An Initial Approach to Explaining SLA Inconsistencies*

Carlos Müller, Antonio Ruiz-Cortés, Manuel Resinas

Dpto. Lenguajes y Sistemas Informáticos
ETS. Ingeniería Informática - Universidad de Sevilla (Spain - España)
41012 Sevilla (Spain - España)
{cmuller, aruiz, resinas}@us.es

Abstract An SLA signed by all interested parties must be created carefully, avoiding contradictions between terms, because their terms could carry penalties in case of failure. However, this consistency checking may become a challenging task depending on the complexity of the agreement. As a consequence, an automated way of checking the consistency of an SLA document and returning the set of inconsistent terms of the agreement would be very appealing from a practical point of view. For instance, it enables the development of software tools that make the creation of correct SLAs and the consistency checking of imported SLAs easier for users. In this paper, we present the problem of explaining WS-Agreement inconsistencies as a constraint satisfaction problem (CSP), and then we use a CSP solver together with an explanation engine to check the consistency and return the inconsistent terms. Furthermore, a proof-of-concept using Choco solver in conjunction with the Palm explanation engine has been developed.
Keywords: Service Level Agreement, WS-Agreement, Consistency Checking, Debugging, Quality of Service.

1 Introduction

SLAs consist of a set of terms that include information about functional features, non-functional guarantees, compensation, termination terms and any other terms with relevant information to the agreement. An agreement signed by all interested parties should be redacted carefully because a failure to specify their terms could carry penalties to the initiating or responding party. Therefore, agreement terms should be specified in a consistent way, avoiding contradictions between them. However, depending on the complexity of the agreