtained by the \textit{whyAreNotCompliant} as follows:
whyAreNotCompliant(T.GTs, O.Terms) = whyNotImplies(map(T.GTs), map(O.Terms))
(7.3)

whyAreNotCompliant(T.CCs, O.Terms) = whyNotImplies(map(T.CCs), map(O.Terms))
(7.4)

Notice that in this case, in order to find out the explanations we had to follow a more elaborated scheme than in previous checking. This new scheme was needed because the operation explains is only applicable on non-satisfiable CSPs and in this case, we are interested in knowing why the operation implies(A, B) does not hold (returns false) which according to equation 7.6 amounts to find out the explanations of a satisfiable constraint.

\[ \neg \text{implies}(A, B) \iff \text{solve}(A \land \neg B) \neq \emptyset \]  
(7.5)

In this sense, to the best of our knowledge, CSP solvers do not offer an operation to explain why an implication restriction is not satisfiable. Thus, as whyAreNotCompliant has been very often used, and it seems it will keep like that in the future, we decided to consider that whyNotImplies should be incorporated in the solution-space and thus, in the interface or our CSP solver.

There exist a number of approaches to implement whyNotImplies. In our case, taking into account that most SLAs terms describe constraints on a small set of variables, we
considered to follow an strategy based on the *quickxplain* algorithm [53] that can be found in Appendix §C.2.

### 7.4 Automated Checking of Validity Criteria

According to our approach to automate the automated analysis depicted in Figure §7.1, the operations to check the validity criteria of iAgree rely on the *core operations*. Thus, they do not need to be aware of the existence of CSPs solvers, which is an extra benefit from the design software point of view. The reference implementation of this set of operations is done in Java, but for the sake of simplicity we show them using a more mathematical style.

#### 7.4.1 Checking and Explaining Conditionally Inconsistent terms

This kind of conflict is caused when a conditional term makes the document inconsistent when its condition holds, which contradicts usual expectations. In other words, a conditionally inconsistent term is a guarantee term with the following characteristics: (1) it is not inconsistent, and (2) when its qualifying condition holds, then its SLO and, hence, the guarantee term, does not hold. Consequently, to detect if a document has at least a conditionally inconsistent term if the following holds:

\[
\text{existCondInconsTerms}(D) \iff \neg \text{existInconsistencies}(D) \land \\
\exists GT_i \in D.\text{Terms} \cdot \\
\text{existInconsistencies}(D.\text{remove}(GT_i, SLO_i, QC_i))
\]  

(7.6)

Where \(D.\text{Terms}\) denotes the set of terms of an iAgree document; \(GT_i\) denotes the i-th guarantee term belongs \(D.\text{Terms}\) whose QC and SLO are \(QC_i\) and \(SLO_i\), respectively; and \(D.\text{remove}\) denotes a function that replaces the i-th GT of the document \(D\) by its SLO and QC.

In turn, in order to explain why a document \(D\) has conditionally inconsistent terms we can do it by tracing back the explanations return by \(\text{whyExistInconsistencies}\) as follows:

\[
\text{whyExistCondInconsTerms}(D) = \\
\bigcup_{\forall GT_i \in D.\text{Terms}} \\
\text{whyExistInconsistencies}(D.\text{remove}(GT_i, SLO_i, QC_i))
\]  

(7.7)
For instance, the conditionally inconsistent term in Figure §6.22 (cf. Page 108) is checked and explained as Figure §7.6 depicts.

7.4.2 Checking and Explaining Dead terms

A dead term is a guarantee term whose qualifying condition has a contradiction with itself or one or more terms and/or creation constraints of the document making the term dead because its SLO can never be applied since its precondition never holds. Therefore, to check if a document has at least a dead term, we can check if the followings holds:

\[
\text{existDeadTerms}(D) \Leftrightarrow \neg \text{existInconsistencies}(D) \land \\
\exists GT_i \in D.\text{Terms} \cdot \\
\text{existInconsistencies}(D.\text{replace}(GT_i, QC_i)) 
\] (7.8)

Where \( D.\text{Terms} \) denotes the set of terms of an iAgree document; \( GT_i \) denotes the \( i \)-th guarantee term belongs \( D.\text{Terms} \) whose QC is \( QC_i \); and \( D.\text{replace} \) denotes a function that removes a given GT from the document \( D \) and consider its QC.

In order to explain why a document \( D \) has dead terms we can do it by tracing back the explanations obtained by the \textit{whyExistInconsistencies} as follows:
\[
\text{whyExistDeadTerms}(D) = \bigcup_{\forall GT_i \in D.\text{Terms}} \text{whyExistInconsistencies}(D.\text{replace}(GT_i, QC_i))
\]

(7.9)

For instance, the dead term in Figure §6.21 (cf. Page 107) is checked and explained as Figure §7.7 depicts.

Figure 7.7: Checking and explaining the dead term of Template of Figure §6.21

Appendix §D.1 includes a complete example for the partial dead term included in the document of Figure §6.26.

### 7.4.3 Checking and Explaining if a Document is Valid

The first necessary condition to consider a document \( D \) as valid is to be free of conflicts. This can be expressed as:

\[
\text{valid}(D) \iff \neg \text{existInconsistencies}(D) \land \\
\text{existCondInconsTerms}(D) \land \\
\text{existDeadTerms}(D)
\]

In the case of templates, an additional condition is that the terms be compliant with the creation constraints. This can be expressed as:
areCompliant\( (T.CCs, T.Terms) \)

In the case of offer with a referenced template \( T \), the strong compliance notion of iA-gree forces to check if there exists compliance between \( D \) and \( T \). This can be expressed as:

\[
\text{areCompliant}(T.GTerms, O.Terms) \land \text{areCompliant}(T.CCs, O.Terms)
\]

Regarding to explain why a document \( D \) is not valid we can do it by tracing back the explanations obtained as follows:

\[
\text{whyIsNotValid}(D) = \begin{cases} 
\text{whyExistInconsistencies}(D) \cup \\
\text{whyExistCondInconsTerms}(D) \cup \\
\text{whyExistDeadTerms}(D) \cup
\end{cases}
\]

And the corresponding \text{whyAreNotCompliant} operation, depending on the kind of document.

### 7.5 Normalization Operations

#### 7.5.1 Variants Normalization

We propose a function to get the variants of iAgree documents. The function iterates through the nested term compositors, and processes them according to their semantics. More specifically, let \( \text{variants}(T) \) be a function that, given a set of composed terms, \( T = \{t_1, \ldots, t_n\} \), returns the set of variants that it defines, then \( \text{variants}(T) \) can be defined as follows:

\[
\text{variants}(t) = \begin{cases} 
\{t\} & \text{if } T = \{t\} \text{ and } T \text{ is not a composite term} \\
\bigcup_{p \in \mathcal{P}(T) - \varnothing} \text{variants}(\text{All}(p)) & \text{if } T \text{ is OneOrMore} \\
\{\bigcup_{i=1}^n j_i \mid \bigwedge_{i=1}^n j_i \in \text{variants}(t_i)\} & \text{if } T \text{ is All} \\
\bigcup_{\forall t_i \in T} \text{variants}(t_i) & \text{if } T \text{ is ExactlyOne}
\end{cases}
\]

where \( \mathcal{P}(S) \) is the power set of \( S \) and (1), (2) and (3) depends on the type of terms compositor:
(1) If \( T \) is an optional terms compositor (\textit{OneOrMore}), then every composed term is optional, but there must be at least one term selected. Therefore, the variants are all combinations of the composed terms. For instance, in the example of Figure §7.8, the variants of the one or more compositor are \( \{G_3,G_4,\{G_3,G_4\}\} \). Note that, in the definition, the use of \textit{variants(All(\( p \)))} is necessary due to the nesting of term compositors, i.e., the terms nested in a one or more compositor could be term compositors themselves.

(2) If \( T \) is a mandatory terms compositor (\textit{All}), then every composed term is mandatory. Therefore, the variants should always contain one variant of each of the composed terms. For instance, in the example of Figure §7.8, the variants of the nested mandatory compositor always contain one variant of \( G_2 \), which is \( \{G_2\} \), and one variant of \textit{OneOrMore}(\( G_3,G_4 \)), which are \( \{G_3,G_4,\{G_3,G_4\}\} \). Consequently, the variants of this compositor are: \( \{\{G_2,G_3\},\{G_2,G_4\},\{G_2,G_3,G_4\}\} \).

(3) If \( T \) is an alternative terms compositor (\textit{ExactlyOne}), then every composed term is an alternative. Therefore, the variants of the alternative compositor are all of the variants of each composed terms. For instance, in the example of Figure §7.8, the variants of the alternative compositor are the union of the variants of \( G_1 \), which is \( \{G_1\} \), and the variants of \textit{All}(\textit{OneOrMore}(\( G_2,G_3 \))), calculated above. Hence, the variants of the ExactlyOne compositor are: \( \{G_1,\{G_2,G_3\},\{G_2,G_4\},\{G_2,G_3,G_4\}\} \).

### 7.5.2 Temporal Normalisation

The temporal normalisation of an iAgree document is being tackled by other authors [68], and hence, it is out of the scope of the current dissertation. Nevertheless, Figure §7.9 intuitively depicts the result of applying the temporal normalisation to a document with three GTs.

The example included within Figure §7.9 shows the worst case possible, i.e. a case in which the temporal projection of the different periods generates the higher number of possible maximal intervals. Where a maximal interval is comprised of all the intervals in which a combination of GTs holds. In the worst case, the highest number of possible maximal intervals, obtained while normalizing an iAgree document that
7.6 SUMMARY

In this chapter we have introduced our approach to automate the analysis of iAgree Documents. The formalisation of iAgree semantics using CSPs has been justified as paradigm to provide a sound basis for the implementation of the operations related to validity criteria checking. Such an implementation is based on a minimum set of core operations defined in terms of standard CSP-based reasoning operations. Such core operations are used by other analysis operations exposed in the chapter to provide a complete support for the automated checking of the iAgree validity criteria. These core operations and the normalisation concept itself were not so clear for us.

Figure 7.9: Applying temporal normalisation to a document with three GTs

includes n GTs included in a mandatory terms compositor (All) can be computed as:

\[ \sum_{i=1}^{n} C_{n,i} \]

Where \( C_{n,i} \) is the number of combinations that can be obtained from \( n \) GTs taken in groups of between 1 and \( n \) terms, and it is equivalent to:

\[ \frac{n!}{i!(n-i)!} \]

In the example of Figure §7.9, which depicts an example or worst scenario, the temporal normalization would return a set of 7 documents with the following validity periods and GTs:

\[ \{\{i_1\},\{GT_1\}\},\{\{i_2\},\{GT_1,GT_2\}\},\{\{i_3,i_8\},\{GT_1,GT_2,GT_3\}\},...,\{i_7\},\{GT_3\}\} \]

Where the \( i_i \) are expressed as pairs \( (\text{begin}_{\text{instant}},\text{end}_{\text{instant}}) \).
before this dissertation, and hence, in recent works [89, 90] we provided tedious formalisations (some of them are included in Appendix §D with the complete examples of this chapter) that can be improved with the content of the current chapter.

Thanks to the provided semantics, the applicability of the developed technique goes beyond iAgree and could be potentially applicable for any WSAC. The only requirement is that the semantics of the languages used in the WS–Agreement document can be interpreted as a CSP. This requirement is quite reasonable since, according to [116], it can be assumed that for automated analysis purposes any SLA can be regarded as a CSP regardless of the specification language.
In this chapter we detail the support provided by IDEAS, the developed Integrated Development Environment for SLA-driven Solutions. After an introduction to the IDEAS conceptual layers, we mention the main features that make it an industry-ready tool in Section §8.1. Afterwards, Section §8.2 reports the important validation effort that has been carried out in the pursuit of real-world usefulness for iAgree. iAgree, with its automated validity criteria checking, has made easier several SLA-driven solutions supporting the agreement creation and monitoring processes. Section §8.3 exposes the ADA analysis operation catalogue by by detailing their components and interfaces. Section §8.4 provides information about the iAgree studio, that provides an editor to design iAgree documents, and several facilities to validate and analyse them by using the ADA operations. Finally, a summary of the content closes the chapter in Section §8.5.
8.1 INTRODUCTION

We have developed a tool with a set of facilities to ease the development of iAgree-based SLA-driven solutions. The provided facilities go far beyond a reference implementation for the CSP-based techniques detailed in Section §7.3, and includes several extensions points, different distribution models, and some additional analysis operations that we will introduce in this chapter. Thus, we consider our tool, not only as an analysis framework, but an Integrated Development Environment for SLA-driven Solutions (IDEAS).

The conceptual layers of IDEAS depicted in Figure §8.1 shows that its main component is ADA, iAgree Document Analyser (see ADA Core layer in the Figure). ADA provides a reference implementation for a catalogue of analysis operations whose main responsibilities are both, to normalize and to validate iAgree documents. In addition, some iAgree document management operations have been considered in order to use them in the developed applications. For instance to translate from different syntaxes to iAgree and vice versa (see the iAgree translator component in Figure §8.9). IDEAS provides: (1) an extension point for each ADA core component in order to evolve the IDE analysis capabilities as shown at the bottom of the Figure §8.1, and (2) multiple distribution models to be deployed as a Java library, or as a web service. These extension mechanism and distribution models have been a necessity due to the incremental development of IDEAS by adding new features and analysis operations step-by-step.

Finally, some key features taken into account in the inception and development of IDEAS are summarized as follows.

Ready-to-use. The first desktop releases developed in [71, 72] were key to give to know the conflicts checking and the analysis operations. However, to make the analysis operations use more accessible, we designed a user-friendly web application (cf. Section §8.4).

Understandability. The first recognized difficulty encountered by many IDEAS end users is the reading and writing the standard XML syntax of WS–Agreement, which is not exactly human-readable. It is already difficult to write and read in WS–Agreement, let alone trying to find errors and explaining them. However, IDEAS provides the possibility to translate between the human-readable iAgree syntax and the XML-based serialized notation of WS–Agreement (see Metamodels and Translations elements of Figure §8.1).

Interoperability. Convinced of its usefulness, some end-users wanted to incorporate ADA analysis capabilities into their systems (at that moment we started to think about the concept of SLA-driven application). With a view to ease this adoption in as many
8.1. INTRODUCTION

scenarios as it were possible, we designed a triple distribution model (see Interfaces layer of Figure §8.1), namely: as a Java library, as an OSGi\(^1\) service, and as a web service. As a Java Library, ADA can be used from any Java client application. As an OSGi service, it allows OSGi developers in general and Eclipse developers in particular to use it from Eclipse plug-ins or other OSGi compliant tools. Finally, as a web service, ADA can be used from any WSDL consumer application. We are currently working on providing a RESTful interface.

**WSAC-independence.** ADA includes an adapter to deal with documents written in other WSACs than iAgree. Two alternatives to build this adapter were considered: mapping the other WSACs to iAgree, or mapping the other WSACs to CSP. We chose the former due to translations between the models we have studied were reasonably simple. Examples of these translations can be found in Appendix §E.

**Solver independence.** The automation provided by ADA is supported by mapping an iAgree document to a CSP and then finding solutions, or reasons why there are no solutions to such CSP. For this second part of the process there are many CSP solvers which have different capabilities regarding the type or scope of restrictions that can be solved, its licence model, etcetera. Knowing that these differences can hinder the use of ADA in some organisations, we have protected our design from the possible variations derived from using a different CSP Solver (see Selector element in Figure §8.1) than Choco constraint solver\(^2\), which is the one ADA includes by default. For this reason, we have designed the interface ISolve (cf. Section §8.3), with which interacts both the CSP Mapping component, which performs the semantic mapping function to map WS–Agreement documents into the concrete CSP used, and the checker components that validates the iAgree documents. In our implementation of ADA, interface ISolve is implemented by component Choco Adapter, which interacts with Choco constraint solver. Therefore, the only requirement to add support for a new CSP engine is to provide an implementation to interface ISolve.

**Licence model.** Our experience in developing industry-ready tools (over 14 tools in last 5 years\(^3\) has caused us to reflect upon many aspects [133]. One of the lessons learned is that distributing or licensing our tools as Open Source Software to a certain extent facilitates not only their adoption by third-parties but also its upkeep by some of these users. On occasions, some of these third-parties find it interesting to contribute with the tool evolution and maintenance by adding new features or evolving existing ones; or even creating new related tools in order to build a software ecosystem [16]. Currently, ADA is licensed as LGPLv3 [43].

---

\(^1\)[www.osgi.org](http://www.osgi.org)


\(^3\)[www.isa.us.es/tools](http://www.isa.us.es/tools)
8.2 **Supporting SLA-driven Solutions**

An important validation effort has been carried out in the pursuit of real-world usefulness for iAgree. As major conclusion we can claim that iAgree and its automated validity criteria checking has supported the processes of creating and monitoring agreement documents, stated as the goal of the WS–Agreement protocol in the specification (cf. Page 3 in [10]).

During such an evaluation of the support provided by iAgree we have identified a number of useful operations that have been generalised and included in the ADA catalogue of IDEAS, exposed in Section §8.3. These operations are either *analysis operations* that extracts useful information from the iAgree documents, or other *auxiliary operations* that make easier the development of the SLA-driven solutions.
8.2. SUPPORTING SLA-DRIVEN SOLUTIONS

For each SLA-driven solution described we detail the problem to solve, describe the operations that are used, and draw some conclusions.

8.2.1 Agreement Creation SLA-driven Solutions

The support provided by iAgree to create WS–Agreement documents has been validated in different ways. In this section we describe some of them such as: using the iAgree automated validity checking to enhance the analysis capabilities of both, the WSA4J framework, and a SOA-Governance system; and reducing the learning curve of WS–Agreement to our students.

Backing up WSAG4J Project

Problem to solve. In our first iAgree validation scenario we revised WSAG4J, a framework developed by members of the GRAAP working group of the Open Grid Forum to provide an implementation of the WS–Agreement protocol (see Section §3.5). We noticed that the SLA-driven solution to create agreements provided by WSAG4J lacked mechanisms to check and explain conflicts in WS–Agreement documents which might led to undesired situations: SLAs with inconsistencies, dead terms and conditionally inconsistent term, and failures during the agreement creation process due to inconsistent templates or offers.

Checking and explaining validity criteria. It was necessary to build an adapter to deal with the documents written in the WSAC of WSAG4J, we refer to it simply as WSAG4J for the sake of brevity. Two alternatives to build this adapter were considered: mapping WSAG4J to iAgree, or mapping WSAG4J to CSP. As mentioned in Section §8.1, we chose the former due to translations between the both WSACs are reasonably simple (see Appendix §E for the complete mapping examples).

We focus in this section in the negotiation process performed to create a further agreement with the WS–Agreement protocol that is supported by WSAG4J. At the beginning of the negotiation process, a provider who intend to publish their computing service capabilities in the WSAG4J server may create a template as the Figure §8.2 depicts. At the provider side, an automated checking of the template validity before publishing it in the WSAG4J server, is very appealing to assure it is free of conflicts and non-compliance between their terms and CCs. Thus, the template of Figure §8.2 would be highlighted as conflicting due to the ApplicationVersion value assignment of C3 that contradicts the C2 creation constraint. Figure §8.3 shows how the conflicts are explained to the provider as output for the code showed in Document 8.1.

4The template of Figure §8.2, and offer of Figure §8.4 are iAgree documents with intentional inconsistencies. Both documents have been inspired and mapped from WSAG4J conflict-free documents developed by Wäldrich et al. to test the WSAG4J framework. Specifically, compute-test.xml, and valid_compute-test-offer.xml. The computing service is described in the original file by using the job submission description language (JSDL) as depicted in the XML document of Section §E.1, but we have mapped it to iAgree in the current section.
CHAPTER 8. SLA-DRIVEN SOLUTIONS SUPPORT

Template UNICORE6-COMPUTE version 1.0 //with conflicts deliberately injected

Context:
Provider as Responder;
Metrics [generalMetrics, WSAG4J.metrics] //including the metrics detailed below
   ApplicationName: string;
   ApplicationVersion: float;
   InputFilename: string;
   OutputFilename: string;
WSAG4JSession: 12115dfe963–419678bc139598cf;

AgreementTerms
Service APPLICATION_STD1 available at . . . . //JSDL description
   Global description
      ApplicationName = "WISDOM−PACK";
      ApplicationVersion = 1.0;
      InputFilename = "/input.dat";
      OutputFilename = "/output.dat";

Creation Constraints:
C1: ApplicationName belongs {WISDOM−PACK};
C2: ApplicationVersion <= 1;
C3: ApplicationVersion = 1.2; //provider mistake

Figure 8.2: Template to test WSAG4J with some intentional inconsistencies

that isValid and whyIsNotValid operations check and explain each and every conflict
that may turn the template into invalid (cf. Section §7.4).

File template = new File("./UNICORE6-COMPUTE.wsag");
if isValid(template){
   System.out.println("The iAgree template document is valid");
}
else {
   Set<Set<AgElement>,ConflictType> explanation = whyIsNotValid(template);
   System.out.println(explanation); //toString() overloaded.
}

Document 8.1: Java code to check and explain validity criteria

Explaning Result:
Inconsistent terms:
explainedBy: C2 (ApplicationVersion <= 1),
C3 (ApplicationVersion = 1.2)

Figure 8.3: Explanations for the document of Figure 8.1

Considering that the provider removed C3 from the template of Figure §8.2 after the
previous checking, it is ready to receive agreement offers for the published template.
On the customer side, when an offer as such included in Figure §8.4 is prepared from
scratch for the published template, the customer may be interested in checking weather
it is valid before sending a formal offer to the provider. It may use the source code
showed in Document 8.2 to do it. The output for the program depicted in Figure §8.5
would return as explanation that in the guarantee term G1, the customer demands
to the provider a higher value for ApplicationVersion than fixed by himself in the
service description term.
8.2. SUPPORTING SLA-DRIVEN SOLUTIONS

**AgreementOffer** MyConflictingOffer for Template UNICORE6-COMPUTE version 1.0

Provider as Responder:

 Metrics [iAgree.generalMetrics, WSAG4J.metrics] //including the metrics detailed below

 ApplicationName: string;
 ApplicationVersion: float;
 InputFilename: string;
 OutputFilename: string;
 WSAG4JSession: 12115dfe58b–1426078d6fe4e1be;

**AgreementTerms**

**Service** APPLICATION_STD_1 available at ...

**Global description**

 ApplicationName = "WISDOM-PACK";
 ApplicationVersion = 1.0;
 InputFilename = "/input.dat";
 OutputFilename = "/output.dat";

**Monitorable Properties**

 global:
 ApplicationVersion;

**Guarantee Terms**

 G1: Provider guarantees ApplicationVersion > 1.0; //customer mistake

---

**Figure 8.4:** Offer to test WSAG4J with some intentional inconsistencies

```java
File template = new File("./UNICORE6-COMPUTE.wsag");
File conflictingOffer = new File("./MyConflictingOffer.wsag");
if(isValid(conflictingOffer, template){
    System.out.println("The iAgree offer is valid");
} else {
    Set<Set<AgElement>,ConflictType> explanation = whyIsNotValid(
        conflictingOffer, template);
    System.out.println(explanation); //toString() overloaded.
}
```

**Document 8.2:** Java code to check and explain validity criteria of an offer

```
Explaining Result:
Inconsistent terms:
explainedBy: APPLICATION_STD_1 (ApplicationVersion=1.0),
G1 (ApplicationVersion>1.0).
```

**Figure 8.5:** Explanations for the document of Figure §8.4

Instead of preparing the agreement offer from scratch, the customer may prepare its agreement offer in base of the valid published template. Obviously, an offer whose terms are described based on the terms and constraint of a published template has more probabilities to be accepted by the responder party than another offer developed from scratch.

Therefore, we consider that would be appealing to have an operation for the initiator such as `getCompliantOffer(iAgree template)` such that, given a template it provides
a basic compliant agreement offer including the template terms. The Document 8.3 shows the program that the consumer would need to execute. As the Document 8.4 denotes, the getCompliantOffer operation requires some auxiliary operations to extract parts of the iAgree documents such as getContext(iAgree document), getSDTs(iAgree document), getSRs(iAgree document) and getGTs(iAgree document). The compliant offer returned by the getCompliantOffer operation could be modified by the consumer in order to include additional conditions on guarantee terms. However, such new terms may impose non-compliant conditions with the template terms and creation constraints and thus, the use of aforementioned isValid and whyIsNotValid operations would be needed to solve the possible problems.

```java
File template = new File("./UNICORE6-COMPUTE.wsag");
AgOffer myCompliantOffer = getcompliantOffer(template);
```

Document 8.3: Java code to get a basic compliant offer from a template

```java
public AgOffer getCompliantOffer(Template t){
    AgOffer compliantOffer = createAgOffer("GeneratedCompliantOffer",
        //by default offer name
t.getContext(),
        //adapted to the offer
t.getSDTs(),
        //by default values
t.getSRs(),
t.getGTs());

    return compliantOffer;
}
```

Document 8.4: Java code of the getCompliantOffer operation

**Conclusions.** As a major conclusion we can claim that the provider and the consumer are benefited from the automated validity criteria checking provided by iAgree because of they assure that the interchanged documents are valid. Thus, the probability of having problems in the next step of the agreement creation process or even while service is being consuming, decreases. Furthermore, new auxiliary operations were identified such as: getCompliantOffer, getContext, getSDTs, getSRs and getGTs. These new operations have been generalized in order to be used in other scenarios.

**Backing up a SOA–Governance System**

**Problem to solve.** Other validation scenario with respect to the creation of agreements took place in the SOA–Governance System used by our regional governmental organization. The infrastructure of the regional governmental organization is distributed in the different departments both logically and physically and is usually managed autonomously in each location. During recent years there has been a shift toward SOA and currently there is an important number of core services replicated in the infrastructure with different QoS capabilities. From an architectural point of view, the
infrastructure is designed as a federated bus of services. In this context, each department represents a node with two main elements: An Enterprise Service Bus and the a SLA Management Infrastructure (SLAMI) system that provides different horizontal functionalities (such as monitoring, transactions or security). All the different nodes are integrated conforming the global infrastructure.

The different services are deployed in the bus and the consumers applications ask the bus for the appropriate provider. In Figure §8.6 an architectural conceptualization is shown: Each of the nodes correspond with a department; the nodes are composed of applications for end-users and the bus providing core common services. As depicted, the bus can provide access to external services from other nodes. Specifically, an SLAMI system is integrated in each node as part of the management system so each service consumption is based on the terms in an SLA. Such an SLA is created based on iAgree templates specified for the node manager including: the selected provider for the application and the concrete QoS properties in which the consumption is developed. Since such templates can be as complex as manager decides, an automated checking of its validity is very appealing from the manager’s point of view.

Checking and explaining validity criteria. We decided to deploy ADA as a Service (ADAaaS in Figure §8.6) to enable, to any node of the organization, the automated checking of iAgree validity criteria. Thus, the SLAMI system of any node is enable to create an SLA from preferences that are validated by using validated iAgree templates to describe them.

For instance, the template of Figure §8.7 describes preferences of Node M manager of Figure §8.6 to provide ADAaaS for other nodes as Node 1. The template includes as guarantee terms: a certain availability on predefined time periods, and both, a certificate and an interface version to support the service provisioning. Such template must

---

5Note that the template does not include real values for privacy reasons.
Figure 8.7: Template of Node_M to provide ADAaaS as a Service for Node_1

be validated because it is used by the SLAMI system of ESB_M to create the SLA that will supervise the ADA service provisioning through the federated bus. The template of Figure §8.7 represents our first try in defining the preferences for sharing ADA as a Service in the scenario, and although it seems to be valid at a first sight, it is conflicting for any certificate and availability period, defined in the document. Thus, after using the isValid and whyIsNotValid operations we obtain the following conflicts as explanation:

1. If the consumer node uses Cert1 certificate, G1 and G2 are inconsistent terms; but if other certificate is used by the consumer node G2 is a conditionally inconsistent term.

2. If the availability period is WorkingDays, G1 is a conditionally inconsistent term (except for Node 1 because G1 would be returned as inconsistent terms); and for any of the other availability periods, G2 is a conditionally inconsistent term.
Once the problems in the template of Figure §8.7 were solved, the SLAMI system could execute the program depicted in Document 8.5 in order to create an SLA from the template. Note that the program includes the previously identified \texttt{getCompliantOffer} operation, and another interesting auxiliary operation to get an SLA from an agreement offer (see program depicted in Document 8.6).

```java
File template = new File("./FASTServiceProvisioning.wsag");
AgOffer myCompliantOffer = getCompliantOffer(template);
Agreement sla = getAgreement(myCompliantOffer);
```

Document 8.5: Java code to generate the agreement from the template

```java
public Agreement getAgreement(AgOffer o){
    Agreement sla = createAgreement(o.getName(),
             o.getContext(),
             o.getSDTs(),
             o.getSRs(),
             o.getGTs());

    return sla;
}
```

Document 8.6: Java code of the getAgreement operation

**Conclusions.** As a major conclusion we can claim that in this SLA-driven solution, the node managers are benefited from using the automated checking of validity criteria detailed in Section §7.3 \texttt{isValid} and \texttt{whyIsNotValid} because they assure the validness of their preferences included in iAgree templates. Furthermore, during this evaluation we were aware of both, the unidentified kind of conflict that we coined as conditionally inconsistent term, and the inconsistent terms that do not wholly invalidate an agreement document. Moreover, the new auxiliary operation \texttt{getAgreement} was identified and generalized in order to be used in other scenarios.

**Backing up the SLA Learning**

**Problem to solve.** Similarly to the previous evaluation in the SOA–Governance System, we were aware of non-identified conflicts that turn iAgree documents to invalid. Thus, we decided to conduct an experiment with our M.Sc students. The students had to choose a real-world service such as PayPal, Amazon EC2, Business Process Management Systems or NetOpen EAI, and create its corresponding iAgree template and a compliant offer from scratch.

**Checking and explaining validity criteria.** In this case, iAgree Studio (see Section §8.4) was key to ease the edition and validation of iAgree documents. The by-default templates creation with empty sections for SDTs, SPs, GTs, and CCs was considered very useful for the students. Thus, a new auxiliary operation \texttt{createTemplate}(). Further-
more, a new auxiliary operation \texttt{getPricePolicyCCs} (Map price), was found also useful to provide a set of creation constraints from the price policy tables of some service providers\footnote{AmazonS3 published its price policy at \url{aws.amazon.com/es/s3/pricing/} and Google storage publishes it at \url{developers.google.com/storage/docs/pricing-and-terms}}.

A novel kind of conflict, coined by us as dead term, was identified in a couple of agreements. Figure §8.8 includes the first dead term (G2.1) what we were aware about. Notice that SLO of term G3 makes impossible QC of (G2.1) holds.

![Figure 8.8: Template modelled by the students for PayPal service with a dead term](image)

Templates with XOR compositors lead the students to find useful the possibility to know all their variants. Thus, we decided to make public the operation \texttt{getVariants} that was only used until now for normalization purposes as is described in Section §7.5.1.

\footnote{AmazonS3 published its price policy at \url{aws.amazon.com/es/s3/pricing/} and Google storage publishes it at \url{developers.google.com/storage/docs/pricing-and-terms}}
8.2. SUPPORTING SLA-DRIVEN SOLUTIONS

Conclusions. 46 iAgree documents\(^7\) were created during the experiment: 24 templates and 22 offers. Amongst them, there were documents with up to 14 service properties, 9 guarantee terms, 5 conditional guarantee terms, 12 variants, 4 items creation constraints or 14 creation constraints. We drew as a conclusion that real-world SLAs make an intensive use of qualify conditions and variants; and we also found another unidentified kind of conflict: the dead terms. Finally, three new auxiliary operations were found, namely, createTemplate, getPricePolicyCCs, and getVariants.

8.2.2 Agreement Monitoring SLA-driven Solutions

Problem to solve. We took the opportunity to validate our analysis operations in agreement monitoring, more specifically, in analyzing whether an agreement is fulfilling and in case it is being violated to know the causes.

In order to analyse the SLA fulfillment it is mandatory to monitor the quality of service offered by the provider meanwhile the service, subject of the SLA, is being consumed. The monitoring approach used was SALMon [101] due to several reasons. SALMon may operate on any available technology (SOAP-based, RESTful, etc.) with minor and localised changes; it may interoperate easily with other frameworks; it is easily extensible to monitor new metrics; and it combines passive monitoring and on-line testing [117] (cf. Appendix §F for more details).

However, SALMon monitor is configured manually with the endpoint and the metrics to monitor; it checks very simple SLOs included in ad-hoc non-WS–Agreement SLAs. Moreover, it does not provide explanations for the violations. Thus, as SALMon had room for improvement, we decided to collaborate with SALMon developers in the development of SALMonADA.

Currently, SALMonADA provides support for [89]: (1) monitoring iAgree SLAs with expressive SLOs and QCs; (2) configuring automatically the monitor configuration through a Monitoring Management Document (MMD) generated from the SLA information; and (3) validating the SLA fulfillment at runtime by checking if their terms are violated and, explaining the violations the causes with a report including the violated terms and the violating monitored measures. Note that the monitored measures were included in the MMD.

Leveraging iAgree analysis operations. Several auxiliary operations were needed during SALMonADA development. Firstly, an operation to generate a MMD document solely from the information included in the SLA (generateMMD(Agreement)). As Document 8.7 depicts, the MMD includes some information needed to configure the monitors, such as the service endpoint and the service properties, which must be monitored for each service operation. The generated MMD is used by SALMon to include the different monitored measures obtained at runtime.

\(^7\)These documents are available at http://www.isa.us.es/ada
Secondly, considering that SALMon has monitored the service and the MMD is updated with monitored measures, two analysis operations are needed to check the SLA fulfillment \(\text{isFulfilled}(\text{Agreement}, \text{MMD})\) and explain the causes of SLA violations \(\text{whyIsNotFulfilled}(\text{Agreement}, \text{MMD})\). Note that a monitoring period is needed to know when the monitoring must be performed, in this scenario we have considered that the monitoring period is the global period of the SLA to avoid the normalization of the validity periods.

Finally, as checking the SLA fulfillment can be interpreted as checking the compliance between the monitored measures and the SLA, our first approach to implement it by using \(\text{areCompliant} \) operation as is depicted in Document 8.8. Recently, we realized that there is a simpler, and computationally more efficient, way to carry out this checking. Considering that the monitored measures are a set of properties–value pairs, checking if these measures are violating the agreement amounts to check if the resulting of replacing the actual values of these properties in the SLA become it in an inconsistent SLA. Document 8.9 depicts this approach.

```java
public MMD generateMMD(Agreement sla){
    MMD generatedMMD = new MMD();
    generatedMMD.setEndpoint(sla.getEndpoint);
    List<SP> spList = sla.getSPs(); //getSPs() returns an ArrayList
    Iterator iter = spList.iterator();
    while (iter.hasNext()) {
        SP sp = iter.next();
        Scope scope = sp.getScope();
        String name = sp.getName();
        generatedMMD.addServiceMetric(scope, name);
    }
    return generatedMMD;
}
```

**Document 8.7: Java code of the getMMD operation**

```java
public boolean isFulfilled(Agreement sla, 
MMD mmd, 
ValidityPeriod monitoringPeriod){
//The monitoring period is considered to be the global period
Template pseudoTemplate = createTemplateFromSLA(sla);
//The CCs would include the SLA terms
AgOffer pseudoOffer = createAgOfferFromMMD(mmd);

List<Term> oTerms = pseudoOffer.getTerms();
List<CC> tCCs = pseudoTemplate.getCCs();

return areCompliant(oTerms, tCCs);
}
```

```java
public collection<Collection<AgElement>,ConflictType> whyIsNotFulfilled( 
    Agreement sla, 
    MMD mmd,
```
8.2. SUPPORTING SLA-DRIVEN SOLUTIONS

```java
//The monitoring period is considered to be the global period
Template pseudoTemplate = createTemplateFromSLA(sla);
    //The CCs would include the SLA terms
AgOffer pseudoOffer = createAgOfferFromMMD(mmd);

List<Term> oTerms = pseudoOffer.getTerms();
List<CC> tCCs = pseudoTemplate.getCCs();

return whyIsNotCompliant(oTerms, tCCs);
```

Document 8.8: Compliance-based approach for the SLA fulfillment

```java
public boolean isFulfilled(Agreement sla, MMD mmd, ValidityPeriod monitoringPeriod){
    //The monitoring period is considered to be the global period
    AgOffer mergedDoc = mergeSLAMMD(sla, MMD);
    //substitute the MMD monitored measures within the SLA

    return existInconsistencies(mergedDoc);
}
```

```java
public Collection<Collection<AgElement>,ConflictType> whyIsNotFulfilled(Agreement sla, MMD mmd, ValidityPeriod monitoringPeriod){
    //The monitoring period is considered to be the global period
    AgOffer mergedDoc = mergeSLAMMD(sla, MMD);

    return whyExistConflicts(mergedDoc);
}
```

Document 8.9: Consistency-based approach for the SLA fulfillment

Conclusions. Thanks to this scenario we have added two operations `isFulfilled` and `whyIsNotFulfilled` to our catalogue. In turn, these new operations require auxiliary operations, one of them is `generateMMD`, that provides the monitoring information to configure the monitors; and the others depends on the approach chosen to implement the SLA fulfillment. In the compliance-based approach, two auxiliary operations are needed: `createTemplateFromSLA` and `createAgOfferFromMMD`, which generate spurious documents from the SLA and the MMD, respectively. In turn, in the consistency-based approach it is needed the operation `mergeSLAMMD`, that replaces the monitored measures in the SLA.

Last, but not least, we have also realized that sometimes it is more interesting not to monitor the SLA as a whole, but to monitor some specific guarantee terms. In fact, this is more WS–Agreement-compliant because the WS–Agreement specification iden-
tifying three states what a guarantee term may evolves on: Not determined, Fulfilled and Violated (see Section §3.4). Thus, if we define operations to determine whether a guarantee term, or a set of them, are or not violated, we will provide very interesting operations for supporting WS–Agreement Protocol.

\[
isFulfilled(GTs,MMD,\text{MonitoringPeriod});
\]
\[
whyIsNotFulfilled(GTs,MMD,\text{MonitoringPeriod});
\]

8.3 ADA Catalogue of Analysis Operations

In this section we detail our reference implementation for the CSP-based technique detailed in Section §7.3. We have classified such an implementation in a novel catalogue of analysis operations that is summarised in Table §8.1. Such a catalogue improves the automated analysis of WS–Agreement documents and makes iAgree the first WSAC proposed with automated validity checking support, i.e., support for conflicts checking and explaining. Current implementation allows the specification of SLOs, QCs and CCs using logical operators, such as \(\land, \lor, \neg, \Rightarrow, \Leftrightarrow\), relational operators, such as \(=, \neq, >, \geq, <, \leq\), and algebraic operators defined on the domain of the SDT attributes, service properties and literals.

The analysis operations included in the catalogue are grouped by the interface in which they are included. The components and interfaces of IDEAS architecture are

![Figure 8.9: IDEAS components diagram.](image-url)
showed in Figure §8.9. Some of these interfaces are public and comprise analysis operations that are available for components out of the ADA Core, and the other interfaces are restricted to the ADA Core components.

<table>
<thead>
<tr>
<th>Analysis Operations Catalogue</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ICore public ops.</strong></td>
</tr>
<tr>
<td>existInconsistencies</td>
</tr>
<tr>
<td>whyExistInconsistencies</td>
</tr>
<tr>
<td>areCompliant(O,Ts,T.GTs)</td>
</tr>
<tr>
<td>areCompliant(O,Ts,T.CCs)</td>
</tr>
<tr>
<td>whyAreNotCompliant(O,Ts,T.GTs)</td>
</tr>
<tr>
<td>whyAreNotCompliant(O,Ts,T.CCs)</td>
</tr>
<tr>
<td><strong>IChecker public ops.</strong></td>
</tr>
<tr>
<td>existConflicts</td>
</tr>
<tr>
<td>existDeadTerms</td>
</tr>
<tr>
<td>existCondInconsisTerms</td>
</tr>
<tr>
<td>areCompliant(O,T)</td>
</tr>
<tr>
<td>whyExistConflicts</td>
</tr>
<tr>
<td>whyExistDeadTerms</td>
</tr>
<tr>
<td>whyExistCondInconsisTerms</td>
</tr>
<tr>
<td>whyAreNotCompliant(O,T)</td>
</tr>
<tr>
<td><strong>ISolve private ops.</strong></td>
</tr>
<tr>
<td>solve</td>
</tr>
<tr>
<td>explain</td>
</tr>
<tr>
<td>implies</td>
</tr>
<tr>
<td>whyNotImplies</td>
</tr>
<tr>
<td><strong>INormalisation &amp; IMap private ops.</strong></td>
</tr>
<tr>
<td>getVariants</td>
</tr>
<tr>
<td>temporalNormalisation</td>
</tr>
<tr>
<td>map</td>
</tr>
<tr>
<td><strong>IDocManager public ops.</strong></td>
</tr>
<tr>
<td>createTemplate</td>
</tr>
<tr>
<td>create...</td>
</tr>
<tr>
<td>exportToWSAG4jOffer</td>
</tr>
<tr>
<td>import...</td>
</tr>
<tr>
<td>addCCs</td>
</tr>
<tr>
<td>add...</td>
</tr>
<tr>
<td>getCompliantOffer</td>
</tr>
<tr>
<td>getAgreement</td>
</tr>
<tr>
<td>getAgVariants</td>
</tr>
<tr>
<td><strong>IFulfillment public ops.</strong></td>
</tr>
<tr>
<td>isFulfilled</td>
</tr>
<tr>
<td>whyIsNotFulfilled</td>
</tr>
<tr>
<td>generateMMD</td>
</tr>
<tr>
<td>mergeSLAMMD</td>
</tr>
<tr>
<td><strong>IVValidator public ops.</strong></td>
</tr>
<tr>
<td>isValid</td>
</tr>
<tr>
<td>whyIsNotValid(T)</td>
</tr>
<tr>
<td>whyIsNotValid(O,T)</td>
</tr>
</tbody>
</table>

Legend: Ts: terms, O:Offer, T:Term.

Table 8.1: Catalogue of analysis operations

In the following sections we explain the responsibilities of each component of the IDEAS architecture and the analysis operations included within their interfaces (cf. Figure §8.9).

### 8.3.1 iAgree Validity Criteria Checkers

This component supports the automated validation of iAgree documents by means of the interfaces *ICore* and *IChecker*. The implementation of both interfaces has been
thoroughly verified with a test suite comprising 1074 test cases (cf. Appendix §G). The $ICore$ interface depicted in Document 8.10 contains the core analysis operations whose semantics is described in Section §7.3.

```java
interface ICore {
    public boolean existInconsistencies(iAgreeDocument doc);
    public Collection<Collection<AgElement>> whyExistInconsistencies(
        iAgreeDocument doc);
    public boolean areCompliant(Collection<Term> oTerms, Collection<GT>tGTs);
    public Collection<Collection<AgElement>> whyAreNotCompliant(
        Collection<Term> oTerms, Collection<GT> tGTs);
    public Collection<Collection<AgElement>> whyAreNotCompliant(
        Collection<Term> oTerms, Collection<CC> tCCs);
}
```

Document 8.10: ICore interface

The $IChecker$ interface depicted in in Document 8.11 contains the analysis operations whose semantics is included in Sections §7.4 and §7.4.3.

```java
interface IChecker {
    public boolean existConflicts(iAgreeDocument doc);
    public boolean existDeadTerms(iAgreeDocument doc);
    public boolean existCondInconsisTerms(iAgreeDocument doc);
    public boolean areCompliant(Offer offer, Template template);
    public Collection<Collection<AgElement>,ConflictType> whyExistConflicts(
        iAgreeDocument doc);
    public Collection<Collection<AgElement>> whyExistDeadTerms(
        iAgreeDocument doc);
    public Collection<Collection<AgElement>> whyExistCondInconsisTerms(
        iAgreeDocument doc);
    public Collection<Collection<AgElement>> whyAreNotCompliant(
        Offer offer, Template template);
}
```

Document 8.11: IChecker interface in java

### 8.3.2 Choco Adapter

This component protects ADA from the variations due to changes in the CSP solver, as well as, support the interaction with the CSP solver, $Choco$ in the current version of ADA. It offers the interface $ISolve$, depicted in Document 8.12, which contains the core CSP operations whose semantics is included in Section §7.3 and Appendix §C.

```java
interface ISolve {
    public CSPSolution solve(CSP csp);
}
8.3. ADA CATALOGUE OF ANALYSIS OPERATIONS

```java
public Collection<CSPConstraint> explain(iAgreeDocument doc);
public boolean implies(Collection<CSPConstraint> c1,
    Collection<CSPConstraint> c2);
public Collection<Collection<CSPConstraint>> whyNotImplies(
    Collection<CSPConstraint> c1,
    Collection<CSPConstraint> c2);
```

Document 8.12: ISolve interface in java

8.3.3 iAgree Normalisation and CSP Mapping

These components supports the normalisation and mapping of iAgree documents by means of the interfaces INormalisation and IMap depicted in Document 8.13. Interface INormalisation provides the normalisation described in Section §7.5. The interface IMap supports the iAgree to CSP mapping included in Section §7.2.2.

```java
interface INormalisation {
    public Collection<Collection<GT>> getVariants(iAgreeDocument doc);
    public Collection<Collection<GT>> temporalNormalisation(
        Collection<Collection<GT>>);
}

interface IMap{
    public CSP map(iAgreeDocument doc);
}
```

Document 8.13: Interfaces INormalisation and IMap

8.3.4 iAgree Validator

This component supports the automation of validity criteria checking by means of the interface IValidator depicted in Document 8.14, whose semantics is described in Section §7.4.3. Depending on the input document or documents, it uses the required operations of the ICore and IChecker interfaces, in order to return whether the documents are valid or not and in this latter case, to provide and explanation.

```java
interface IValidator {
    public boolean isValid(Template t);
    public boolean isValid(AgOffer o, Template t);
    public Collection<Collection<AgElement>,ConflictType> whyIsNotValid(
        Template t);
    public Collection<Collection<AgElement>,ConflictType> whyIsNotValid(
        AgOffer o,
        Template t);
}
```

Document 8.14: IValidator interface in java
isValid. This operation implements the *valid* described in Section §7.3. Firstly, it normalizes the input document or documents and then, depending on the input document or documents, it uses the required operations of the *IChecker* interface, in order to return true if the documents are valid and false otherwise.

whyIsNotValid. This operation implements the *whyIsNotValid* described in Section §7.3. Firstly, it normalizes the input document or documents and then, depending on the input document or documents, it uses the required operations of the *IChecker* interface, in order to return the explanations for the conflicts and non-compliance of the input documents. Note that such an explanation is a set of agreement elements, either terms or creation constraints, that causes an inconsistency, a dead term, a conditionally inconsistent term, or a non-compliance.

8.3.5 Fulfillment Analyser

This component supports the SLA fulfillment analysis described in Section §8.2.2 by means of the interface *IFulfillment* depicted in Document 8.15. This component requires components that implement the interfaces *IValidator*, *IDocumentManager* and *ISearch*.

```java
interface IFulfillment {
    public MMD generateMMD(Agreement sla);
    public MMD simulateServiceConsumption(Agreement sla, ValidityPeriod monitoringPeriod);
    public boolean isFulfilled(Agreement sla, MonitoredData data, ValidityPeriod monitoringPeriod);
    public Collection<Collection<AgElement>, ConflictType> whyIsNotFulfilled(Agreement sla, MonitoredData data, ValidityPeriod monitoringPeriod);
    Template createTemplateFromSLA(Agreement sla);
    AgOffer createAgOfferFromMMD(MMD mmd);
    AgOffer mergeSLAMMD(Agreement sla, MMD mmd);
}
```

Document 8.15: IFulfillment interface in java

8.3.6 iAgree Document Manager

This component supports a set of management operations of iAgree documents by means of the interface *IDocManager* depicted in Document 8.16. This interface contains operations to create and import documents, create and delete elements such as GTs, CCs, and other interesting operations. In the following, we describe some of them.
createTemplate. This operation creates a by-default template with empty sections for SDTs, SRs, GTs, and CCs, and by-default name, if it does not receive input parameters. If one or more of the input parameters is null, an empty section is created for them. Finally, an operation to create a template from a normalised document is included. In the complete interface other similar methods to create offers and agreement are included.

exportToWSAG4JOffer. This operation creates an agreement offer in the WSAC of WSGA4J. Similar operations are included to create templates in the WSAC of WSAG4J and to create iAgree documents from tWSAG4J documents.

getCompliantOffer. This operations create a by-default offer compliant with the template that receives as parameter. If a condition is included it is considered in the created offer. For instance to create an offer with a set of GTs that minimize the service price or response time.

getVariants and getAgVariants. These operations are needed to provide normalized agreement documents and set of guarantee terms for the external components due to the getVariants operation is provided by the INormalisation interface just for the ADA Core components.

```java
interface IDocManager {
    public Template createTemplate();
    public Template createTemplate(String name,
                                    Context context, Collection<SDT> sdts,
                                    SR sr, Collection<SP> sps,
                                    Collection<GT> gts, Collection<CC> ccs);
    public Template createTemplate(Collection<Collection<GT>>);  
    public WSAG4JOffer exportToWSAG4JOffer(AgOffer o);
    public iAgreeDocument createGT(iAgreeDocument doc,
                                    String obligated,SLO slo,
                                    String id,
                                    QC qc, Scope scope,
                                    ValidityPeriod lp, BVL bvl);  //optional
    public Template addCCs(Template t, Collection<CC> ccs);
    public AgOffer getCompliantOffer(Template t);
    public AgOffer getCompliantOfferForOptimalTemplate(Template t,
                                                       OptimalCondition condition);
    public Agreement getAgreement(AgOffer o);
    public Collection<iAgreeDocument> getAgVariants(iAgreeDocument doc);
    public Collection<Collection<GT>> getVariants(iAgreeDocument doc);
}
```

Document 8.16: IDocManager interface in java
8.3.7 iAgree Search Engine

This component includes some interesting operations to retrieve terms, part of terms, CCs, and context information. Some conditions can be imposed to the returned elements. For instance, a `getTerms` operation is included that takes as inputs both: (1) an iAgree document, and (2) a specific condition that must identify a set of terms un-ambiguously; and the operation returns such a set of terms. For instance, those terms of the document whose scope is a specific service operation. Note that this condition is useful for the previous fulfillment checking operations in order to identify which of the terms must be monitored for a certain operation.

8.4 iAgree Studio

In order to ease the development of iAgree-based SLA-driven solutions, we have incorporated a web application called iAgree studio. iAgree studio, does not only provide an iAgree editor, but it also provides several facilities to validate and analyze documents by using the operations provided by ADA, which have been detailed in Section §8.3. Its user interface is summarized in the diagram of Figure §8.10. As figure §8.10 denotes, iAgree studio presents the information according to the kind of document loaded. Its main features are the following.

1. Basic file management operations, *File* menu depicted in the Figure, including:

   • *create by-default iAgree documents* (offers, templates, or agreements) to make easier its description from scratch.
   
   • *create iAgree-based scenarios* to launch several analysis operations by using a simple Javascript syntax. Two scenarios are detailed later in the section (cf. Figures §8.12 and §8.13).
   
   • perform typical file system operations with the documents and scenarios such as: *open, save, save as, print, or rename*.
   
   • *download documents and scenarios* in several formats, namely, iAgree, WS–Agreement serialised XML-based, FAST preferences [39].

2. Common advanced operations that apply to any iAgree document, *Main window* menu, including:

   • Document edition including facilities such as *undo, redo, copy, paste, and find*.
   
   • Document validation by means of a unique *validate* button that performs a different checking and explaining depending on the kind of iAgree document currently loaded.
• Variants extraction of an iAgree document. The elements can be ordered by several conditions such as the number of comprised terms, the penalties of comprised terms, etc.

• Elements extraction by using the options included in the Search menu, such as: Context, Terms, SDTs, service properties, and GTs. The elements can be ordered by several conditions depending on: penalties, the scope of a GT, their validity on a specific validity period, the relation with a service property, etc.

• Support for typical CRUD operations on GTs, i.e. to create a by-default empty GT with its mandatory and optional elements (cf. Section §6.4) and to read, update, or delete an specific GT.

• Checking if a condition on service properties or SDTs is included in the document, and to show exactly where, in case it were included.

3. Template-specific operations that were included in the Tool menu when a template is loaded. The supported operations are:

   • To obtain the list of CCs, that can be ordered by several conditions, depending on: the relation with a service property or SDT attribute, to show them from the less to the more restrictive, etc.

   • To create a new CC.

   • To validate its compliance with a specific agreement offer.

   • To create a by-default compliant offer for the template. Several conditions can be imposed on such created compliant offer. For instance, to maximize or minimize penalties, rewards, service properties, or SDT attributes. In addition, the conditions could include FAST preferences to consider them in the creation of the compliant offer.

   • To create price policy CCs from price tables.

4. Offer-specific operations that included in the Tool menu when an agreement offer is loaded. The supported operations are:

   • To validate its compliance with a specific template.

   • To create an agreement from the agreement offer.

5. To perform agreement-specific operations that were included in the Tool menu when an agreement is loaded. The supported operations are:

   • To obtain the parties involved in the agreement.

   • To obtain the information needed to monitor the agreement (cf. Appendix §F). These information includes the service endpoint, and the service properties and the service operations they are related to.

   • To create a simulated service execution log.
• To analyze the agreement fulfillment. Such an analysis may check if the agreement is violated for the simulated service execution log, or a log reported by a monitoring tool. In case of violations, an explaining report is showed with the violated terms and the violating measures.

The former web application that inspired iAgree studio is available on-line and can be used from any platform including mobile devices. Such former version of the iAgree studio uses the WSDL interface to consume ADA as a service and it offers both, (1) an edit of iAgree documents called iAgree editor, and (2) an iAgree scenarios developer.

The iAgree editor depicted in Figure §8.11 provides the following features:

1. Support for handling iAgree documents with a line numbered editor and syntax highlighting for the iAgree keywords ((1) in the Figure).

2. Skeletons for the creation of new iAgree documents has been included (it appears by pressing the New template/offer option of File menu). These skeletons have specially developed for users not familiar with WS–Agreement structure and iAgree syntax ((2) in the Figure).

3. Several iAgree documents have been preloaded in order to serve as examples and to reduce the learning curve ((3) in the Figure).

4. Support for the most relevant ADA analysis operations to check and explain the iAgree documents validity criteria ((4) in the Figure). Note that the analysis operations showed in the menu depends on the kind of iAgree document that is open in the editor panel. Thus, the three operations depicted the Figure appears for an iAgree template, but an agreement offer that is based on a template will show two analysis operations more, namely, the compliance checking and the non-compliance explanation.

5. The result of the iAgree documents checking and explaining is shown in a message (the message shown in the Figure at (5) corresponds to explaining the inconsistencies of an iAgree document).

The iAgree scenarios developer, that is depicted in Figure §8.12, allows to launch several analysis operations consecutively by using a simple JavaScript notation, and the result is shown in a log window. Note that this developer scenario allows experimenting with more than one analysis operation. In the sample scenario of Figure §8.12, a different message is shown in the log window depending on the result of analysing an

8 http://labs.isa.us.es/apps/iagreeditor/
9 http://www.isa.us.es:8081/ADAService?wsdl
iAgree document\textsuperscript{10}. Thus, the application shows if the document includes inconsistencies or not, and then, if it is needed, an explanation for the inconsistencies found in the document.

In Figure §8.13, an interesting scenario is included to assure that a template and agreement offer are conflict-free, before checking its compliance. In the log windows it is show that the documents are not compliant and the corresponding explanation.

\section*{8.5 \textbf{Summary}}

In this chapter we have introduced IDEAS, our integrated development environment to ease the development of iAgree-based SLA-driven solutions. Specifically, we have presented its layer structure and the main features its provides. In addition, we have exposes some SLA-driven solutions whose development have been made easier by using the iAgree automated validity checking. These solutions provide a set of new analysis and auxiliary operations that enrich the ADA catalogue of analysis operations included within IDEAS. The ADA catalogue is detailed in the chapter by explaining the components of the IDEAS architecture and the interfaces that provide the operations. Finally, we have presented the iAgree studio developed to make easier the edition and analysis of advanced SLA-driven solutions.

\textsuperscript{10}The document used in the example of Figure §8.12 includes a partial inconsistency similar to such depicted in Figure §6.27 but applied to a translation service providing scenario
Figure 8.10: iAgree Studio user interface navigability.
Figure 8.11: Edition capabilities of the iAgree front-end to try ADA
Figure 8.12: An iAgree scenario launched to explain conflicts if they are found.

Figure 8.13: An iAgree scenario to explain compliance problems.
Part IV

Final Remarks
Conclusions and Future Work

Where a calculator on the ENIAC is equipped with 18,000 vacuum tubes and weighs 30 tons, computers in the future may have only 1,000 vacuum tubes and perhaps weigh 1.5 tons.

Anonymous (1949),
Popular Mechanics

I was expecting applause but I suppose stunned silence is equally appropriate.

Sheldon Lee Cooper,
Fictional character on The Big Bang Theory

At this point we can affirm that we have improved the automated analysis of WS–Agreement documents. In Section §9.1 we expose the conclusions we have come to the end of this dissertation. Finally, in Section §9.2 we discuss some of the decisions we have made in this dissertation highlighting its main limitations and possible extensions.
CHAPTER 9. CONCLUSIONS AND FUTURE WORK

9.1 Conclusions

The main conclusion of this dissertation is:

Main Conclusion

The current support to develop SLA-driven solutions can be enhanced with IDEAS.

We are convinced that our dissertation is only a first, baby step, but it is in the right way. We are contributed in three key points: the language (iAgree), the software development kit (the analysis operations catalogue) and the development environments (IDEAS). These three aspects determine the capabilities of many software development tools, and to the best of our knowledge, in the case of SLA-driven solutions we are pioneers in providing a novel integrated development tool. It has been a long path to run, but we are convinced that our research strategy was the right one.

Throughout this dissertation we took some decisions that lead us to achieve our goal successfully. Firstly, we decided not reinventing the wheel and we bet for WS-Agreement. Many times we were about to abandon this decision, but at the end we have realized that in spite of the fact it is a specification hard to understand without examples, hardly supported by the promoters, it is a good enough start point. Secondly, we decided to validate our work in real-world scenarios, what has supposed an important extra effort. It also forces us to develop publicly-available tools to show our progress in any time and any device. Furthermore, some of IDEAS end-users needs help us to identify new and interesting analysis operations. Finally, we decide to leverage our knowledge in automated analysis what lead us to the need of identifying core operations and working with normalized documents. At following we expose some more specific conclusions.

Regarding SLA specification, iAgree has resulted really useful as fully-fledged language. Specially interesting has been the definition of the iAgree validity criteria on the basis of a novel catalogue of conflicts. In these sense, we found that the use of qualifying conditions and term compositors are at the root of two new kinds of conflicts: dead terms and conditionally inconsistent terms. Furthermore, the use of optional terms, by means of term compositors, leads to situations where the conflicts affect only partially the agreement document. In turn, in order to deal with partial conflicts the notion of variants in a WS-Agreement document have been rigorously defined. These findings are valid, not only for WS-Agreement documents, but for any SLA specification model incorporating conditional and optional terms.
iAgree was conceived as an intermediate fully-fledged specification language for SLAs, i.e. to serve as the target domain into which other WSACs and SLAs should be translated in order to be benefited by its automated analysis support. Our experience leveraging iAgree in WSAG4J project allow us to claim that our initial conception has been found useful for other colleagues. Regardless of whether the translation is manual or automated, the approach works and it is probably enough for research purposes. However, our strategy for automating the analysis of iAgree allows us to consider that it can be also considered as a mechanism to leverage the powerful of IDEAS. We refer to reuse our conceptual framework as a guideline for endowing with similar capabilities to SLA specification languages regardless of they are or not WSAC.

Regarding automated analysis, our strategy consisted in relying solely on a small set of CSP-based techniques: finding solutions, explaining the lack of solutions, the constraint implication, and explaining of the non-compliance. These techniques are generally incorporated by many solvers which means our solution can be easily shared and reproduced. On top these CSP-based reasoning operations we have defined four core operations. In turn, on top these core operations we have defined a set of higher level analysis operations which is inherently open–ended. As result, we have obtained a set of operations that may help not only in the validation, but also in the edition, monitoring and creation of iAgree documents; and potentially on any needs that can be interpreted as information extraction. For instance, we have developed operations to get a compliant offer for a given template, or to generate the monitoring management information for a given SLA.

With respect to the applicability of the developed technique, it goes beyond iAgree and could be potentially applicable for any WSAC. The only requirement is that the semantics of the languages used in the WS–Agreement document can be interpreted as a CSP. This requirement is quite reasonable since, according to [116], it can be assumed that for automated analysis purposes any SLA can be regarded as a CSP regardless of the specification language.

Regarding the SLA-driven solutions support, IDEAS makes easier the development of SLA-driven solutions and we have validated it in advanced agreement creation and monitoring scenarios. IDEAS has been incrementally developed with a set of facilities to edit and analyse, not only iAgree documents, but also that we call *scenarios*, that allows to launch several analysis operations by using a simple Javascript syntax. However, we must validate it with more SLA-driven solutions, and some facilities must be included in the tool suite that integrates IDEAS in order to enhance its usefulness. In the following Section §9.2 we discuss some of these potential enhancements.

As a final conclusion we conjecture that both, the use of ad-hoc WSACs and the lack of a commonly accepted WSAC, are delaying the adoption of WS–Agreement by researchers and practitioners. In this regard, we are confident that iAgree will provide a foundation on which SLA-driven solutions focused in WS–Agreement can be built. Furthermore, apart from our technique’s inherent value, having developed IDEAS as open source tooling support which can be quickly integrated and reused and easy-to-
use is a determining factor to achieve a useful result and settles the basis to spread the use of WS–Agreement. As an example, the development of the SALMonADA core (see Section §8.2.2 and Appendix §F) took only 3 weeks.

9.2 **Discussion, limitations and extensions**

In this section we discuss some of the decisions we have made in this dissertation highlighting its main limitations and possible extensions.

Regarding the specification language, although iAgree is more human-readable than other XML-based SLA models, it admits an important room of improvement before considering it as an end-user notation. Our next step in this point is to explore linguistic and templates patterns used long time ago in Requirements Engineering, in SLAs [115], and in process performance indicators in business process management [31].

Current support of the so-called Business Value List in WS–Agreement allows to specify penalties, rewards, or preferences, between others. However, neither iAgree nor other WSAC explore this element of WS–Agreement. In this line it is not only need to devise specific sublanguages for describing them, but also devise validity criteria, and review the notion of compliance accordingly.

Regarding the analysis, although in the dissertation we provide a way to describe time-aware documents with iAgree, such kind of documents are not currently supported by the analysis operations included in the proposed catalogue. At the moment, in our on-going work we have studied how to: (1) detect and explain conflicts within time-aware WS–Agreement documents; and (2) detect conflicts in the validity periods definitions. The process of Figure §9.1 depicts the steps needed to deal with time-aware documents by using some of the temporal algorithms that normalize the validity periods proposed in [68]. This is considered for a further extension of our work because the technique depicted in the process is currently in an initial development state, and the other analysis operations do not currently support time-aware iAgree documents as input. Nevertheless, as Figure §9.2 denotes, we have developed an editor as proof-of-concept with support for time-aware iAgree documents.

Another very promising work line is the definition of new analysis operations such as those focused in knowing the cost of violating an SLA or a subset of guarantee terms; those focused on creating an optimal offer from a given template according to a given optimization criteria and last, simulating the execution of a service; but not least, those operations that can be applied on a set of SLAs: knowing the total cost of violating a set of SLAs, computing the SLA or template that a vendor should prepare taking into account that the service to be offered needs a number of services regulated by SLAs, etcetera.
Regarding SLA-driven solutions support, we are very interested in adapting SAL-MonADA in order to be able to monitor a set of guarantee terms instead of the SLA as a whole, and taken into account the validity periods. Finally, we are currently devising how to integrate FAST in IDEAS. We are sure, FAST will be a perfect companion for the future of IDEAS.
Figure 9.2: iAgree editor with analysis support for time-aware documents.
PART V

APPENDICES
iAgree Abstract Syntax

In this appendix we detail the abstract syntax of iAgree WSAC by using the EBNF notation. We use the natural language to clarify some elements of the EBNF.

Document:

\[ \text{document} ::= \text{identSec contextSec termsSec creationConstraintSec} \]

iAgree, our proposed WS–Agreement configuration, supports the three kinds of document mentioned in the specification: templates, offers, and agreements. All of them sharing the abstract syntax with the exception of a possible creation constraints section in templates.

Context:

\[ \text{contextSec ::= [initiatorIdent} [responderIdent] providerRole] \]

Where the provider role can be agreement initiator or responder

Terms:

\[ \text{termsSec ::= ident All } \{ \text{term} \} + \]
\[ \text{term ::= serviceTermSec | guaranteeTermSec} \]
\[ \{ \text{term} \} + \] | \[ \text{OneOrMore} \{ \text{term} \} + \] | \[ \text{ExactlyOne} \{ \text{term} \} + \]

Where All, OneOrMore, and ExactlyOne are an and–like, or–like and xor–like terms compositors, respectively.

ServiceTerm:

\[ \text{serviceTermSec ::= termName serviceName} \]
\[ \text{(serviceDescriptionExpression} \] | \[ \text{variableSet} \] | \[ \text{reference} \]
\[ \text{variableSet ::= } \{ \text{varName location Metric} \} + \]

Where location is an XPATH to locate a variable value, Metric is described in a suitable metric expression, and reference denotes where the service is described.
GuaranteeTerm:

guaranteeTermSec ::= termName obligated \{scope\} * [qualifyingExpression]
  serviceLevelObjective businessValueList

serviceLevelObjective ::= sloExpression |
  | (keyPerformanceIndicator targetValue)

businessValueList ::= [importance] \{penalty\} * \{reward\} * \{preference\}
  \{businessValueExpression\}*

penalty, reward ::= (assessmentTime | assessmentServiceCount)
  \{Unit\} Amount

preference ::= \{serviceTermReference\} * \{utility\}* 

Where the obligated may be the provider or the consumer, the scope denotes which service or service operation is affected by the guarantee term. The qualifyingExpression and the sloExpression are assertions, where the qualifying enables the SLO. The importance value denotes a relative importance between GTs. The preference is expressed by referencing a service term with an associated utility to such term. The businessValueExpression describes any business value.

CreationConstraints:

creationConstraintSec ::= \{Item\} * \{constraintExpression\}*

Item ::= name location itemConstraint

Where constraintExpression is an assertion delimiting a template compliance, location is an XPATH to locate an item value, and itemConstraint is an XSD restriction.
The iAgree concrete syntax has been designed to be human-readable, although it is not as understandable as the natural language for non technical users, as the example of Figure §B.1 denotes, it is merely a first step in order to avoid the use of the XML tags of the specification. We have used such a syntax in our user interfaces of WS–Agreement tools. At following we detail this human-readable concrete syntax by using the EBNF notation. Note that ident is an universal identifier that denotes a unique element.


document ::= identSec contextSec termsSec creationConstraintSec

identSec ::= Template ident
         | Agreement ident
         | AgreementOffer ident [for Template ident]

contextSec ::= partiesRoles [expirationTime] [globalPeriod] [definedPeriod] metrics
partiesRoles ::= Provider ident as Responder; 'Consumer ident';
               | Provider ident; 'Consumer ident as Responder';
expirationTime ::= Expiration Time' : date,'
date ::= digit digit '/' digit digit '/' digit digit digit

globalPeriod ::= Global Period' : period',
definedPeriod ::= Defined Period' : {ident' : period;'}*
period ::= [from hour..'hour] [on weekDay' -' weekDay] during date..'date
         | ident {(and | except) ident}
         | ident {(and | except) (ident | period)}*
hour ::= digit digit '/' digit digit
weekDay ::= 'SUN' | 'MON' | ... | 'SAT'

metrics ::= Metrics' : ['uri{' '/uri} * '{metricDef}]*
uri ::= (see RFC6874 [126])
metricDef ::= metricName : dataType;'
metricName ::= ident
dataType ::= boolean
         | integer
         | float
         | float'{'digit} * '..{'digit} *'}'
         | set {'ident {,ident} * '}'
         | enum {'ident {,ident} * '})
termsSec ::= AgreementTerms [ident] {terms} +
terms ::= serviceTerm
| serviceProperties
| guaranteeTermSec
| All‘ : {terms} +
| One Or More between‘ : {terms} +
| Exactly One between‘ : {terms} +

serviceTerm ::= Service ident [available at uri ] [operations]
serviceDescription ::= Global Description {ident [‘ = value] ;’}* |
| Description for ident‘ : {ident [‘ = value] ;’}* |
value ::= ident | {digit} +
serviceProperties ::= Monitorable Properties propScope
propScope ::= global‘ : {ident‘ ;’} +
| for ident‘ : {ident‘ ;’} +

guaranteeTermSec ::= Guarantee Terms guaranteeTerm
| guaranteeTerm

guaranteeTerm ::= ident ‘ : (Provider | Consumer) Guarantees guarantee
| [localPeriod] [scope] [qualifyingCondition] [importance]
| {penalty} * {reward} * [utility] {businessValue} *
guarantee ::= slo ‘ ;’
| kpi ‘ : target‘ ;’
slo ::= predicate
predicate ::= predicate opL (predicate | predTerm)
opL ::= ‘and’ | ‘or’ | ‘xor’ | ‘not’ | implies’
predTerm ::= predExpression opC predExpression
opC ::= ‘ = ’ | ‘ ! = ’ | ‘ > ’ | ‘ >= ’ | ‘ < ’ | ‘ <= ’
predExpression ::= predExpression opA (predExpression | ident | ‘true’ | ‘false’)
opA ::= ‘ + ’ | ‘ – ’ | ‘ * ’ | ‘ / ’
kpi, target ::= ident
localPeriod ::= on (ident | period)’ ;’
scope ::= upon {ident} + ‘ ;’
qualifyingCondition ::= onlyIf predicate‘ ;’
importance ::= relative importance‘ : (ident | {digit})* ;’
penalty ::= subject to a penalty of expression [unit] per assessment‘ ;’
reward ::= subject to a reward of expression [unit] per assessment‘ ;’
expression ::= (ident | {digit})*
unit, assessment ::= ident
businessValue ::= utility | customValue
utility ::= utility expression [if selected‘ : ident] ;’
customValue ::= business value‘ : expression‘ ;’
creationConstraintSec ::= Creation Constraints\[constraint\] * [localPeriod]
  [qualifyingCondition];

constraint ::= ident\[ident\] belongs \{'ident \{ident\} * \}'
  | ident \ predicate

iAgree can be serialised in XML following the mapping included in Tables §B.1 and §B.2. Such a serialisation is used in our proposed automated technique described in the Chapter §8. The following document includes the template of Figure §2.2 but written in the XML-based serialised syntax of iAgree.

```xml
<wsag:Template wsag:templateId="1.0" xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <wsag:Terms wsag:Name="AWS-S3">
    <wsag:All>
      <wsag:ServiceDescriptionTerm wsag:Name="SDT_AWS-S3" wsag:ServiceName="AWS-S3">
        <OfferItem name="Region" wsag:Metric="metrics/countries"></OfferItem>
        <OfferItem name="Interface" wsag:Metric="metrics/interfaceType"></OfferItem>
        <OfferItem name="DataDurability" wsag:Metric="metrics/float">99.99999999999</OfferItem>
        <OfferItem name="RNS" wsag:Metric="metrics/boolean">false</OfferItem>
        <OfferItem name="StorageSize" wsag:Metric="metrics/size"></OfferItem>
        <OfferItem name="FirstProject" wsag:Metric="metrics/boolean"></OfferItem>
        <OfferItem name="TotalPrice" wsag:Metric="metrics/float"></OfferItem>
        <OfferItem name="SupportPlanPrice" wsag:Metric="metrics/float"></OfferItem>
      </wsag:ServiceDescriptionTerm>

      <wsag:ServiceProperties wsag:Name="SP_AWS-S3" wsag:ServiceName="AWS-S3">
        <wsag:VariableSet>
          <wsag:Variable wsag:Name="MUP" wsag:Metric="metrics/percent"></Variable>
          <wsag:Location name="MUP"></Location>
        </wsag:Variable>
        <wsag:Variable wsag:Name="TransferredGb" wsag:Metric="metrics/float"></Variable>
        <wsag:Location name="TransferredGb"></Location>
        <wsag:Variable wsag:Name="ResponseTime" wsag:Metric="metrics/float"></Variable>
        <wsag:Location name="ResponseTime"></Location>
        <wsag:Variable wsag:Name="ReadRequests" wsag:Metric="metrics/integer"></Variable>
        <wsag:Location name="ReadRequests"></Location>
        <wsag:Variable wsag:Name="WriteRequests" wsag:Metric="metrics/integer"></Variable>
        <wsag:Location name="WriteRequests"></Location>
        <wsag:Variable wsag:Name="OnlineReportingSupport" wsag:Metric="metrics/boolean"></Variable>
        <wsag:Location name="OnlineReportingSupport"></Location>
        <wsag:Variable wsag:Name="TurnAroundTime" wsag:Metric="metrics/integer"></Variable>
        <wsag:Location name="TurnAroundTime"></Location>
        <wsag:Variable wsag:Name="PhoneSupport" wsag:Metric="metrics/boolean"></Variable>
        <wsag:Location name="PhoneSupport"></Location>
      </wsag:VariableSet>
    </wsag:ServiceProperties>
  </wsag:All>
  <wsag:Context>
    <wsag:AgreementResponder name="Provider"></wsag:AgreementResponder>
  </wsag:Context>
</wsag:Terms>
</wsag:Template>
```
<wsag:GuaranteeTerm wsag:Obligated="ServiceProvider" wsag:Name="G1">
  <wsag:ServiceLevelObjective>
    <wsag:CustomServiceLevel>MUP\geq99.9</wsag:CustomServiceLevel>
  </wsag:ServiceLevelObjective>
</wsag:GuaranteeTerm>

<wsag:GuaranteeTerm wsag:Obligated="ServiceConsumer" wsag:Name="G2">
  <wsag:ServiceLevelObjective>
    <wsag:CustomServiceLevel>(TransferedGbMUP\&(StorageSize*100) AND ReadRequests\lt; WriteRequests)</wsag:CustomServiceLevel>
  </wsag:ServiceLevelObjective>
</wsag:GuaranteeTerm>

<wsag:GuaranteeTerm wsag:Obligated="ServiceProvider" wsag:Name="G3">
  <wsag:QualifyingCondition>
    Interface="SOAP"
  </wsag:QualifyingCondition>
  <wsag:ServiceLevelObjective>
    <wsag:CustomServiceLevel>ResponseTime\lt;1000</wsag:CustomServiceLevel>
  </wsag:ServiceLevelObjective>
</wsag:GuaranteeTerm>

<wsag:GuaranteeTerm wsag:Obligated="ServiceProvider" wsag:Name="G4">
  <wsag:QualifyingCondition>
    Interface="REST"
  </wsag:QualifyingCondition>
  <wsag:ServiceLevelObjective>
    <wsag:CustomServiceLevel>ResponseTime\lt;700</wsag:CustomServiceLevel>
  </wsag:ServiceLevelObjective>
</wsag:GuaranteeTerm>

<wsag:OneOrMore>
  <wsag:GuaranteeTerm wsag:Obligated="ServiceProvider" wsag:Name="G10.1">
    <wsag:ServiceLevelObjective>
      <wsag:CustomServiceLevel>OnlineReportingSupport="True"</wsag:CustomServiceLevel>
    </wsag:ServiceLevelObjective>
    <wsag:GuaranteeTerm>
      <wsag:ServiceLevelObjective>
        <wsag:CustomServiceLevel>TurnAroundTime=15</wsag:CustomServiceLevel>
      </wsag:ServiceLevelObjective>
    </wsag:GuaranteeTerm>
  </wsag:GuaranteeTerm>
  <wsag:GuaranteeTerm wsag:Obligated="ServiceProvider" wsag:Name="G10.2">
    <wsag:ServiceLevelObjective>
      <wsag:CustomServiceLevel>PhoneSupport="True"</wsag:CustomServiceLevel>
    </wsag:ServiceLevelObjective>
    <wsag:GuaranteeTerm>
      <wsag:ServiceLevelObjective>
        <wsag:CustomServiceLevel>ExtendedSupport="True"</wsag:CustomServiceLevel>
      </wsag:ServiceLevelObjective>
    </wsag:GuaranteeTerm>
  </wsag:GuaranteeTerm>
</wsag:OneOrMore>
</wsag:GuaranteeTerm>

<wsag:CreationConstraints>
  <wsag:Constraint>
    <Name>C1</Name>
    <Content>(Region = "US") OR (Region = "EU_Irland")</Content>
  </wsag:Constraint>
  <wsag:Constraint>
    <Name>C2</Name>
    <Content>(FirstProject=true AND StorageSize <= 5) IMPLIES TotalPrice=0</Content>
  </wsag:Constraint>
  <wsag:Constraint>
    <Name>C3</Name>
    <Content>RRS = "True" IMPLIES (StoragePrice = (0.05 * StorageSize))</Content>
  </wsag:Constraint>
</wsag:CreationConstraints>
<wsag:Constraint>
  <Name>C4</Name>
  <Content>RRS = "False" IMPLIES (StoragePrice = (0.12 * StorageSize))</Content>
</wsag:Constraint>

<wsag:Constraint>
  <Name>C5</Name>
  <Content>StorageSize &lt;= 5000</Content>
</wsag:Constraint>

<wsag:Constraint>
  <Name>C6</Name>
  <Content>({FirstProject = "False" OR StorageSize &gt; 5) IMPLIES (TotalPrice = StoragePrice + SupportPlanPrice)}</Content>
</wsag:Constraint>

<wsag:Constraint>
  <Name>C7</Name>
  <Content>RRS = "True") IMPLIES (DataDurability&gt;=99.99)</Content>
</wsag:Constraint>

<wsag:Constraint>
  <Name>C8</Name>
  <Content>RRS = "False") IMPLIES (DataDurability&gt;=99.999999999)</Content>
</wsag:Constraint>

<wsag:CreationConstraints>
<wsag:Template>

iAgree (XML) document B.1: Template of Figure §2.2 in the concrete XML-based syntax
### WS–Agreement XML Element

<table>
<thead>
<tr>
<th>Template/Agreement/AgreementOffer Id=&quot;id&quot;</th>
<th>iAgree Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Name&gt;name&lt;/...&gt;</td>
<td>Template id - name</td>
</tr>
<tr>
<td>&lt;Context&gt;</td>
<td>AgreementOffer id - name for Template id - name</td>
</tr>
<tr>
<td>&lt;AgreementInitiator&gt;initiator name&lt;/...&gt;</td>
<td>Context:</td>
</tr>
<tr>
<td>&lt;AgreementResponder&gt;responder name&lt;/...&gt;</td>
<td>Provider: initiator/responder name (as Responder)?,</td>
</tr>
<tr>
<td>&lt;ExpirationTime&gt;datetime&lt;/...&gt;</td>
<td>Consumer: initiator/responder name (as Responder)?,</td>
</tr>
<tr>
<td>&lt;TemplateName&gt;templateName&lt;/...&gt;</td>
<td>ExpirationTime: datetime?,</td>
</tr>
<tr>
<td>&lt;AgreementOffer id=&quot;id&quot; for Template id=&quot;name&quot;&gt;</td>
<td>Metrics:</td>
</tr>
<tr>
<td>&lt;Context&gt;</td>
<td>metric:: datatype,</td>
</tr>
<tr>
<td>&lt;AgreementInitiator&gt;AgreementInitiator</td>
<td>metric:: datatype,</td>
</tr>
<tr>
<td>&lt;Provider&gt;AgreementResponder</td>
<td>AgreementResponder</td>
</tr>
<tr>
<td>&lt;Consumer&gt; initiator/responder name &lt;/...&gt;</td>
<td>AgreementTerms</td>
</tr>
<tr>
<td>&lt;ExpirationTime&gt;datetime&lt;/...&gt;</td>
<td>Service(1) at Service Reference(1)</td>
</tr>
<tr>
<td>&lt;TemplateName&gt;templateName&lt;/...&gt;</td>
<td>Monitorable Properties(1)</td>
</tr>
<tr>
<td>&lt;Context&gt;</td>
<td>Guarantee Terms</td>
</tr>
</tbody>
</table>
| <Terms Name="terms set name">          | GuaranteeTerm(1),...
| <All>                                  | ServiceReferences(1) |
| <OneOrMore/ExactlyOne>...</...>        | GuaranteeTerm(1),...
| <ServiceDescriptionTerm>(1)...</...>   | + |
| <ServiceProperties>(1)...</...>        | GuaranteeTerm(1),...
| <ServiceReferences>(1)...</...>        | + |
| <ExpirationTime>datetime</...>          | GuaranteeTerm(1),...
| <TemplateName>templateName</...>       | + |
| <Context>                               | Consumer/Provider guarantees(1),...
| <Terms Name="terms set name">          | Creation Constraints |
| <All>                                  | + |
| <OneOrMore>                            | |
| <ExactlyOne>                           | |

<table>
<thead>
<tr>
<th>&lt;CreationConstraints&gt; &lt;/Template/Agreement/AgreementOffer&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;All&gt;</td>
</tr>
<tr>
<td>term/terms compositor1,...</td>
</tr>
<tr>
<td>term/terms compositor2,...</td>
</tr>
<tr>
<td>...&lt;/...&gt;</td>
</tr>
<tr>
<td>&lt;CreationConstraints&gt;</td>
</tr>
<tr>
<td>&lt;Item Name=&quot;item,&quot;&gt;...&lt;/...&gt;</td>
</tr>
<tr>
<td>&lt;Constraint&gt;</td>
</tr>
<tr>
<td>&lt;Name&gt;constraint1&lt;/...&gt; &lt;Content&gt;GC expression1&lt;/...&gt;</td>
</tr>
<tr>
<td>&lt;/Constraint&gt;</td>
</tr>
<tr>
<td>&lt;/CreationConstraints&gt;</td>
</tr>
</tbody>
</table>

### Table B.1: WS–Agreement schema to iAgree mapping: Metadata and Compositors

| (1) The mapping of these terms is included in Table §B.2 |

---

180
<table>
<thead>
<tr>
<th>WS–Agreement XML Element</th>
<th>iAgree Element</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;ServiceDescriptionTerm Name=&quot;TermName&quot; ServiceName=&quot;service&quot;&gt;</code></td>
<td>Service service available at reference</td>
</tr>
<tr>
<td><code>&lt;OfferItem name=&quot;attribute1&quot;&gt;value1&lt;/...&gt;,...</code></td>
<td>Operations: op1</td>
</tr>
<tr>
<td><code>&lt;OfferItem name=&quot;attribute2&quot;&gt;value2&lt;/...&gt;</code></td>
<td>Global Description: attribute1; [= value1]</td>
</tr>
<tr>
<td><code>&lt;ServiceDescriptionTerm&gt;</code></td>
<td>Description for op1: attribute2; [= value2]</td>
</tr>
<tr>
<td><code>&lt;ServiceReference Name=&quot;TermName&quot; ServiceName=&quot;service&quot;&gt;</code></td>
<td><code>&lt;ServiceReference Name=&quot;TermName&quot; ServiceName=&quot;service&quot;&gt;</code></td>
</tr>
<tr>
<td>&quot;reference&quot;</td>
<td>Description for op1:</td>
</tr>
<tr>
<td><code>&lt;ServiceReference&gt;</code></td>
<td><code>&lt;ServiceReference&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ServiceProperties Name=&quot;TermName&quot; ServiceName=&quot;service&quot;&gt;</code></td>
<td>Monitorable Properties:</td>
</tr>
<tr>
<td><code>&lt;VariableSet&gt;</code></td>
<td><code>&lt;VariableSet&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Variable Name=&quot;var1&quot; Metric=&quot;metric1&quot;&gt;</code></td>
<td><code>&lt;Variable Name=&quot;var1&quot; Metric=&quot;metric1&quot;&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Location&gt;location1&lt;/...&gt;</code></td>
<td><code>&lt;Location&gt;location1&lt;/...&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Variable&gt;</code></td>
<td><code>&lt;Variable&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Variable Name=&quot;var2&quot; Metric=&quot;metric2&quot;&gt;</code></td>
<td><code>&lt;Variable Name=&quot;var2&quot; Metric=&quot;metric2&quot;&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Location&gt;location2&lt;/...&gt;</code></td>
<td><code>&lt;Location&gt;location2&lt;/...&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Variable&gt;</code></td>
<td><code>&lt;Variable&gt;</code></td>
</tr>
<tr>
<td><code>&lt;VariableSet&gt;</code></td>
<td><code>&lt;VariableSet&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ServiceProperties&gt;</code></td>
<td><code>&lt;ServiceProperties&gt;</code></td>
</tr>
<tr>
<td><code>&lt;GuaranteeTerm Name=&quot;TermName&quot;&gt;</code></td>
<td><code>&lt;GuaranteeTerm Name=&quot;TermName&quot;&gt;</code></td>
</tr>
<tr>
<td><code>Obligated=&quot;ServiceProvider/Consumer&quot;&gt;</code></td>
<td><code>&lt;GuaranteeTerm Name=&quot;TermName&quot;&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ServiceScope ServiceName=&quot;service&quot;&gt;</code></td>
<td><code>&lt;ServiceScope ServiceName=&quot;service&quot;&gt;</code></td>
</tr>
<tr>
<td><code>scope&lt;/...&gt;*</code></td>
<td><code>scope&lt;/...&gt;*</code></td>
</tr>
<tr>
<td><code>&lt;QualifyingCondition&gt;QC expr&lt;/...&gt;?</code></td>
<td><code>&lt;QualifyingCondition&gt;QC expr&lt;/...&gt;?</code></td>
</tr>
<tr>
<td><code>&lt;ServiceLevelObjective&gt;</code></td>
<td><code>&lt;ServiceLevelObjective&gt;</code></td>
</tr>
<tr>
<td>`&lt;CustomServiceLevel&gt;SLO expr&lt;/...&gt;</td>
<td><code>&lt;CustomServiceLevel&gt;SLO expr&lt;/...&gt;</code></td>
</tr>
<tr>
<td><code>&lt;KPIName&gt;KPI&lt;/...&gt;</code></td>
<td><code>&lt;KPIName&gt;KPI&lt;/...&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Target&gt;target&lt;/...&gt;</code></td>
<td><code>&lt;Target&gt;target&lt;/...&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ServiceLevelObjective&gt;</code></td>
<td><code>&lt;ServiceLevelObjective&gt;</code></td>
</tr>
<tr>
<td><code>&lt;BusinessValueList&gt;</code></td>
<td><code>&lt;BusinessValueList&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Importance&gt;importance&lt;/...&gt;?</code></td>
<td><code>&lt;Importance&gt;importance&lt;/...&gt;?</code></td>
</tr>
<tr>
<td><code>&lt;Penalty&gt;</code></td>
<td><code>&lt;Penalty&gt;</code></td>
</tr>
<tr>
<td><code>&lt;AssessmentInterval&gt;</code></td>
<td><code>&lt;AssessmentInterval&gt;</code></td>
</tr>
<tr>
<td>`&lt;TimeInterval&gt;time&lt;/...&gt;</td>
<td><code>&lt;TimeInterval&gt;time&lt;/...&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Count&gt;count&lt;/...&gt;</code></td>
<td><code>&lt;Count&gt;count&lt;/...&gt;</code></td>
</tr>
<tr>
<td><code>&lt;AssessmentInterval&gt;</code></td>
<td><code>&lt;AssessmentInterval&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ValueUnit&gt;unit&lt;/...&gt;</code></td>
<td><code>&lt;ValueUnit&gt;unit&lt;/...&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ValueExpr&gt;expr&lt;/...&gt;</code></td>
<td><code>&lt;ValueExpr&gt;expr&lt;/...&gt;</code></td>
</tr>
<tr>
<td><code>&lt;Penalty&gt;</code></td>
<td><code>&lt;Penalty&gt;</code></td>
</tr>
<tr>
<td><code>*</code></td>
<td><code>*</code></td>
</tr>
<tr>
<td><code>&lt;Reward&gt;...&lt;/...&gt;*</code></td>
<td><code>&lt;Reward&gt;...&lt;/...&gt;*</code></td>
</tr>
<tr>
<td><code>&lt;Preference&gt;</code></td>
<td><code>&lt;Preference&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ServiceTermReference&gt;ref&lt;/...&gt;*</code></td>
<td><code>&lt;ServiceTermReference&gt;ref&lt;/...&gt;*</code></td>
</tr>
<tr>
<td><code>&lt;Utility&gt;utility&lt;/...&gt;*</code></td>
<td><code>&lt;Utility&gt;utility&lt;/...&gt;*</code></td>
</tr>
<tr>
<td><code>&lt;Preference&gt;</code></td>
<td><code>&lt;Preference&gt;</code></td>
</tr>
<tr>
<td><code>&lt;CustomBusinessValue&gt;customBV&lt;/...&gt;</code></td>
<td><code>&lt;CustomBusinessValue&gt;customBV&lt;/...&gt;</code></td>
</tr>
<tr>
<td><code>&lt;BusinessValueList&gt;</code></td>
<td><code>&lt;BusinessValueList&gt;</code></td>
</tr>
<tr>
<td><code>&lt;GuaranteeTerm&gt;</code></td>
<td><code>&lt;GuaranteeTerm&gt;</code></td>
</tr>
</tbody>
</table>

Table B.2: WS–Agreement schema to iAgree mapping: Terms and Creation Constraints
Template AmazonS3 version 1.3 //fragment of 1.0 with validity periods

Provider Amazon as Responder;
Consumer Me;
Global Period: during 01/01/2012..12/31/2013 except from 23:00 to 00:00 on SUN;
Defined Period:
FreeTrialQuota: during 01/01/2012..12/31/2012;
Metrics [iAgree.generalMetrics, Amazon.metrics]

AgreementTerms
Service AWS-S3 available at. aws.amazon.com/s3
Global description
Region;
Interface;
DataDurability;
RRS;
StorageSize;
FirstProject;
TotalPrice;

Monitorable Properties
global:
MUP;
ResponseTime;

Guarantee Terms
G1: Provider guarantees MUP >= 99.9;
on GlobalPeriod;

G2: Consumer guarantees TransferredGb < StorageSize * 100
AND ReadRequests > WriteRequests;
on GlobalPeriod;

G3: Provider guarantees ResponseTime < 1000;
on GlobalPeriod;
onlyIf (Interface = SOAP);

G4: Provider guarantees ResponseTime < 700;
on GlobalPeriod;
onlyIf (Interface = REST);
...

Creation Constraints:
C1: Region belongs {US, EU_Ireland};
C2: TotalPrice = 0
onlyIf (FreeTrialQuota
onlyIf (FirstProject and StorageSize <= 5)
...

Figure B.1: Excerpt of AmazonS3 service scenario template in iAgree
**C.1 Introduction**

Constraint programming is a mature field in artificial intelligence [62]. The expressive power of *Constraint Satisfaction Problems* (CSPs) and the wide catalogue of open-source solvers has turned them into one the most used paradigms to deal with automated analysis in several fields such as the feature models [132] or SLAs [107, 108]. Constraint programming is a constantly evolving research topic where many algorithms and heuristics are arising yearly to improve the performance of commercial and free solvers.

A CSP is a declarative paradigm to model and solve problems using constraints [135]. It is defined as a 3–tuple \((V,D,C)\) where \(V\) is a set of variables, each ranging on a finite domain from set \(D\), and \(C\) is a set of constraints restricting the values that the variables can take simultaneously. A solution to a CSP is an assignment to each variable of a value from its corresponding domain so that all constraints are satisfied simultaneously.

Consider for instance, the CSP: \((\{a,b\},\{\{0,1,2\},\{0,1,2\}\},\{a+b<4\})\) where both variables \(a\) and \(b\) take value in the domain \(\{0,1,2\}\) and are constrained by \(a+b<4\). The only value assignment that does not satisfy \(a+b<4\) is \(\{a \mapsto 2, b \mapsto 2\}\), so there are eight solutions.

The next section details some operations to obtain conclusions from CSPs.

**C.2 Useful CSP Operations**

There are four basic operations to obtain conclusions from CSPs:

**Operation 1 - Searching for all the solutions.**
This operation searches for all the value assignments that satisfy all the constraints in the CSP. For the given example, this operation obtains 8 solutions.

**Operation 2 - Searching for one random solution (a.k.a solve).**
This operation searches for one solution that satisfies all the constraints. It is a non-deterministic operation since the solution obtained depends on the search algorithms which can use random criterion or heuristics to search for a solution.
Operation 3 - Satisfiability (a.k.a. sat, simply).
A CSP is satisfiable if there is at least one valid value assignment. This operation detects if a CSP is valid or not. An example of inconsistent CSP take is $\{(a, b), \{0,1,2\}, \{0,1,2\}\{a + b < 0\}$ since there is no possible value assignment satisfying the constraints.

Operation 4 - Constraint propagation.
Searching for an equivalent CSP such that the available values in the domain are reduced according to the constraints in the CSP. For example, this operation obtains for an input CSP $\{(a, b), \{0,1,2\}, \{0,1,2\}\{a + b < 2\}$, another equivalent CSP in the form $\{(a, b), \{0,1\}, \{0,1\}\{a + b < 2\}$ since it is not possible to find for any solution that assigns 2 as a value for either $a$ or $b$.

In many real–life applications, it is needed to find a good solution to a CSP rather than anyone. A solution quality or goodness is usually measured by an application–dependent function called objective function. The goal is finding a solution that satisfies all the constraints and minimise or maximise the objective function. Such problem is known as a Constraint Optimisation Problem (COP) is a 4-tuple $\langle V, D, C, O \rangle$ that adds an optimisation function $O$ to a CSP. An objective function maps every solution in the CSP to a numerical value that is used for maximisation or minimisation. The adoption of COPs allow to define a new operation:

Operation 5 - Optimisation.
This operation searches for the solution/s that satisfy all the constraints in the CSP and minimise or maximise the objective function. If we define a COP $\langle \{a, b\}, \{0,1,2\}, \{0,1,2\}\{a + b < 4\}$ where the optimisation function is $O(s) = a$, which maximises the value of $a$. There are two solutions in the original CSP $\{\{a \mapsto 2, b \mapsto 0\}, \{a \mapsto 2, b \mapsto 1\}\}$, that maximises the value of the objective function and are therefore the solutions of the COP.

Moreover, in other real–life applications, it is needed to know the reason why a CSP is not satisfiable. The goal is finding the CSP constraints that cannot be satisfied, or in other words, finding which constraints must be removed from the CSP to make it satisfiable. Several heuristics can be used to find the constraints that explain why a CSP is not satisfiable, for instance by using the constraint propagation operation as U. Junker does in its quickxplainer proposal [53]. Thus, a new operation can be defined as follows:

Operation 6 - Explanation (a.k.a. explain, simply).
This operation searches for a minimal set of constraints $c \subseteq C$ that makes it impossible to find a valid assignment of all elements in $V$ that satisfies $c$, i.e., that makes $\text{solve}(V, D, c) = \emptyset$. For instance, for the CSP: $\langle \{a, b, d\}, \{0..2\}, \{0..2\}, \{0..2\}\{a + b < 1, a = 1, d > 1\}$, the resulting $c$ would be $\{a + b < 1, a = 1\}$ because the minimum allowed value for $b$ is 0.
**Operation 7 - Constraint Implication (a.k.a. implies, simply).**

Let $\delta$ be a CSP of the form $(V,D,C_\delta)$, and let $\alpha$ be a CSP of the form $(V,D,C_\alpha)$. The solution space of $\delta$ is a subset of the solution space of $\alpha$ iff the CSP defined as $(V,D,C_\delta \rightarrow C_\alpha)$ is satisfiable, or iff its equivalents CSP $[64] V,D,C_\delta \land \neg C_\alpha$ is not satisfiable. For instance, the solution space of $(\{a,b\},\{0,1,2\},\{0,1,2\}\{a+b < 3\})$ is a subset of the solution space of $(\{a,b\},\{0,1,2\},\{0,1,2\}\{a+b < 4\})$, because $(\{a,b\},\{0,1,2\},\{0,1,2\}\{(a+b < 3) \rightarrow (a+b < 4)\})$ is satisfiable. A possible solution for such a constraint implication is $a = 0$, and $b = 1$.

**Operation 8 - not Constraint Implication (a.k.a. whyNotImplies, simply).**

This operation is under review in a on-going technical report [91]. In such a technical report we are proposing a generalisation from the code developed for the whyIsNotCompliant operation that is showed in Document C.1.

```java
public void execute(ChocoAnalyzer choco) {
    explanations = new HashMap<AgreementError, Explanation>();
    int size = this.docs.size();
    if (size == 2) {
        Template t;
        AgreementOffer o;
        AbstractDocument doc1 = docs.get(0), doc2 = docs.get(1);

        if (doc1 instanceof Template && doc2 instanceof AgreementOffer) {
            t = (Template) doc1;
            o = (AgreementOffer) doc2;
            Context context = o.getContext();
            String tName = t.getName(), tId = t.getId();
            String refName = context.getTemplateName();
            String refId = context.getTemplateId();
            if (!tName.equalsIgnoreCase(refName) || !tId.equalsIgnoreCase(refId) ||
                !validScopes(o, t)) {
                AgreementError error = new AgreementError(Utils.single2Collection(
                    (AgreementElement) context));
                Explanation exp = new Explanation(new LinkedList<AgreementElement>());
                explanations.put(error, exp);
            }

            Collection<AbstractDocument> altsTemplate = getAlternatives(t, choco, false);
            Collection<AbstractDocument> altsOffer = getAlternatives(o, choco, false);
            Iterator<AbstractDocument> itTemplate = altsTemplate.iterator();
            while (itTemplate.hasNext()) {
                Template template = (Template) itTemplate.next();
                Iterator<AbstractDocument> itOffer = altsOffer.iterator();
                while (itOffer.hasNext()) {
                    AgreementOffer offer = (AgreementOffer) itOffer.next();
                }
            }
        }
    }
}
```
Map<ServiceScope, AbstractDocument> offerViews = getViews(offer, choco);
Map<ServiceScope, AbstractDocument> tempViews = getViews(template, choco);
Map<AbstractDocument, AbstractDocument> matchedViews = matchViews(offerViews, tempViews);
Set<Entry<AbstractDocument, AbstractDocument>> entries = matchedViews.entrySet();

for (Entry<AbstractDocument, AbstractDocument> e : entries) {
    AgreementOffer viewOffer = (AgreementOffer) e.getKey();
    Template viewTemp = (Template) e.getValue();
    ChocoComplianceOp complianceOp = new ChocoComplianceOp();
    complianceOp.addDocument(viewTemp);
    complianceOp.addDocument(viewOffer);
    complianceOp.execute(choco);
    boolean compliant = complianceOp.isCompliant();

    if (!compliant) {
        ChocoTranslator trans1 = new ChocoTranslator(viewTemp);
        trans1.translate();
        ChocoComplianceTranslator trans2 = new ChocoComplianceTranslator(viewOffer, trans1.getChocoVars(), trans1.getServicePropsMap());
        trans2.translate();
        Collection<Variable> chocoVars = trans2.getChocoVars().values();
        Map<AgreementElement, Constraint> templateConstraints = trans1.getChocoConstraints();
        Map<AgreementElement, Constraint> offerConstraints = trans2.getChocoConstraints();
        Collection<Variable> usedVars = extractUsedVars(chocoVars, offerConstraints);
        allTemplateConstraints = templateConstraints.values();
        allOfferConstraints = offerConstraints.values();
        Map<AgreementError, Explanation> res = quickxplain(usedVars, offerConstraints, templateConstraints);

        if (level == REFINE_ALL) {
            res = refineAgreementErrors(res, offerConstraints, trans1.getChocoConstraints());
            res = refineExplanations(res, offerConstraints, trans1.getChocoConstraints());
        } else {
            if (level == REFINE_OFFER) {
                res = refineAgreementErrors(res, offerConstraints, trans1.getChocoConstraints());
            } else if (level == REFINE_TEMPLATE) {
                res = refineExplanations(res, offerConstraints, trans1.getChocoConstraints());
            }
        }
        explanations.putAll(res);
    }
}
else {
    System.err.println("The first document must be a template
          and the second one must be an offer");
}
}

---
iAgree (XML) document C.1: Part of the java code developed for whyIsNotCompliant operation
FORMER DEFINITIONS FOR THE VALIDITY CRITERIA CHECKING

Current Appendix includes the complete examples exposed in Figures of the automated validity criteria checking of Section §7.3. Note that the included rigorous definitions follows the style used in previous works [89, 90] in order to compare with the new formalisation style followed in the dissertation.

D.1 EXAMPLES FOR DEAD TERMS SECTION

Previous definition style to detect a dead term: A guarantee term is a dead term if it can never be applied if all of the mandatory terms of the agreement are fulfilled, i.e., if its QC can never be true provided that the other terms of the agreement are fulfilled. Therefore, to detect that a term is dead, we just have to check whether its QC contradicts the remaining terms of the agreement. This can be expressed in terms of a CSP as follows: let GTi be a guarantee term whose qualifying condition is QC_{GTi}, GTi is a dead term if adding its QC as a new constraint to the document it makes it inconsistent:

\[\text{dead}(\text{GT}_i, \Delta) \iff \text{solve}(\text{map}(\Delta)) \neq \emptyset \land \text{solve}(V, D, (C \setminus \text{map}(\text{GT}_i) \cup \text{QC}_{\text{GT}_i})) = \emptyset\]  \hspace{1cm} (D.1)

The dead term of template of Figure §D.1 is detected as follows:
\textbf{APPENDIX D. FORMER DEFINITIONS FOR THE VALIDITY CRITERIA CHECKING}

\begin{verbatim}
solve\{\text{TransferredGb,StorageSize,ReadRequests,WriteRequests,ResponseTime,MUP}\}, \\
\{\text{float, float, integer, integer, float, float[0..100]}\}, \\
\{(\text{TransferredGb} < \text{StorageSize} * 100) \text{ and (ReadRequests} > \text{WriteRequests)}, \\
(\text{WriteRequests} > \text{ReadRequests}) \Rightarrow \text{ResponseTime} < 0.1, \text{ MUP >= 99.9}\} \neq \emptyset \\
\wedge solve\{\text{TransferredGb,StorageSize,ReadRequests,WriteRequests,ResponseTime,MUP}\}, \\
\{\text{float, float, integer, integer, float, float[0..100]}\}, \\
\{(\text{TransferredGb} < \text{StorageSize} * 100) \text{ and (ReadRequests} > \text{WriteRequests)}, \\
(\text{WriteRequests} > \text{ReadRequests}), \text{ MUP} > 99.9\} = \emptyset
\end{verbatim}

Previous definition style to explain dead terms:

\[\text{dead}_{\text{exp}}(\text{GT}_i, \Delta) = \text{explain}(V, D, (C \setminus \text{map}((\text{GT}_i) \cup \text{QC}_{\text{GT}_i}))) = c\] (D.2)

The previous example would be explained as follows:

\begin{verbatim}
explain\{\text{TransferredGb,StorageSize,ReadRequests,WriteRequests,ResponseTime,MUP}\}, \\
\{\text{float, float, integer, integer, float, float[0..100]}\}, \\
\{(\text{TransferredGb} < \text{StorageSize} * 100) \text{ and (ReadRequests} > \text{WriteRequests)}, \\
(\text{WriteRequests} > \text{ReadRequests}), \text{ MUP} > 99.9\} = \\
\{(\text{TransferredGb} < \text{StorageSize} * 100) \text{ and (ReadRequests} > \text{WriteRequests)}, \\
(\text{WriteRequests} > \text{ReadRequests})\}
\end{verbatim}

\section*{D.2 Examples for Cond. Incons. Terms Section}

\begin{figure}[h]
\centering
\begin{verbatim}
Template AmazonS3 version 1.7
...
G1: Provider guarantees MUP >= 99.9;
...
G6: Consumer guarantees StorageSize >= 30
...
G12: Consumer guarantees StorageSize < 20;
    onlyIf (Region NOT US);
...
\end{verbatim}
\caption{Document with a conditionally inconsistent GT}
\end{figure}

Previous definition style to detect a conditionally inconsistent term: \textit{A guarantee term is a conditionally inconsistent term if when its qualifying condition is true (i.e., it is enabled), its service level objective is always false (i.e., it cannot be fulfilled). Consequently, to
detect that a term is conditionally inconsistent, we have to check whether its qualifying condition and service level objective contradict each other taking into account the other terms of the agreement. In terms of a CSP, this can be expressed as follows: let $GT_i$ be a guarantee term whose qualifying condition is $QC_{GT_i}$ and service level objective is $SLO_{GT_i}$, $GT_i$ is a conditionally inconsistent term if:

$$
condInconsistent(GT_i, \Delta) \iff \exists D, (C \setminus map(GT_i) \cup QC_{GT_i} \cup SLO_{GT_i}) = \emptyset
$$

The conditionally inconsistent term of offer of Figure §D.2 would be detected, by means of its CSP mapping as follows:

$$
solve(\{StorageSize, MUP, Region\}, \{float, float[0..100], set\{US, west-US, EU_Ireland, Pacific_Asia, Southamerica, GovCloud_US\}\},
\{Region = US \Rightarrow StorageSize >= 30, StorageSize < 30, MUP >= 99.9\} ) \neq \emptyset
\land solve(\{StorageSize, MUP, Region\}, \{float, float[0..100], set\{US, west-US, EU_Ireland, Pacific_Asia, Southamerica, GovCloud_US\}\},
\{Region = US, StorageSize >= 30, StorageSize < 30, MUP >= 99.9\} ) = \emptyset
$$

Previous definition style to explain conditionally inconsistent terms:

$$
condInconsistent_{exp}(GT_i, \Delta) = explain(V, D, (C \setminus map(GT_i) \cup QC_{GT_i} \cup SLO_{GT_i})) = c
$$

The, previous example would be explained as follows:

$$
explain(\{StorageSize, MUP, Region\}, \{float, float[0..100], set\{US, west-US, EU_Ireland, Pacific_Asia, Southamerica, GovCloud_US\}\},
\{Region = US, StorageSize >= 30, StorageSize < 30, MUP >= 99.9\} ) =
\{StorageSize >= 30, StorageSize < 30\}
$$

D.3 EXAMPLES FOR COMPLIANCE SECTION

Previous definition style to detect a non-compliance between a template and an agreement offer: There is a non-compliance between an agreement offer and a template when one or more of the agreement offer terms contradict one or more of the template terms or creation constraints. This can be expressed in terms of a CSP as follows\(^1\): let $\alpha$ be an agreement offer and $\theta$ a template, $\alpha$ is non-compliant with $\theta$ if the following condition is fulfilled:

\(^1\)We have inspired in the conformance definition of A. Ruiz-Cortés thesis [115]
### Template AmazonS3 version 1.0

#### Guarantee Terms
- **G1**: Provider guarantees $\mu \geq 99.9$; subject to a penalty of 1 Service Credit per 1 failed request; // simplified penalty
- **G2**: Consumer guarantees $\text{TransferredGb} < \text{StorageSize} \times 100$ and $\text{ReadRequests} > \text{WriteRequests}$;
- **G3**: Provider guarantees $\text{ResponseTime} < 1000$; only if $(\text{Interface} = \text{SOAP})$;
- **G4**: Provider guarantees $\text{ResponseTime} < 700$; only if $(\text{Interface} = \text{REST})$;
- ... G5-G9 omitted for spaces reasons ...
- **G10**: One or More between:
  - **G10.1**: Provider guarantees $\text{OnlineReportingSupport} = \text{true}$;
  - **G10.2**: Provider guarantees $\text{TurnAroundTime} = 15$;
  - **G10.3**: Provider guarantees $\text{PhoneSupport} = \text{true}$;
  - **G10.4**: Provider guarantees $\text{ExtendedSupport} = \text{true}$;

#### Creation Constraints:
- **C1**: Region belongs \{US, EU_Ireland\};
- **C2**: $\text{TotalPrice} = 0$; on FreeTrialQuota;
  - only if $(\text{FirstProject} = \text{true} \text{ and } \text{StorageSize} \leq 5)$;
- **C3**: $\text{StoragePrice} = 0.05 \times \text{StorageSize}$; only if $\text{RRS} = \text{true}$;
- **C4**: $\text{StoragePrice} = 0.12 \times \text{StorageSize}$; only if $\text{RRS} = \text{false}$;
- **C5**: $\text{StorageSize} \leq 5000 \text{ TB}$;
- **C6**: $\text{TotalPrice} = \text{StoragePrice} + \text{SupportPlanPrice}$; only if $(\text{FirstProject} = \text{false} \text{ or } \text{StorageSize} > 5)$;
- **C7**: DataDurability $\geq 99.99$; only if $(\text{RRS} = \text{true})$;
- **C8**: DataDurability $\geq 99.99999999$; only if $(\text{RRS} = \text{false})$;

Figure D.3: Excerpt of the Template of AmazonS3 service scenario of Figure §2.2
nonCompliant(\(\alpha, \theta\)) \Leftrightarrow \text{nonCompliant(map}(\alpha) \Rightarrow \text{map}(\theta)) = \\
solve(\text{map}(\alpha) \land \neg\text{map}(\theta)) \neq \emptyset \quad (D.5)

V = \{ \text{Region, RRS, Interface, DataDurability, StorageSize, FirstProject, StoragePrice, TransferredGb, ReadRequests, WriteRequests, ...} \}
D = \{ \text{set\{US, west\_US, EU\_Ireland, Pacific\_Asia, Southamerica, GovCloud\_US\}, boolean, enum\{SOAP, REST\}, float[0..100], float, boolean, float, float, integer, integer, ...}\},
C = \{ \text{(TransferredGb < StorageSize * 100) AND (ReadRequests > WriteRequests), Region in\{US, EU\_Ireland\}, RRS \Rightarrow StoragePrice = 0.05 * StorageSize, not RRS \Rightarrow StoragePrice = 0.12 * StorageSize, StorageSize <= 5000, ...}\},

Figure D.4: Excerpt of the CSP generated from template of Figure §D.3.

<table>
<thead>
<tr>
<th>AgreementOffer</th>
<th>AmazonS3NonCompliantOffer</th>
<th>version 1.2</th>
<th>for Template</th>
<th>AmazonS3</th>
<th>version 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>AgreementTerms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service</td>
<td>AWS-S3 available at. aws.amazon.com/s3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Global description</td>
<td>Region = US;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interface = SOAP;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DataDurability = 99.99999999;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RRS = False; //Reduced Redundancy Storage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>StorageSize = 500 GB;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FirstProject = False;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TotalPrice = ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>StoragePrice = 40; //non-compliance cause 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SupportPlanPrice = ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G2: Consumer guarantees</td>
<td>TransferredGb &lt; StorageSize * 100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AND ReadRequests &lt; WriteRequests //non-compliance cause 2;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G3: Provider guarantees</td>
<td>MLP &gt;= 99.99;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure D.5: Agreement offer non-compliant with template of Figure §6.30

Figures §D.4 and §D.6 includes the CSPs returned by the map function when the compliance is being checked for the examples of Figures §D.3\(^2\) and §D.5, respectively. In the example the agreement offer is not compliant with the template due to the existence of solutions for the CSP generated by the implication operation: \(\text{solve}(\text{map}(\alpha) \land \neg\text{map}(\theta)) \neq \emptyset\) as follows:

\(^2\)Note that the CSP mapped from the template does not include all template elements, but only such relevance for the offer terms, by simplicity.
\[ V = \{ \text{Region}, \text{RRS}, \text{Interface}, \text{DataDurability}, \text{StorageSize}, \text{FirstProject}, \text{StoragePrice}, \text{TransferredGb}, \text{ReadRequests}, \text{WriteRequests}, \ldots \} \]

\[ D = \{ \text{set\{US, west\_US, EU\_Ireland, Pacific\_Asia, Southamerica, GovCloud\_US\}}, \text{boolean}, \text{enum\{SOAP, REST\}}, \text{float[0..100]}, \text{float, boolean, float, float, integer, integer, \ldots} \}, \]

\[ C = \{ \text{Region = US, Interface = SOAP, DataDurability = } 99.999999999, \text{RRS = False, StorageSize = 500, FirstProject = False, StoragePrice = 40,} \]
\[ \text{TransferredGb < StorageSize * 100) AND (ReadRequests < WriteRequests), \ldots} \}, \]

\[ \text{solve( \{} \text{Region, RRS, Interface, DataDurability, StorageSize, FirstProject, StoragePrice,} \]
\[ \text{TransferredGb, ReadRequests, WriteRequests, \ldots} \}, \]
\[ \{ \text{set\{US, west\_US, EU\_Ireland, Pacific\_Asia, Southamerica, GovCloud\_US\}}, \text{boolean}, \text{enum\{SOAP, REST\}}, \text{float[0..100]}, \text{float, boolean, float, float, integer, integer, \ldots} \}, \]
\[ \{ \text{Region = US, Interface = SOAP, DataDurability = } 99.999999999, \text{RRS = False, StorageSize = 500, FirstProject = False, StoragePrice = 40,} \]
\[ \text{TransferredGb < StorageSize * 100) AND (ReadRequests < WriteRequests),} \]
\[ \neg ( \text{TransferredGb < StorageSize * 100) AND (ReadRequests > WriteRequests),} \]
\[ \text{Region in\{US, EU\_Ireland\}, RRS } \Rightarrow \text{StoragePrice = } 0.05 \ast \text{StorageSize,} \]
\[ \neg \text{RRS } \Rightarrow \text{StoragePrice = } 0.12 \ast \text{StorageSize, StorageSize } < = 5000 \} \} \neq \emptyset \]

In order to explain the non compliance between the documents of the example, we apply and \textit{areCompliant} operation mentioned in Section §7.3.3 and a first group of conflicting constraints would be discovered when the constraints were grouped by the following three variables: RRS, StoragePrice, StorageSize. At this step, the following constraints would be grouped:

- In the template: RRS \(\Rightarrow\) StoragePrice = 0.05 \ast StorageSize, not RRS \(\Rightarrow\) StoragePrice = 0.12 \ast StorageSize, StorageSize \(<\) = 5000.

- In the offer: RRS = False, StorageSize = 500, StoragePrice = 40.

In that case, after a recursive call to \textit{areCompliant} with the extracted groups to refine the explanation, we would get as explanation \{not RRS \(\Rightarrow\) StoragePrice = 0.12 \ast StorageSize\} from template and \{RRS = False, StorageSize = 500, StoragePrice = 40\} from offer; because \{StorageSize \(<\) = 5000, RRS \(\Rightarrow\) StoragePrice = 0.05 \ast StorageSize\} are non conflicting constraints.

The second group of conflicting constraints would be discovered when the constraints were grouped by the following four variables: TransferredGb, StorageSize, ReadRequests, and WriteRequests. At this step, the following constraints would be grouped:
D.3. EXAMPLES FOR COMPLIANCE SECTION

- In the template: \((\text{TransferredGb} < \text{StorageSize} \times 100) \text{ AND } (\text{ReadRequests} > \text{WriteRequests}), \text{StorageSize} \leq 5000\).

- In the offer: \(\text{StorageSize} = 500, (\text{TransferredGb} < \text{StorageSize} \times 100) \text{ AND } (\text{ReadRequests} < \text{WriteRequests})\).

In that case, after a recursive call to \textit{areCompliant} with the extracted groups to refine the explanation, we would get as explanation \{\(\text{TransferredGb} < \text{StorageSize} \times 100\) AND \(\text{ReadRequests} > \text{WriteRequests}\)} from template and \{\(\text{TransferredGb} < \text{StorageSize} \times 100\) AND \(\text{ReadRequests} < \text{WriteRequests}\}\) from offer; because \{\text{StorageSize} = 500\} is not conflicting.
TRANSLATING FROM OTHER WSACs TO iAgree

The expressive and domain-independent sublanguages included in iAgree (see Chapter §6) make it especially suitable to become an intermediate WSAC into which other WSACs can be translated. Furthermore, we have performed successful translations from other commonly used WSACs to iAgree without losing the benefits of the domain specific languages used in those WSACs. However, no formal evaluation of the language has been developed in this sense and it will be performed in a further research.

In this appendix we include the translation of several documents specified with other WSACs as WSAG4J proposed by Wäldrich et al in [96] and SWAPS, proposed by Oldham et al in [100]. But also an ad-hoc SLA specification provided by Oriol et al in [101] is translated to iAgree.

E.1 Translating WSAG4J Documents to iAgree

The XML of Document E.1 has been taken from a release distribution of the WSAG4J version. It includes the aforementioned sublanguages: JSDL as SDT sublanguage, and XMLSchema for SP sublanguage and for creation constraints. In turn, the translated iAgree Document E.2 uses the sublanguages defined in previous Chapter §6.

```xml
<wsag:Template xmlns:wsag="http://schemas.org/ws-agreement">
  <wsag:TemplateId>1</wsag:TemplateId>
  <wsag:Name>UNICORE6-COMPUTE</wsag:Name>
  <wsag:Context>
    <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
    <wsag:TemplateName>UNICORE6-COMPUTE</wsag:TemplateName>
    <eng:WSAG4JSession>12115dfe963-419678bc139598cf</eng:WSAG4JSession>
  </wsag:Context>
  <wsag:Terms>
    <wsag:ServiceDescriptionTerm xmlns:wsag="http://schemas.org/ws-agreement">
      <wsag:Name>APPLICATION_STD_1</wsag:Name>
      <jsdl:JobDefinition xmlns:jsdl="http://www.w3.org/2001/XMLSchema">
        <jsdl:JobDescription>
          <jsdl:Application>
            <jsdl:ApplicationName>WISDOM-PACK</jsdl:ApplicationName>
            <jsdl:ApplicationVersion>1.0</jsdl:ApplicationVersion>
            <jsdl-posix:Environment name="InputFilename">/input.dat</jsdl-posix:Environment>
            <jsdl-posix:Environment name="OutputFilename">/output.dat</jsdl-posix:Environment>
          </jsdl:Application>
        </jsdl:JobDescription>
      </jsdl:JobDefinition>
    </wsag:ServiceDescriptionTerm>
  </wsag:Terms>
</wsag:Template>
```
APPENDIX E. TRANSLATING FROM OTHER WSACS TO IAGREE

iAgree (XML) document E.1: Template to test WSAG4J written in XML syntax

```
<wsag:Location>declare namespace jsdl, wsag..."APPLICATION_STD_1"/></...>
<wsag:ItemConstraint>
  <xs:sequence xs...jsdl...jsdl-posix...>
  <xs:element fixed=""WISDOM-PACK" name="ApplicationName" type="xs:string"/>
  <xs:element fixed=""1.0"" name="ApplicationVersion" type="xs:string"/>
  <xs:element name="Description" minOccurs="0" type="jsdl:Description_Type"/>
  </xs:sequence>
</wsag:ItemConstraint>
<wsag4j:ConstraintAnnotation xmlns:wsag4j="http://...">
  <wsag4j:Multiplicity>1</wsag4j:Multiplicity>
</wsag4j:ConstraintAnnotation>
<wsag:Location>declare namespace wsag..."..."</wsag:Location>
<wsag:ItemConstraint>
  <xs:sequence xs...>
  <... name="AgreementInitiator" minOccurs="0" maxOccurs="0"/>
  <... name="AgreementResponder" minOccurs="0" maxOccurs="0"/>
  <... name="ServiceProvider" minOccurs="1" maxOccurs="1"/>
  <... name="ExpirationTime" minOccurs="0" maxOccurs="0"/>
  <... name="TemplateId" minOccurs="1" maxOccurs="1"/>
  <... name="TemplateName" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</wsag:ItemConstraint>
<wsag4j:ConstraintAnnotation xmlns:wsag4j="...">
  <wsag4j:Multiplicity>1</wsag4j:Multiplicity>
</wsag4j:ConstraintAnnotation>
<wsag:Location>declare namespace wsag..."..."</wsag:Location>
<wsag:Item>
  <... name="AgreementOffer_Context"/>
</wsag:Item>
<wsag:CreationConstraints>
</wsag:Template>
```

iAgree (XML) document E.2: Template to test WSAG4J written in iAgree syntax

```
Template "Job computing request" - UNICORE6-COMPUTE
Provider Amazon as Responder;
Consumer X;
WSAG4JSession: 12115dcf963-419678bc139598cf;
Metrics [iAgree GeneralMetrics WSAG4J.metrics]
  ApplicationName: string;
  ApplicationVersion: string;
  InputFilename: string;
  OutputFilename: string;

AgreementTerms
Service UNICORE6 available at ... .
  Global description [SDTAPPLICATION_STD\_1]
    ApplicationName = "WISDOM-PACK";
    ApplicationVersion = "1.0";
    InputFilename = "/input.dat";
    OutputFilename = "/output.dat";

Creation Constraints:
C1 [JobDefinition...ApplicationName]: ApplicationName belongs {WISDOM-PACK};
C2 [JobDefinition...ApplicationVersion]: ApplicationVersion belongs {1.0};
C3 [JobDefinition...Description]: Description: string;
//because our explanatory analysis do not analyse them
```

198
E.2 TRANSLATING SWAPS DOCUMENTS TO IAGREE

The XML of Document E.3 has been taken from the Oldham et al. work [100] whose authors do not provide every WS–Agreement element by simplicity, thus elements such as the service description, or the creation constraints, are not provided. It includes the aforementioned sublanguage for predicate tags of WSLA [61]. In turn, the translated iAgree Document E.4 uses the sublanguages defined in previous Chapter §6. Especially interesting is the translating of penalty tags to iAgree syntax.
E.3 TRANSLATING SALMON AD-HOC SLAS TO IAGREE

The Document E.5 is an ad-hoc SLA document used by Oriol et al to test the monitoring framework proposed in [101]. It has a simple predicate sublanguage, but the interest of translating it to iAgree are both: (1) the local validity periods assigned to some of the SLOs, and (2) the terms scope translation. In turn, the translated iAgree Document E.6 uses the sublanguages defined in previous Chapter §6.
APPENDIX E. TRANSLATING FROM OTHER WSACS TO IAGREE

SLA of "ServiceName":
SL01: Current Response Time < 1000 ms for operation O1
SL02: Current Response Time <= 800 ms for operation O2
SL03: Average Response Time <= 1000 ms for operation O3 //in the last 24h
SL04 Average Availability >= 95 //in the last 24h
SL05: Maximum RoundTripTime <= 1200 ms //in the last 24h

iAgree (XML) document E.5: ad-hoc SALMon SLA

Agreement ServiceName-SLA
Provider as Responder;
Defined Period:
Last24: from current to a day before;
Metrics [iAgree.generalMetrics, SALMon.metrics];
  AverageAvailability: integer [0..100];
  MaximumRoundTripTime: integer;
  CurrentResponseTime: integer;
  AverageResponseTime: integer;

AgreementTerms
Service //an specific service description should be provided
Monitorable Properties
global:
  AverageAvailability;
  MaximumRoundTripTime;
for O1:
  CurrentResponseTime;
for O2:
  CurrentResponseTime;
for O3:
  AverageResponseTime;

Guarantee Terms
G1: Provider guarantees CurrentResponseTime < 1000;
on O1;
G2: Provider guarantees CurrentResponseTime < 800;
on O2;
G3: Provider guarantees AverageResponseTime <= 1000;
on Last24;
on O3;
G4: Provider guarantees AverageAvailability >= 95;
on Last24; //upon any operation
G5: Provider guarantees MaximumRoundTripTime <= 1200;
on Last24; //upon any operation

iAgree (XML) document E.6: ad-hoc SALMon SLA written in iAgree
IDEAS has been applied to ease the development of an advanced iAgree-based SLA-solution called SALMonADA (cf. the attached paper from the next page), that combines monitoring and analysis capabilities. Specifically, the compliance checking operations of ADA have been reused to develop two new analysis operations to check and explain the unfulfillment of SLAs. Such new analysis capabilities added to ADA are depicted in Figure §F.1 within the monitoring component package. In such a Figure, the ADA components used from the monitoring components have been filled with a dark colour.

Figure F.1: SALMonADA components diagram reusing ADA Core components.
Comprehensive Explanation of SLA Violations at Runtime

C. Müller(1), M. Oriol (2), X. Franch (2), J. Marco (2), M. Resinas (1), A. Ruiz–Cortés (1), M. Rodríguez (2)

(1) University of Seville, LSI, Seville (Spain), ISA research group, http://www.isa.us.es/
{cmuller, resinas, aruiz}@us.es
(2) Universitat Politècnica de Catalunya, Barcelona (Spain), GESSI research group, http://www.essi.upc.edu/~gessi/
{moriol, jmarco}@lsi.upc.edu, {franch, marcr}@essi.upc.edu

Abstract—Service Level Agreements (SLAs) establish the Quality of Service (QoS) agreed between service-based systems consumers and providers. Since the violation of such SLAs may involve penalties, quality assurance techniques have been developed to supervise the SLAs fulfillment at runtime. However, existing proposals present some drawbacks: (1) the SLAs they support are not expressive enough to model real-world scenarios, (2) they couple the monitoring configuration to a given SLA specification, (3) the explanations of the violations are difficult to understand and even potentially inaccurate, (4) some proposals either do not provide an architecture, or present low cohesion within their elements. In this paper, we propose a comprehensive solution, from a conceptual reference model to its design and implementation, that overcomes these drawbacks. The resulting platform, SALMonADA, receives the SLA agreed between the parties as input and reports the explanations of SLA violations in a timely and highly understandable way. SALMonADA performs an automated monitoring configuration and it analyses highly expressive SLAs by means of a constraint satisfaction problems based technique. We have evaluated the impact of SALMonADA over the resulting service consumption time performance. The results are satisfactory enough to consider SALMonADA for SLA supervision because of its low intrusiveness.

Keywords—service level agreement; SLA; monitoring; analysis; violation detection and explanation; QoS;

I. INTRODUCTION AND MOTIVATION

Service Level Agreements (SLAs) establish the Quality of Service (QoS) agreed between service-based systems consumers and providers, as well as penalty/reward clauses to be applied when the SLA is violated. Detecting such violations may be complicated, consider for instance the following case.

The Amazon Simple Storage Service (AmazonS3) guarantees a monthly uptime percentage equal to or greater than 99.9% in its SLA\(^1\) including a clause to reward their consumers against a lack of service. However, Amazon requires the customer to proof this violation by sending an email within ten business days after the end of the billing cycle in which the errors occurred. Proving the violation demands the customer to compute the monthly uptime percentage by subtracting from 100% the average of the error rates\(^2\).

This Amazon scenario illustrates the need of having techniques to supervise the fulfillment of SLAs [1]. In response to this need, several quality assurance proposals have been formulated. They deal with aspects such as: violation detection, either at runtime [2]–[8] or testing time [9], [10]; violation notification either by push (i.e. that are notifications to subscribed clients that receive the violation reports as soon as they are obtained at runtime) [4], [5], [11] or pull strategies [2], [8], [12]; and violation explanation [13], [14]. The possible adoption of these proposals in real-world scenarios is greatly influenced by the following factors:

1) Which SLAs are supported. Usually, real-world SLAs (e.g. AmazonS3 SLA) describe the parties obligations in natural language and they may comprise complex elements such as: (1) conditional terms subject to a precondition (i.e. if the precondition holds, then the term applies), for instance, AmazonS3 SLA offers a data durability of 99.99% only if the client choose a cheaper reduced redundancy storage; (2) optional or alternative terms are often included (i.e. a set of terms that can be chosen by the customers), for instance, AmazonS3 offers a set of alternative support plan terms.

2) How the violations are detected. In order to detect violations we need to monitor the QoS offered by the providers at runtime (i.e. specific values for monitorable service properties such as the service availability or response time). Such QoS monitoring results are obtained by monitoring techniques [15] that must be previously configured with the location of the service to monitor, and its monitorable service properties. Thus, the SLA violations are detected by checking the agreed QoS against the obtained QoS monitoring result. For instance, in the case of AmazonS3, the SLA determines the need of monitoring and obtaining a specific value for the monthly uptime percentage for each billing cycle in order to detect a possible violation.

3) How the violations are explained. Ideally, the explanation needs to be both comprehensive and timely. For comprehensive, we mean to provide a user-friendly and accurate violation report including not only the violated terms, but also the violation causes and even possible effects, in a form that is easy to understand by humans. For instance, in the AmazonS3 scenario, a comprehensive explanation would require a report including that the violated term is "monthly uptime percentage equal to or
greater than 99.9%” by a monthly uptime percentage of 92% measured in a specific monthly billing cycle. For timely, we mean to communicate the violation as soon as it happens, i.e. when the QoS monitoring result has just been retrieved at runtime and it is checked against the SLA to detect and explain the violations.

4) Which architecture is proposed. In order to ease the maintainability and adaptability of the system, the architecture should comprise loosely coupled and highly cohesive elements separating monitoring from analysis responsibilities.

As shown in Sec. II, current proposals do not deal satisfactorily with the characteristics above. Some of them may be completely tight to a particular SLA notation that are not able to deal with all the aspects required in real-world scenarios, others couple the monitoring configuration to a given SLA specification, and most (if not all) provide very basic information when a violation occurs. Furthermore, not all proposals provide architectures with a clear separation of concerns between monitoring and analysis.

The goal of this work is to design and implement SALMonADA, a service-based system to monitor and analyse SLAs in order to provide timely and comprehensive explanations of violations. SALMonADA main features are:

1) specification of a wide range of SLA structures and complex elements based on the use of WS–Agreement [16].
2) automated monitoring configuration through the analysis of the SLAs but without coupling to a given SLA specification.
3) powerful detection and explanation of the SLAs violations by means of a Constraint Satisfaction Problem (CSP)-based technique. A comprehensive and timely explanation of the SLA violations is notified either by push or pull strategies.
4) low coupling in its service-oriented architecture that supports the independent evolution of the core monitoring and analysis components, or even their independent substitution when moving from one problem domain to another.

The impact of SALMonADA over the resulting service consumption time performance and its scalability have been evaluated over real services. We analysed the results in alternative deployment scenarios and we state that SALMonADA has a low intrusiveness compared to the benefits it provides. Moreover, we have developed a web client as a front-end to try it online.

The paper is organised as follows. The related work is evaluated in Sec. II. The conceptual reference model of SALMonADA is detailed in Sec. III, while the details of its design and development are included in Sec. IV. Section V provides information about the CSP-based technique to analyse the SLA fulfillment. Section VI provides information about how SALMonADA checks the QoS monitoring result against the SLA to provide timely and comprehensive explanations. In turn, Section VII evaluates the impact of SALMonADA over the resulting service consumption time performance. Finally, Sec. VIII concludes the paper with a discussion of our contributions.

II. RELATED WORK

Several service-based systems quality assurance proposals that aggregate monitoring and analysis facilities can be found in the literature. To conduct the search of the related work, we have revised the most relevant conferences and journals in the area, selecting those papers that were scoped in the field of SLA monitoring and analysis. Furthermore, we have increased the results by adding relevant papers obtained from experts in the field. Table I summarizes the results of this state of the art.

We have examined the selected papers under the four factors described in the introduction. The first three factors falls into the functionality of the proposed solution, whereas the fourth factor falls into its architecture.

**Functionality.** Considering the three factors for functionality identified in the introduction, we focus on the following issues: (1) Which SLAs are supported, (2a) How the information to configure the monitor is specified, (2b) How the QoS monitoring result is specified and (3) How the violations are explained.

**Architecture.** The issues arising from this factor are: (1) Which architectural elements are needed and (2) How the architectural elements are structured.

We analyse these issues below:

**Which SLAs are supported.** The proposals fall into one of the following categories:

<table>
<thead>
<tr>
<th>SLAs</th>
<th>Monitoring config.</th>
<th>Monitoring results</th>
<th>Explanation of violations</th>
<th>Architecture elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supported</th>
<th>N/A</th>
<th>N/A</th>
<th>N/A</th>
<th>N/A</th>
</tr>
</thead>
</table>

Table I: Analysis of the Related Work
Particular SLA notation [4], [5], [10], [17]. The supported SLAs include ad-hoc information without considering a general-purpose structure or notation. In addition the proposals are not able to deal with all the aspects required on some real scenarios. For instance, they do not support conditional terms subject to preconditions, expressive Service Level Objectives (SLOs), and optional or alternative terms to specify agreement variants [18].

General-purpose SLA notation [2], [3], [9], [11]–[14], [19], [20]. The supported SLAs consider a general-purpose structure and/or notation. Specifically, in works such as [9], [13], [14], [19] the SLAs support the general-purpose structure proposed in the WS–Agreement specification [16], a highly flexible and widespread SLA notation. An advantage of WS–Agreement over the particular SLA proposals is that it supports the aforementioned aspects that are necessary to model agreements of real scenarios, namely: (1) optional or alternative terms to specify agreement variants, and (2) expressive SLOs that can be guarded by a qualifying condition (QC) to specify conditional terms. However, WS–Agreement just provides a general-purpose schema that must be extended up to eight different points with an internal sublanguage. More specifically, two of the eight points need to be necessarily extended to become a fully-featured language, namely the service description terms (SDT) that defines the service functionality, and the SLOs. This causes that a system that can deal with a particular WS–Agreement notation is not able to deal with other WS–Agreement notations. Moreover, the sublanguages used in [9], [13], [14], [19] are not neither general-purpose to be easily mapped to each other, nor human understandable. Other works support SLAs specified with WSLA [2], [3] and SLA* [11] that include general-purpose structures and notations. However, such notations are based on XML schemas even for the SLOs assertions (being recursive in the case of SLA*) and thus, they are not as human understandable as they should be.

How the information to configure the monitor is specified. We find the following situations:

- Automatic, coupled to the SLA. Some approaches include a mechanism to automatically configure a monitor from the SLA [2], [3], [5], [9], [10], [12]–[14], [17]. However, in these solutions, the monitor can only be used for a concrete SLA specification.
- Automatic, decoupled from the SLA. There are some works [4], [11], [19], [20] which decouple the SLA from the monitor by translating automatically the SLA to another document which includes the information required for monitoring.

How the QoS monitoring result is specified. Some approaches do not describe how the monitoring results are reported [9], [10], [13], [14]. From those that describe it, some provide a log file [20] or an Application Programming Interface (API) [2]–[4], [11], [12], [19] to access the monitoring results. Since these logs and APIs are not standardized, they present significant differences to each other. The lack of a standard prevents the possibility to change easily the monitor and also it requires the analyser to be compatible with the monitor’s API or log. A more effective solution is found in [5] where the authors propose a query language to access the measurements of the monitor. However, in this solution, the monitor must deal with that query language. A more advanced solution is also found at [17], where the authors propose a model to store the results, however such model is not explicitly described.

How the violations are explained. The proposals fall into one of the following categories:

- Just detection. Some works detect SLA violations without explaining the cause [4], [5], [9], [10], [19], [20].
- Partial explanation. Other approaches provide the SLOs that have been violated [2], [3], [11], [12], [17], leading to a partial explanation of the violation as an SLO could be violated for several reasons that are not detailed.
- Precise explanation but not human-understandable. In [13], [14] the authors provide an accurate violation explanation that identifies the violated SLOs and the monitoring results which caused the violation. However, they use Event-Calculus to express both the conditions and monitoring results, which as the authors state, is not user-friendly.

Which architectural elements are needed. We find the following situations:

- The proposal does not include an architecture [9], [10].
- The proposal includes an architecture where monitoring and analysis are performed in the same component [20].
- The proposal implements the QoS monitoring and SLA analysis in two separated components with different responsibilities, increasing the cohesion and reusability [2]–[5], [11]–[14], [17], [19].

How the architectural elements are structured. We classify the approaches into:

- Component-Based Systems (CBS) [2], [3], [13], [14], [17], [20]. The components are specific of the system, and no details regarding deployment or protocols used are described, which makes them difficult to reuse or replace.
- Service-Oriented Architecture (SOA) [4], [5], [11], [12], [19]. This architecture adds the capability of deploying the different constituent services in a distributed manner, adapting or replacing them, in a highly cohesive and loosely coupled system.

Other projects and networks on services, such as [1], [21], are not included in this state of the art because the presented results are general models without a specific solution. As a summary of this state of the art (see Table 1), we can observe that in the functional part, the proposals cover satisfactorily just one or even none of the four identified issues. Improving this situation is the aim of our work. Our solution takes all the features mentioned and either: (1) cover the issues to the same degree of the best proposal of the state of the art or improves the existing proposals by introducing a new strategy, as it will be described in the following sections. This improvement on functionality is accompanied by optimal architectonic decisions (highly decoupled SOA solution).
Our solution includes the usage of platforms from our previous works, SALMon [22] and ADA [23]. The main goals and initial features they had before this approach are as follows: ADA is a platform which main goal has been to check the consistency of an SLA and compatibility between SLA offers and SLA demands. ADA supports general-purpose WS-Agreement but lacks of any monitoring feature, and therefore, also lacks of detection and explanation of violations. In turn, SALMon is a monitoring platform which main goal has been to gather the QoS of web services and check simple conditions for several activities. However, the monitoring configuration was set through an API (not automatic) as well as its monitoring results. Moreover it does not support WS-Agreement and lacked of a comprehensive and timely explanation of violations. A detailed description of these platforms, including their enhancements in this approach are described in Sec. IV-D for ADA and Sec. IV-C for SALMon.

III. THE SALMONADA CONCEPTUAL REFERENCE MODEL

This section presents the conceptual reference model of the SALMonADA platform. This model introduces the relevant human and software agents that participate in the platform, and the data that they need to interchange. Its purpose is to provide a high-level view of the platform before going into the architectural and technological details. The conceptual reference model is shown in Fig. 1, using the SAP-TAM notation [24]. It includes the following agents:

**Client:** is the user of the platform. The client goal is to retrieve the explanations of SLA violations and/or the monitoring results. To fulfill such a goal the client has the responsibility of providing the SLA to monitor. The role of client may be played by the service consumer, the service provider or even a third party interested in monitoring the assessment of the SLA.

**Configurator:** is the agent that configures the monitor with the information included in the SLA. Thus, it decouples the SLA (a contractual specification understood by SLA-dependent agents) from the Monitor, by generating from the SLA a Monitoring Management Document (MMD), which is a specification of the monitoring directives to configure a monitor (see Sect. IV-B for more details about its structure).

**Monitor:** is the agent responsible of monitoring the interaction between the provider and the consumer according to the monitoring directives given in the MMD. The Monitor obtains the measured metrics from such an interaction and updates the monitoring results in the MMD right after each consumer request.

**Analysar:** is the agent that checks if the monitoring results of a service, available in the updated MMD, is compliant or not with the agreed QoS included in the SLA. It ultimately produces the explanations of violations structured in a document designed for this purpose, the Service Level Fulfillment (SLF).

As a summary, the conceptual reference model shows a clear separation of concerns on the management of the SLAs, the MMDs and the SLFs, independent of the concrete technologies used, that are described in the next section.

IV. THE SALMONADA PLATFORM

In this section we present the details of SALMonADA platform which can be viewed as an instantiation of the conceptual reference model presented above. The platform is able to monitor and analyse expressive SLAs specified with WS–Agreement. SALMonADA has a decoupled architecture that integrates into a service-based system two previously existing systems which in turn, have been extended to realize this project: the SALMon monitor [22] and the ADA analyser [23]. Such an architecture is shown in Fig. 2. The core component of SALMonADA is its composer that provides the external interface and controls the execution flow of the system keeping SALMon and ADA decoupled from each other. Moreover, SALMonADA also comprises the MMD Manager service, which is used to generate and manipulate the MMDs independently of the underlying structure of such documents.

In the following subsections, we provide more details about these SALMonADA components. We focus on the internal architectures and responsibilities.

#### A. The SALMonADA composer

The SALMonADA composer is the component that presents the external interface to the client and controls the execution flow of the system. It also orchestrates the composition of ADA and SALMon to: (1) extract the monitoring information from an SLA to be included in an MMD, (2) monitor SLAs, and (3) analyse SLAs. The composer follows the low coupling design principle, so that it is possible to replace the monitor, the analyser, or the MMD manager without affecting the other elements of the platform. This implies that also the formats of the documents (SLA and MMD) can be changed with minimal impact.
Thus, the SALMonADA composer controls the information flow needed for the different constituent services such as: (1) which client is interested in knowing which SLA violations; (2) the MMDs obtained from the SLAs through the MMD Manager; (3) the MMD with the monitoring results obtained by the monitor service; and (4) the pair SLA-MMD with the monitoring results that is required by the analyser service. Moreover, the composer provides the interface to supply to the clients both, the SLF information, and the MMD with the monitoring results to be aware of the QoS evolution at runtime.

The architectural model of the SALMonADA composer is depicted in Fig. 3 and comprises the following elements:

**Controller**: it provides the external interface to the client and controls the execution flow of the system by communicating with the SALMon and ADA services. It is also responsible to interact with the other components of the composer.

**Client Manager**: is responsible for storing and retrieving the different clients of SALMonADA from the client repository. Each client has associated one or more WS–Agreement documents to monitor.

**Publisher**: This component implements the Reference Data Observer SOA pattern [25]. In this case, it notifies the SLF to any subscribed service when a violation occurs.

### B. The MMD Manager service

The MMD Manager is one of the components that delimits the kind of SLA supported by SALMonADA because it generates the MMD from a given SLA. SALMonADA aims to support a general-purpose SLA specification in order to report the SLF of as many SLAs as possible. Although any other MMD Manager service can be developed supporting any SLA specification, we choose to support the WS–Agreement specification due to its flexibility. As stated in Sec. II, the WS–Agreement schema must be extended at least with two sublanguages. In our proposal we use general-purpose sublanguages [18] that make it easier to map from other sublanguages and SLA notations to our supported SLA. Specifically, we support specifications of: (1) the service by means of a simple attribute-value pairs within SDTs, and (2) the SLOs by means of an assertion language that is included in Sec. IV-D. Examples of both sublanguages are included in the SLA of Fig. 4 that comprises several SLOs relating three service properties of the ADA analysis service: the average service availability (AverageAvailability); the average response time of all service operations (GeneralResponseTime); and the average between several response time measures of a given service operation (AverageResponseTime). In this sense, note that the two latter SLOs of the SLA guarantee a certain response time depending on the scoped operations. In addition, such terms are conditional terms because they are guarded by a qualifying condition. That is, the SLOs are only evaluated if the qualifying condition are held. In this case, the QCs are held when the GeneralResponseTime is more than 0.5 seconds.

The MMD Manager receives the WS–Agreement document and generates the corresponding MMD. Fig. 5 provides an extract of the MMD obtained from the WS–Agreement document of Fig. 4 and it depicts our selected MMD structure as well as which SLA information is included in each and every of its XML elements. Thus, the MMD structure is divided in four XML elements: (1) the web service information element which includes the information needed to invoke the service, that is extracted from the service description terms of the SLA; (2) the monitor configuration element which comprises the monitoring time interval, which starts monitoring from
the generation of the MMD until the SLA expiration time; (3) the service metrics which are the service properties used within the SLOs whose scope is the whole service (i.e. to all service operation in the SLA); and (4) the operation metrics which are the service properties used within SLOs scoping to specific service operations but not to all of them. For instance, AverageResponseTime is an operation metric for the explainNonCompliance service, specifically added for SALMonADA, (1) stores the MMDs in the repository, (2) configures the Monitor service as required in the MMD and (3) updates the MMD when new measured metrics are retrieved. A measured metric is a metric with its value in a specific timestamp (see Fig. 7). The SALMon Manager uses the MMD Manager whenever it requires to get or store the monitoring results to the MMD. In such a way, it can be easily extended to support different MMD structures.

**SALMon Manager:** this service, specifically added for SALMonADA, (1) stores the MMDs in the repository, (2) configures the Monitor service as required in the MMD and (3) updates the MMD when new measured metrics are retrieved. A measured metric is a metric with its value in a specific timestamp (see Fig. 7). The SALMon Manager uses the MMD Manager whenever it requires to get or store the monitoring results to the MMD. In such a way, it can be easily extended to support different MMD structures.

**Monitor:** manages and activates the Measure Instruments that will gather the measured metrics(see below). The Measure Instruments are activated based on the metrics specified in the service’s MMD. When activated, the monitor gets the measured metrics through the QoS Repository.

**Measure Instruments:** are the components that include the logic to gather the measured metrics such as response time and availability (metrics belonging to the Software as a Service layer). Each Measure Instrument is responsible to compute a specific metric applied to a particular service and store the results in the QoS Repository.

**ESB:** Instead of invoking the services directly, all requests and responses are sent through the ESB, which in turn, feeds the Measure Instruments with the intercepted messages.

**QoS Repository:** stores the measured metrics. To reduce storage consumption, it stores only the last measurements, the average values and the number of invocations.

**Publisher:** implements the Reference Data Observer SOA pattern [25]. In this case, it notifies the measured metrics to the SALMonADA composer when a measurement is obtained.

Using SALMon, the consumer of the service invokes the services through the provided ESB. The ESB forwards the request to the target service and notifies it to the activated measure instruments associated to the service or operation.

---

**Figure 5.** Extract of the MMD obtained from document of Fig. 4.

**Figure 6.** Architectural Model of the extended SALMon
When new measured metrics are obtained, the SALMon manager is notified and it generates the new MMD with the monitoring results as Fig. 7 depicts. Notice that the measured metrics related to time are defined in milliseconds, as it is a convenient unit to express the monitoring results.

Finally, it is important to address performance and privacy issues during monitoring. The former is evaluated in depth in the experimental results in Sec. VII. The latter, which is strongly affected by the service policies and the deployment configuration, is discussed here considering the different factors involved. On secure services, the ESB does not interfere with the security level in the communications. However the issue arises for non-secure services dealing with sensitive data. For such a reason, the body of the message is never stored. Moreover, the ESB can be deployed on either the client or provider side under their management, so any sensitive data can be encrypted using WS-Security before forwarding the message to the Measure Instruments.

D. The ADA service

ADA is an Agreement Document Analysis framework aimed at extracting useful information from agreement documents at any SLA life-cycle stage [23]. It has been developed based on our previous theoretical works on applying the constraint satisfaction problem (CSP) [28] paradigm to the automated procurement of web services [29]. The main features of ADA [30] are: (1) ready-to-use by detecting and explaining conflicts within and between WS–Agreement documents [30], [31]; (2) functional suitability by supporting the analysis of expressive WS–Agreement documents with conditional, optional, or alternative terms, term scopes, arithmetic-logic expressions inside SLOs, etc [18]; (3) understandability by supporting a plain-text notation [18] that makes reading and writing WS–Agreement documents easier for humans; (4) interoperability through a triple distribution model (Java library, OSGi service, and web service); and (5) CSP solver independence.

Similarly to SALMon, ADA has been enhanced with new components (see elements with a thicker frame in Fig. 8): the ADA Manager, several analysis facilities to detect and explain violations at monitoring, and repositories for SLAs and SLFs. The existing ADA components have also been adapted to interact with the new components. We describe below each ADA component.

ADA Manager: is responsible for SLA storage and retrieval from the repository; as well as the translation between several SLA models to a WS–Agreement-based normalised one using XML that ADA is able to analyse.

ADA Analyser: is responsible for: (1) analysing the fulfilment of the WS–Agreement document given the monitoring results stored in the corresponding MMD; (2) the creation of violation explanations when a violation is detected; and (3) the storage and retrieval of the SLF information. Such analysis is performed by means of a CSP solver tool (see Sec. V for more details) that supports the following assertion language:

\[ P ::= P \ op \ P \ | \ T, \ \text{predicate, where } op \in \{ \land, \lor, \neg \} \]

\[ T ::= E \ op \ C, \ \text{term, where } op \in \{ =, \neq, >, \geq, <, \leq \} \]

\[ E ::= E \ op \ E \ | \ \text{lit}, \ \text{expression, where } op \in \text{an algebraic operator defined on the domain of variables and literals} \]

As a result, we work with an assertion language inside WS–Agreement documents that is not just expressive but also easy to understand by non-technical users (cf. SLOs of Fig. 4). Moreover, as Fig. 9 shows, the reported SLF is also understandable because the violated terms are associated with the violating monitored values, both expressed in a human understandable way.

V. A CSP-BASED TECHNIQUE TO EXPLAIN VIOLATIONS

The detection and explanation of violations, which is the ultimate goal of this paper, is necessarily based on the alignment of the WS–Agreement document, which expresses the requirements on the service, and the MMD document, which
collects monitoring information at runtime. To implement this alignment, we need to provide semantics to both documents and then define the concept of violation and the procedure to get explanations. In our technique, the semantics is defined through semantic mappings. Under this view, the elements of the documents, that are considered as source models, are mapped into a target domain whose semantics have been formally defined [32]. The main advantage of semantic mappings is that they enable the usage of techniques, preferably automated, which are specific to the target domain in order to infer properties in the source models [33].

In our case, we have chosen constraint satisfaction problems (CSP) [28] as the target domain. Solving problems by means of CSPs has been a research topic in Artificial Intelligence for years. In short, a CSP is a triple–tuple of the form \((V, D, C)\) where \(V \neq \emptyset\) is a finite sequence of variables that cannot appear multiple times at different positions in the sequence, \(D \neq \emptyset\) is a finite sequence of variables of domains (assuming that the order in sequences is an implicit mapping between variables and domains) and \(C\) is a finite set of constraints defined on \(V\). A solution of such CSP is a valid assignment to all variables in \(V\) in their domains \(D\) that satisfies \(C\). Consider, for instance, the CSP: \(\{(a, b), \{[0, 2], [0, 2]\}, \{a + b < 4\}\}\). Then, \((a = 2, b = 0)\) is a possible solution since it verifies that \(2 + 0 < 4\), whilst \((a = 2, b = 2)\) is not a valid assignment.

The reason for selecting CSP is twofold. On the one hand, the MMD and the most significant part of an WS–Agreement document are sets of constraints over service properties and, therefore, CSPs can be used to model the detection and explanation problem in a very natural way, as we showed in previous works [29]–[31], [34]–[36]. On the other hand, there is a plethora of CSP solvers available that support a wide range of constraints and can be used to automatically analyse WS–Agreement documents in an efficient manner.

Our proposal to interpret the SLA violations using CSPs is based on the study of the set of solutions obtained from two CSPs. The first one is created from the WS–Agreement document, by mapping the service properties into CSP variables, and the guarantee terms into CSP constraints. The second CSP is created from the MMD by mapping monitored metrics into CSP variables and the metrics with the monitored values into CSP constraints. Then, the SLA is fulfilled if all solutions of the CSP mapped from the MMD are included in the solutions of the CSP mapped from the SLA, and the SLA is violated otherwise.

The proposed CSP-based technique to detect and explain SLA unfulfillments takes into consideration that those WS–Agreement terms whose scope is a specific service operation are only affected by monitored metrics measured while such operation is being executed. For example, the AverageResponseTime attribute of a term whose scope is the explainNonCompliance operation, would be checked for fulfillment with the average of monitored response time of explainNonCompliance operation. To consider this in our system, the original WS–Agreement document \(\Delta\) is separated into several views \(\Delta_{op}\) by the service operation scope. Thus, each view is a WS–Agreement document by itself restricted to both: the set of terms whose scope is a specific service operation; and such terms whose scope are all service operations (i.e. such terms that do not specified any scope such as GT generalAvailability of Fig. 4). In the case a term whose scope includes more than one service operation, but not all of them, it would be included in as many views as scoped service operations. For instance, the term GT generalResponseTimeRelations of Fig. 4 would be included in four views, one for each scoped service operation.

In general, a WS–Agreement view \(\Delta_{op}\) defines some guarantees for a set of service properties whose domain has been previously defined. WS–Agreement views can be modelled in a CSP by means of the semantic mapping \(\text{map}(\Delta_{op})\) that is summarised in Table II, whereas Fig. 10 shows the CSP mapped from the explainNonCompliance view \((\Delta_{explainNonCompliance})\) of the ADA SLA included in Fig. 4.

Concerning MMDs, the original MMD \(M\) is also separated into several views \(M_{op}\) by the monitored service operation. Thus, each view comprises the QoS monitoring result that includes metric–measure pairs, of a unique service operation\(^5\). Table III summarises the MMD semantic mapping \(\text{map}(M_{op})\), whereas Fig. 11 shows the CSP mapped from the monitoring results of the ADA SLA corresponding to the explainNonCompliance operation included in Fig. 7.

In order to carry out the detection and explanation of SLA unfulfillment, we need to use a pair of analysis techniques that have been widely used in CSPs:

- **solve\((V, D, C)\)** tries to find all CSP solutions. To this end, many heuristics and techniques have been developed to solve CSPs in an efficient manner [38], [39].

\(^5\) Note that the measure units must match for a same scale (e.g. time) in the service properties of the SLA and the MMD. If they did not match, a conversion formula would be applied for each CSP constraint, as proposed in some QoS ontologies for Web Services [37]. Fig. 15 includes different measure units for the AverageResponseTime property: seconds in the SLA and milliseconds in the MMD.
V = \{ \text{AverageResponseTime, GeneralResponseTime, AverageAvailability} \}
D = \{ [1..\infty], [1..\infty], [1..100] \}
C = \{ \text{AverageAvailability} \geq 95, \text{GeneralResponseTime} \leq 1.5, \text{GeneralResponseTime} > 0.5 \implies \text{AverageResponseTime} > 2 \}

Figure 10. CSP generated from WS–Agreement view: \( \Delta \)explainNonCompliance-

V = \{ \text{AverageResponseTime, GeneralResponseTime, AverageAvailability} \}
D = \{ [1..\infty], [1..\infty], [1..100] \}
C = \{ \text{AverageAvailability} \leq 100, \text{GeneralResponseTime} = 1.7, \text{AverageResponseTime} = 3.421 \}

Figure 11. CSP generated from MMD view: \( M \)explainNonCompliance-

- \( \text{explain}(V, D, C) \) tries to provide an explanation when such solution is not possible. This explanation is a minimal set of constraints \( c \in C \) that makes impossible to find a solution that satisfies \( c \). For instance, the CSP \( \{[a,b,d], [0.2,0.2,0.2], \{a+b < 1, a > 0, d > 1\}\} \) is not satisfiable, and its possible explanations \( c \) are either \( \{a+b < 1\} \) or \( \{a > 0\} \).

On the basis of these operations and the previously described map functions for the SLA and MMD, we may trace back in order to infer the SLA fulfillment with the MMD measures, providing the precise semantics for detecting the unfulfillment and their explanation, as follows.

\[
\text{unfulfillment}(\Delta_{op}, M_{op}) \implies \text{solve}(\text{map}(M_{op}) \Rightarrow \text{map}(\Delta_{op})) = \emptyset
\]

where the map function returns a CSP for each view as Figures 10 and 11 show; and the expression \( \text{solve}(\text{map}(M_{op}) \Rightarrow \text{map}(\Delta_{op})) = \emptyset \) corresponds to \( \text{solve}(\text{map}(M_{op}) \land \lnot\text{map}(\Delta_{op})) \neq \emptyset \) by the Marriott and Stickney [40] implication and equivalence definition. Thus, in the example above, the WS–Agreement view \( \Delta \text{explainNonCompliance} \) is not fulfilled by the MMD measures \( M \text{explainNonCompliance} \), due to the existing solutions for the CSP generated by the implication operation: \( \text{solve}(\text{map}(M \text{explainNonCompliance}) \land \lnot\text{map}(\Delta \text{explainNonCompliance})) \neq \emptyset \).

In this case, an explanation like such provided in Fig. 9 would be reported using the following formula:

\[
\text{unfulfillment}_{exp}(\Delta_{op}, M_{op}) = \text{quickxplain}(\text{map}(M_{op}) \Rightarrow \text{map}(\Delta_{op}))
\]

where \text{quickxplain} provides an explanation based on the algorithm presented in [41] that implements the aforementioned \text{explain} operation to obtain a minimal subset of unsatisfiable CSP constraints of a given CSP. Note that the key of our approach to apply such an algorithm is to get one CSP through the implication between the CSPs mapped from both, the WS–Agreement and the MMD documents. The algorithm is applied subsequently to the different views of both documents and trace the CSP constraints back to the corresponding WS–Agreement term or MMD measure. An example of the final result is provided in Fig. 9 and in the following Sec. VI.

VI. SALMONADA IN USE

SALMonADA is designed and developed to support push and pull interaction styles with its clients. Thus, a client, based on its own needs, may choose the preferred approach. Independently of the selected approach, a client shall start the use of SALMonADA by subscribing as such a client. Similarly, the client shall stop the SALMonADA monitoring at the end of the interaction, to be unsubscribed as client.

As the sequence diagram of Fig. 12 shows, the client starts the monitoring process by providing a WS–Agreement document to monitor its fulfillment. In case of push approach, the client shall also provide the endpoint where the notification is awaited. Independently of the approach used, \text{ADA} stores the WS–Agreement document, which is also used by the \text{MMD Manager} to generate the MMD. Such MMD will store the information to configure the monitor and all the measured metrics obtained while monitoring the service agreed in the WS–Agreement document. In doing so, a monitoring session is started by \text{SALMon} providing where the monitoring result must be notified to update the MMD after a service consumption. Finally, a client identifier (clientID) is generated to denote the specific monitoring session for this SLA. If several \text{SALMonADA} clients wanted to monitor a unique SLA, for instance the service consumer and provider to be informed about the SLA fulfillment, a different monitoring session would be started and thus, a different clientID would be returned.

When a client wants to be unsubscribed from \text{SALMonADA}, it provides its clientID to stop its monitoring process. The WS–Agreement document and the monitoring session is removed from \text{SALMonADA} by \text{ADA} and \text{SALMon} if there is no other client monitoring the same SLA, otherwise \text{SALMonADA} will keep them while in use by these other clients. In any case, the clientID is removed from the system.

A. Push Interaction Approach

The push approach is the most convenient way to interact with \text{SALMonADA} due to the push nature of its service monitoring and analysing. In this sense, as Fig. 13 shows, the platform incorporates an early analysis notification that supports the analysis of the SLA fulfillment as soon as new measured metrics have been updated in the MMD. Thus, the
SLF notification is sent to the client without any further delay than the analysis time, in contrast with windowed proposals [13] that get periodically the monitored values in a prefixed time interval to analyse them later. Assuming a similar analysis time, the difference between both approaches is higher when the violation affects to a measured metric at the beginning of the prefixed time interval (see MMD1 in Fig. 13). In the case that the violation affects to a measured metric at the end of the prefixed time interval, the notification will be delivered at the same time in both approaches. As the sequence diagram of Fig. 14 depicts, once the client has started to monitor, the provider service included in the reported WS–Agreement document is monitored by the SALMon component. Next, the MMD created from the monitored WS–Agreement document is sent to the MMD Manager with the measured metrics to be updated. Finally, the new MMD is notified to the SALMonADA composer that sends it to the ADA component to analyse the service level fulfillment of the corresponding WS–Agreement document (cf. Sec. IV-D for more details). Then, the client is notified about the WS–Agreement document fulfillment by means of the SLF. If the WS–Agreement is not fulfilled, both the specific violated WS–Agreement terms and the violating metrics, are included as violation explanation. Note that SALMonADA supports the same endpoint acting as different clients, for instance, one of them to get the SLF, and another to store reputation analytics of the service consumer and provider, or even to perform self-adaptation strategies.

For instance, to monitor the WS–Agreement document of ADA (see Fig. 4), the SALMonADA client invokes the startMonitoring method specifying such a document as input. As a result, it receives the client identifier. The WS–Agreement of ADA includes SLOs involving more than just one quality metric. Hence, an appropriate explanation identifying not only the guarantee term that are involved in the violation of the SLA, but also the concrete violating metrics, is required. For instance, some operations have a higher priority and are required to be faster than the average response time of the different methods of the service (AverageResponseTime <= GeneralResponseTime). In this case, SALMonADA would report an explanation just when the violation is detected and analysed, identifying if the violating metric is either AverageResponseTime or GeneralResponseTime because a simple identification of the violated term is not enough to grasp the violation cause. Similarly, SALMonADA supports the explanation of violations of more expressive SLOs. For instance, the provider may guarantee a different average response time limit for the slower service operations, depending on the general response time of the service, as follows: ((GeneralResponseTime >= 0 AND GeneralResponseTime < 2) IMPLIES (AverageResponseTime < 3) AND ((GeneralResponseTime >= 2 AND GeneralResponseTime <= 4) IMPLIES (AverageResponseTime < 5)).

B. Pull Interaction Approach

The pull approach allows the client to actively request the results of SALMonADA for either: the current MMD with the most recent monitoring results obtained by SALMon; or the current SLF of the WS–Agreement document analysed by ADA. The former document is obtained by invoking the getMMD method, whereas the latter is through the getServiceLevelFulfillment method. In both cases, the client is required to provide the clientID as input.

As usual in pull approaches, if the consumer(s) do not invoke the service, there is not new monitoring information and thus, it is possible that the client gets the same monitoring information in consecutive MMD requests.

For demonstration purposes, we have implemented a web application\(^6\) as a SALMonADA client in order to specify or

\(^6\)The SALMonADA web application can be tried at www.isa.us.es/ada/source/SLAnalyzer/. A screencast is available at gessilsi.upc.edu/salmon/ada/
upload the WS–Agreement documents to monitor, execute SALMonADA and receive the results. In this web application, we have introduced the WS–Agreements of ADA and SALMon themselves. By monitoring the SLAs of these services, we assess on the one hand, the functionality of SALMonADA, and on the other, the non-functional aspects of its main components. Such a SALMonADA client uses the pull interaction because the user press the corresponding interface controls to get the MMD and the analysis results. Moreover, as part of the demonstration and to assure that the service subject of the SLA is being requested, we have simulated the consumers that execute ADA and SALMon services.

With the client identifier, the SALMonADA client can get, at any time, the MMD, check if a violation has been produced, and in such a case, receive an explanation of the violation.

As Fig. 15 depicts, the web application highlights as violation explanation that the AverageResponseTime of explainNonCompliance operation is the violating metric because it was measured as 3.427 milliseconds, while the guarantee term obligates the provider to respond in less than 2 seconds. As stated in Sec. V, SALMonADA handles different measurement units and the required unit transformation is performed to evaluate the conditions.

VII. PERFORMANCE AND SCALABILITY EVALUATION

In this section we evaluate both the performance and scalability of SALMonADA. Particularly, we focus on the overhead introduced by including SALMonADA within the consumer-provider interaction, and the maximum number of service request it is able to handle without incrementing such overhead. To do so, we first introduce how every component that appears in its architectural model (see Fig. 2) affects the performance:

SALMonADA composer, MMD Manager and ADA: We have implemented two strategies to avoid them introducing overhead: (1) they are executed concurrently, without interfering the service consumption, because the response of the provider is returned to the consumer before analysing the SLA fulfillment (see Fig. 14), and (2) they are deployed in a different location from the monitored service, and hence they do not share the same resources (e.g. CPU, RAM, etc.). Consequently, although these components might be time-consuming (e.g. the analysis of the SLA), they do not introduce an overhead over the response time of the service.

SALMon: The ESB Apache Synapse included in SALMon adds a low overhead while handling the HTTP messages. The ESB has a non-blocking HTTP transport and multi-threaded mediation, which as we measured, resulted in a negligible 1 - 3 ms overhead. Nevertheless, in our approach, there are three possible locations where SALMon can be deployed: at the server side, at the client side, or in an intermediate server (i.e. in the middle). Depending on the location, the overhead experienced by the consumer varies.

If SALMon is placed at the server or client side, there is an overhead on the resources due to the execution of the monitoring components. However, this overhead can be easily compensated by adding more resources.

If SALMon is placed in the middle, it does not produce an overhead on the resources of the client or server side. However, the deployment of SALMon in an intermediate server adds a network delay from Internet Service Providers due to the redirection of the messages. In this scenario, SALMonADA is not responsible of the overhead introduced by the network, but under heavy usage the components of SALMon might experience a bottleneck. We evaluate by means of an adequate benchmark, (1) the response time overhead under normal operating conditions (i.e. one invocation at a time), and (2) the maximum throughput SALMonADA is able to handle without incrementing such overhead.

a) Setting up the experiment: To perform the evaluation, we invoke a set of real services, and compare the response time by invoking the services both directly and through SALMon. The agents involved in the experiment are the monitored services, the client and SALMon.

To obtain a set of representative services, we started from a list of 393 services available in a public repository. Then we applied the following criteria: (1) We first considered the most recently submitted services under the assumption that recent services are more likely to be available and running than older services. Considering the length of the list, we established as threshold the 1/3 of the complete list. (2) From the resulting 131 services, we removed those ones falling into any of the following situations: were not available, were payment services, required registration or didn’t have stateless operations, resulting in 23 services. (3) We tested these 23 services and removed those ones that had errors in their descriptions (WSDL), or that gave faulty results in their functionality when invoked, resulting in a final list of 11 services from 8 different service providers, deployed on their respective servers (i.e. out of the control of the experiment), see Appendix A for the full list.

The client that invokes those services has been deployed in Seville in an Intel i7 of 2,20 GHz, 8 GB of RAM, a download speed of 13,92 Mbps and upload speed of 8,15 Mbps.

SALMon has been deployed in Barcelona in a dedicated server Intel 2,6Ghz, 6GB RAM, a download speed of 56,57 Mbps and upload speed of 15,71 Mbps.

b) Conducting the experiment: We first conducted the experiments under normal conditions executing 100 service calls per each service in both direct and redirected forms.

7http://www.xmethods.net/ve2/Directory.po
by using synchronous calls (i.e. one invocation at a time). Then, we conducted asynchronous calls to test the maximum throughput of the system starting from a throughput of 1 invocation per second to a throughput of 100 invocations per second. We performed 100 invocations per each throughput in both direct and redirected forms.

c) Analysing the results: One key issue regarding the analysis of the results is dealing with outliers (e.g. network failures that increase the response time of an invocation). Commonly used methods to deal with outliers require that the data follow a Gaussian distribution [42]. However, from the experiment results we have observed that response times do not follow a Gaussian distribution, but an exponentially modified Gaussian or inverse Gaussian distribution. For instance, the distribution of the response time of one of the monitored services is depicted in Fig. 16. As shown, the population grows rapidly on the left-hand side and decreases slowly on the right-hand side in the form of a tail. Those elements that are far away from the mean are considered outliers. To deal with these outliers, we followed the methods described and evaluated by Ratcliff for dealing with response time outliers [43]. Although Ratcliff studied response time of people in the field of psychology, the results can be applied to any model that follows the inverse Gaussian distribution. According to Ratcliff, we will not compute directly the average response time (which is not a robust estimator in front of outliers), but we will use two other robust estimators, namely, the inverse transformation and removing outliers at a standard deviation distance. The first estimator consists on applying the inverse response time (1/R) on each individual invocation, calculate the average, and then invert the result. The second estimator consists on calculating the average response time after removing the outliers at a standard deviation distance. We computed these methods over the invocations on each service for both directed and redirected invocations. As a result, we got two robust estimators per each service. We applied these estimators to the response time of direct and redirected invocations in order to calculate the response time overhead introduced in the service interaction by the deployment of SALMon in the middle. We decided to relate the two parameters with a linear interpolation curve fitting method with the aim of obtaining mathematical functions approximating the response time overhead. Fig. 17 shows the obtained functions for each of the two applied robust estimators methods, which are: \( y = 0.9x + 88.3ms \) and \( y = 1.0x + 86.3ms \).

Then, using the same estimators, we calculated what is the maximum throughput without incrementing such overhead. As shown in the results of Fig. 18, the maximum throughput of redirected invocations is 41 invocations per second. Above this number, the overhead response time raises significantly. Notice that some monitored services present a lower maximum throughput, but as shown, it is because they are unable to process such amount of invocations (either direct or redirected). We conclude that the response time overhead of the deployment of SALMon in the middle has a constant value between 86 and 89 ms, and its maximum throughput is 41 invocations per second.

We must remark that this overhead does not interfere with the SLA analysis, because the monitored response time corresponds to the real response time of the service. Nevertheless, as the consumer of the service experiences this delay, it is worthy to mention some deployment strategies to mitigate any concern.
Deploying SALMon at the client or server side is suitable for service consumers that require extremely fast response times and need to avoid the 86-89 ms overhead caused by the network delay. Moreover, as discussed before, this deployment is also suitable for services dealing with sensitive data.

On the other types of services, we argue that a deployment in the middle is preferred, since this solution is less intrusive to both the client and the provider server, as it does not require the installation of the monitor in their infrastructures. Moreover, if a throughput higher than 41 requests per second is required, multiple instances of SALMon can be replicated in different servers.

VIII. Conclusions and Discussion

In this paper we have presented a solution to monitor and analyse SLAs in order to provide timely detection and comprehensive explanations of their violations. Such information is really appealing for decision-making activities performed at runtime. For instance, the consumers of AmazonS3 scenario mentioned in Sec. I would be benefited from using SALMonADA because they are able to decide if they have to claim for a reward or not due to the service level fulfillment information provided. Other scenarios that may benefit from our proposal are the renegotiation of SLAs or the adaptation of SBSs. Our solution addresses satisfactorily the different issues identified in Sec. II:

- The SALMonADA’s supported SLAs are general-purpose because they follow the WS–Agreement [16] structure, completed with general-purpose sublanguages [18]. Thus, the considered fully-fledged language for SLAs can be easily mapped to other SLA notations. Moreover, the sublanguages notation is more human understandable than such proposed in [2], [3], [9], [11], [13], [14], [19] making it easier to be managed for human users.
- The SALMon platform is able to extract automatically from the SLA the information needed to configure the monitor in order to detect violations. This is done through a document, the Monitoring Management Document (MMD), which supports decoupling the SLA structure from the monitor service.
- SALMonADA makes use of the MMD itself to store the monitoring results coming from the monitor. The advantage of our approach is to have the MMD as the unique document that centralise all monitoring-related information without coupling to a specific API or query language.
- We introduce the Service Level Fulfillment (SLF) as a document to explain accurately the violations by identifying explicitly the violated terms and the violating monitored results. The explanations are computed through the application of a powerful CSP-based mechanism. SALMonADA supports to report such SLF information either as soon as a violation is detected (push), or when the client request it (pull).
- SALMonADA’s architecture keeps the monitor and analyser services decoupled from each other allowing thus independent evolution and eventually selective substitution. The MMD and SLF documents support this decoupling.
- The organization of SALMonADA as a SOA supports the interoperability of the platform with other tools.
- Moreover, we have performed an evaluation of the impact that SALMonADA has over the performance of the service consumption. The evaluation has been performed over real services using suitable estimators for response time to evaluate both its overhead and scalability. Although a low overhead is added either in the resources consumption or in the service response time, we expose how to mitigate it either by adding more resources, or deploying SALMonADA in alternative locations. Despite such an overhead, the client and the service provider are benefited from using our proposal. On the one hand, the client gets the real QoS of the service, which is required for several activities, such as self healing, to claim rewards due to SLA penalties, etc. On the other hand, the service provider can manage its services with the knowledge of the QoS, which is helpful to take appropriate decisions such as adding more resources to a specific service, renegotiating the SLA, etc.
- As future work we plan to study how to integrate SALMonADA on different self healing systems as such proposed in [13], [19]. In doing so, we will demonstrate that SALMonADA can be easily deployed with other monitors and analysers, as well as supporting other MMD and SLA structures.

Acknowledgment

This work has been partially supported by: S–Cube, the European Network of Excellence in Software Services and Systems; the European Commission (FEDER); the Spanish Government under the CICYT projects SETI (TIN2009–07366), TAPAS (TIN2012–32273) and ProS–Req (TIN2010–19130–C02–01); and by the Andalusian Government under the projects THEOS (TIC–5906) and ISABEL (P07–TIC–2533).

References

The automated analysis operations of WS–Agreement documents proposed in this PhD use the complex CSP-based techniques included in Chapter §7 as well as auxiliary code for agreement documents handling (XML or iAgree documents mappings to CSP, predicates parsings to CSP, etc). Implementing such document handling and CSP-based analysis operations is a time-consuming and complex task that easily leads to defects in checking and explanation solutions. In this context, we need specific testing mechanisms to increase our confidence in the quality and reliability of our developed tool. In this section, we present a test suite for ADA that comprises a set of implementation–independent test cases to validate the functionality of the eight main ADA analysis operations (included in the ICore and IChecker interfaces in Section §8.3).

For the test suite design we inspired in [120] in which popular techniques from the software testing community were used to develop a representative set of input-output combinations to validate FAMA a feature model analysis tool. Specifically, we used three black-box testing techniques to design our test cases, namely: equivalence partitioning [92, 110], pairwise testing [28, 48], and error guessing [28]. The suite is composed of 1074 test cases. Each test case is designed in terms of the inputs (an agreement offer, a template, or both) and expected outputs of the ADA analysis operations under test. Table §G shows the general inputs and outputs for each ADA analysis operation under test.

G.0.1 Inputs selection

We found two testing techniques to be helpful for the selection of a suitable set of input WS–Agreement documents, namely: equivalence partitioning and error guessing. Next, we explain these techniques and how we used them.

Equivalence partitioning. This technique is used to reduce the number of test cases to be developed while still maintaining a reasonable test coverage (i.e. the degree to which the test cases verifies the test requirements) [92, 110]. In this technique, the input domain of the program is divided into disjoint partitions (also called equivalence classes) in which the program is expected to process the set of data input in a similar (i.e. equivalent) way. According to this testing approach, only one or a few test cases of each partition are needed to evaluate the behaviour of the program for the corre-
<table>
<thead>
<tr>
<th>Operation ID</th>
<th>ADA operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>incs</td>
<td>inconsistencies checking</td>
<td>WS-Ag doc</td>
<td>Boolean value</td>
</tr>
<tr>
<td>incsExp</td>
<td>inconsistencies explanation</td>
<td>WS-Ag doc</td>
<td>Collection of (term, agreement elements)</td>
</tr>
<tr>
<td>deadTerms</td>
<td>dead terms checking</td>
<td>WS-Ag doc</td>
<td>Collection of terms</td>
</tr>
<tr>
<td>deadTermsExp</td>
<td>dead terms explanation</td>
<td>WS-Ag doc</td>
<td>Collection of (term, agreement elements)</td>
</tr>
<tr>
<td>ciTerms</td>
<td>cond.inconsistent terms checking</td>
<td>WS-Ag doc</td>
<td>Collection of terms</td>
</tr>
<tr>
<td>ciTermsExp</td>
<td>cond.inconsistent terms explanation</td>
<td>WS-Ag doc</td>
<td>Collection of (term, agreement elements)</td>
</tr>
<tr>
<td>compliance</td>
<td>compliance checking</td>
<td>Template, Offer</td>
<td>Boolean value</td>
</tr>
<tr>
<td>nonComplExp</td>
<td>non-compliance explanation</td>
<td>Template, Offer</td>
<td>Collection of (Offer term, conflicting template elements)</td>
</tr>
</tbody>
</table>

Table G.1: Inputs and outputs for ADA analysis operations addressed in the suite.

sponding partition. Thus, selecting a subset of test cases from each partition is enough to test the program effectively while keeping a manageable number of test cases.

The potential number of input WS–Agreement documents is limitless due to the possible infinite number of nested term compositors (it is not limited in WS–Agreement specification [10]) and the possible agreement element combinations including terms and creation constraints handling different kinds of predicates. To select a representative set of these, we propose dividing input WS–Agreement documents into equivalence classes according to the different comprised agreement elements, i.e. context elements, guarantee terms with or without qualifying conditions, creation constraints, etc. Therefore, according to this technique, if a WS–Agreement document with a single agreement element (e.g. a conditional guarantee term) is correctly managed by ADA, we could assume that those with more than one agreement element of the same type would also be processed successfully.

To keep equivalence classes in a manageable level, we propose dividing the input domain into the following four groups of partitions\(^1\) for those analysis operations with just a WS–Agreement document as input:

1. **Templates with different terms, but fixed context and creation constraints.** Inputs from these partitions would help us to reveal failures when processing templates with isolated terms or a set of them. 11 partitions have been created including or excluding: nested term compositors, conditional guarantee terms, service level objective with integer or string predicates, and term scopes.

2. **Templates with different creation constraints, but fixed terms.** Inputs from these partitions would help us to reveal failures when processing templates with isolated creation constraints or a set of them. 7 partitions have been created including or excluding items and general constraints of integer or string predicates.

\(^1\)We assume that obtained partitions are non-disjoint and they do not completely cover the domain, due to the high number of different agreement elements in WS–Agreement documents.
3. **Agreement Offers with different context, but fixed terms.** Inputs from these partitions would help us to reveal failures when processing agreement offers with isolated context. 2 partitions have been created depending on the specification of a mandatory information in the context.

4. **Agreement Offers with different terms, but fixed context.** Inputs from these partitions would help us to reveal failures when processing agreement offers with isolated terms or a set of them. 11 partitions have been created including or excluding: nested term compositors, conditional guarantee terms, service level objective with integer or string predicates, and term scopes.

As a result of the application of this technique we got 31 WS–Agreement documents representing a manageable part of the input domain for such analysis operations with a single WS–Agreement document as input.

**Error guessing.** This is a software testing technique based on the ability of the tester to predict where faults are located according to its experience on the domain [28]. Using this technique, test cases are specifically designed to exercise typical error-prone points related to the type of system under test.

Following the guidelines of error guessing, we propose the use of conflicting WS–Agreement documents including one or more conflicts inside as suitable inputs to check whether inconsistencies, dead terms, conditionally inconsistent terms, and non-compliance are correctly checked or explained by ADA analysis operations. This way, we kept the number of input documents for this techniques under test in a reasonable level while still having a fair confidence in the ability of our tests to reveal failures. For each kind of conflict we designed a different set of inputs comprising 11 Templates and 7 agreement offers for conflicts of one WS–Agreement document; and 7 template-offer pairs for compliance conflicts.

For instance the template of a translation service provider depicted in Figure §G.1 was included to check if ADA realized about conflicts involving several agreement elements. Specifically, the template has an inconsistency between G1 and G2 by C1 constraint. The reason for that inconsistency is as follows: as the G1 SLO makes the C1 implication antecedent be fulfilled, the implication consequent is taken into account. Such a consequent satisfies the qualifying condition of G2 and so its SLO is enabled being inconsistent with G1. Note that although Service description term includes the inconsistent constraint InputFileSize=10, it is not considered because service description terms values in templates are simply by default values as Section §2.2 details.

Due to the complexity of handling a template and an agreement offer together in order to analyse compliance conflicts, we prepared 20 template-offer pairs more guided by our experience. They were divided in the following three groups:
Figure G.1: Template got from error guessing technique

1. 7 pairs of templates with different terms, but fixed context and creation constraints; against fixed agreement offers.

2. 7 pairs of templates with different creation constraints, but fixed terms; against fixed agreement offers.

3. 6 pairs of agreement offers with different terms, but fixed context; against fixed templates.

G.0.2 Inputs combination

We found pairwise testing technique to be helpful for the inputs combination in order to get a wider inputs sample for our test suite. Next, it is summarized the technique description and our own application.
Pairwise testing (also called 2-wise testing). This is a combinatorial software testing method focusing on testing all possible discrete combinations of two input parameters [28, 48]. According to the hypothesis behind this technique, most common errors involve one input parameter. The next most common category of errors consists of those dependent on interactions between pairs of parameters and so on. Thus, the main goal of this technique is to address the second most common cases of errors (those involving two parameters) while keeping the number of test cases in a manageable level.

Although pairwise is specially recommended to combine input values in those programs receiving more than one input parameter we used it for such analysis operations that receive just a single WS–Agreement document as input as follows: we consider interesting to combine the agreement elements of two of the partition classes exposes in previous section in order to obtain a more significant number of use cases covering more combinations of agreement elements. Thus, we used a pairwise combination strategy to create a test case for each possible combination of elements of two partition classes. As an example, we create templates with different terms and different creation constraints combinations; or agreement offers with different terms and a context without several information. Note that some combinations were not applicable (e.g. agreement offers cannot include creation constraints). Thus, our test suite is improved with 81 additional input documents for the use of this pairwise strategy.

For compliance and nonComplExp analysis operations, that receives two WS–Agreement documents as inputs (an agreement offer and a template), we applied pairwise by creating a test case for each possible combination of agreement offers and templates got from the equivalence partitions selected for the analysis operations that receive a single document as input.

As an example, consider the operation incs. In previous section, we studied these inputs in isolation and selected representative 18 templates and 13 agreement offers. Using pairwise testing for compliance and nonComplExp analysis operations, we may create 234 test cases, a test case for each possible combination of documents (i.e. 18*13 potential test cases). Due to some combination were not applicable (e.g. fixed template-fixed offer is dummy) and to create a manageable number of inputs we just consider such 120 pairs including: templates with different terms and creation constraints, against fixed offers (42 inputs); and templates against offers both with different terms and creation constraints (78 inputs).

G.0.3 Test cases report

To conclude the design of our suite, we organized the selected inputs and their expected outputs into test cases; 1074 in total as table §G.0.3 shows. For their specification, we followed the guidelines of the IEEE Standard for Software Testing Documentation [2]. As an example, Table §G.3 depicts four of the test cases included in the suite to test explaining operations. For each test case, an ID, description, inputs, expected outputs and intercase dependencies (if any) are presented. Intercase depen-
Table G.2: Number of inputs got from each testing technique for the ADA analysis operations

The four samples included in Table §G.3 require the previous execution of a checking technique to assure that inputs includes a conflict inside. As an example, test case EI-31 tries to reveals failures when obtaining the explanation of inconsistencies between a guarantee term of a nested term compositor and a service description term. Note that test case I-31 should be executed beforehand to assure that there exist an inconsistency in such an input. As a more complex test case example, test case ENC-105 tries to reveals failures when obtaining the explanation of non-compliance between a guarantee term of the agreement offer and a creation constraint of the template. Note that test case CC-105 should be executed beforehand to assure that there exist a non-compliance conflict in such an input. Moreover, CC-105 requires a previous execution of CDC-13, and CDC-24 test cases to assure each input were conflict-free, and so on.

G.0.4 Test cases adequacy criteria

The test suite adequacy can be denoted by many factors such as a simple enumeration of errors detected by the test suite; or more exhaustive techniques as mutation testing [32] to refine the test cases; or studying the whole code covering of the test suite to know the tested part of the code. In future work we will apply such exhaustive techniques but currently, for the sake of simplicity, we expose the adequacy of our test suite by an enumeration of the detected errors. Thanks to the test cases we found more than 10 errors while handling or analysing WS–Agreement documents with ADA.

Errors detected while handling documents. We found that several mandatory agreement elements were not correctly managed by ADA. This is the case of the creation constraint section in templates and template name and identifier in the agreement offers context. Moreover, WS–Agreement documents with variants and scopes were not correctly managed because a NullPointerException were thrown. Both cases were solved.
<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>Description</th>
<th>Input</th>
<th>Expected Output</th>
<th>Deps</th>
</tr>
</thead>
<tbody>
<tr>
<td>EI-31</td>
<td>Detecting errors while explaining inconsistencies between a guarantee term of a nested term compositor with a service description term</td>
<td>Agreement Offer with OneOrMore term compositor with a nested ExactlyOne of 2 guarantee terms</td>
<td>Image Translation of Service description, inconsistent by term GTTranslationTime2</td>
<td>I-31</td>
</tr>
<tr>
<td>EDT-124</td>
<td>Detecting errors while explaining the origin of two dead terms by the use of a contradictory predicate in qualifying conditions</td>
<td>Agreement Offer with two guarantee terms with contradictory qualifying conditions</td>
<td>term GTInputErrors1 is dead by term GTInputErrors1; term GTInputErrors2 is dead by term GTInputErrors2</td>
<td>DT-124</td>
</tr>
<tr>
<td>ECIT-18</td>
<td>Detecting errors while explaining the origin of two conditionally inconsistent terms by the same creation constraint</td>
<td>Template with 3 creation constraints with an integer predicate, and a another constraint with a string predicate</td>
<td>term GTTranslationTime1 is conditionally inconsistent by creation constraint TranslationTime; term GTTranslationTime2 is conditionally inconsistent by creation constraint TranslationTime</td>
<td>CIT-18</td>
</tr>
<tr>
<td>CC-105</td>
<td>Detecting errors while checking the compliance between a guarantee term of the offer and a creation constraint of the template</td>
<td>Template with a service description term and 2 creation constraints; and an Offer with conditional guarantee term</td>
<td>The Agreement Offer is non-compliant with the Template</td>
<td>CDC-13, CDC-24</td>
</tr>
<tr>
<td>ENC-105</td>
<td>Detecting errors while explaining the origin of non-compliance between a guarantee term of the offer and a creation constraint of the template</td>
<td>Template with a service description term and 2 creation constraints; and an Offer with conditional guarantee terms</td>
<td>term GTTranslationTime1 of the Offer is non-compliant with creation constraint TranslationTime of the Template</td>
<td>CC-105</td>
</tr>
</tbody>
</table>

Table G.3: Four of the test cases included in the suite with an adequate exception treatment.

Errors detected while analysing documents. We found several errors while analysing WS–Agreement documents with specific combinations of agreement elements:

- Attribute-value pairs of service description terms in templates are by default values but it were considered as value assignments.

- In WS–Agreement documents with variants inside, dead and conditionally inconsistent terms were returned by deadTerms and ciTerms operations several times. Specifically, in the same number of variants that include them.

- When deadTerms or ciTerms operation were used with a conflicting document a NullPointerException were returned.
• When a term were dead or conditionally inconsistent by an attribute-value pair assignment of a service description term, the responsible value assignment were not explained by \texttt{deadTermsExp} or \texttt{ciTermsExp} operation.

• When the cause of a term to be dead or conditionally inconsistent were not found in a variant, the term were classified as non dead or conditionally inconsistent by \texttt{deadTermsExp} or \texttt{ciTermsExp} operation before studying the other variants.

• When several dead or conditionally inconsistent terms were included in the same variant of a document and all of them with the same qualifying condition, only one of them were returned by \texttt{deadTerms} or \texttt{ciTerms} operation.

• When the cause of a term to be dead or conditionally inconsistent is itself, the explanation were not correctly returned by \texttt{deadTermsExp} or \texttt{ciTermsExp} operation.


The need for Service Level Agreements (SLAs) to supervise the services consumption is increasing in business applications and Cloud scenarios due to violation of SLA terms may carry penalties to the parties. Such a need for SLAs boosts the importance and complexity of SLA lifecycle supporting systems, SLA-driven solutions from now on. After a study of the existing literature we conclude that these SLA-driven solutions can be significantly enhanced in both: (1) the SLA specification languages, and (2) the techniques to extract useful information from the SLAs, analysis techniques, in advance.

In the one hand, we suggest that existing SLA specification languages can be improved by turning them into: i) domain-independent, ii) standard-recommendation-compliant, and iii) expressive-enough to ease the description of Service Level Objectives (SLOs), constraints, and validity periods for their comprised elements. In addition, the specification languages must define validity criteria for checking that supported SLAs satisfy some basic properties such as the consistency, and the compliance between SLA documents. Furthermore, explanations must be provided when the SLAs do not satisfy such basic properties. On the other hand, the proposed analysis techniques include some drawbacks that can be found in emerging techniques. To overcome such drawbacks there must be developed: a) fully-functional reference implementations, b) techniques with a reuse-oriented design, c) effective extension mechanisms, and d) user-friendly interfaces.

The main goal of this dissertation is to improve the current support to develop SLA-driven solutions by adopting the WS-Agreement standard recommendation and considering the aforementioned suggestions for improvement.

The main thesis contributions are a WS--Agreement-compliant language called iAgree, and an Integrated Development Environment for SLA-driven Solutions (IDEAS) whose main component is an iAgree Document Analyser (ADA). As minor contributions, two solutions have been developed to improve the existing tooling support to create and monitor SLAs.

The cornerstone of our proposal to improve the SLA specification languages has been the definition of the WS-Agreement Configurations that comprise the sublanguages needed to describe the different parts of WS-Agreement documents. Regarding our proposal to improve the analysis techniques, the key has been the organization of such techniques in a catalogue of basic analysis operations that can be combined to support more advanced SLA-driven solutions. The applicability of our results is limited to those SLAs that can be translated to a constraint satisfaction problem, that is enough to support real-world SLAs, in our experience.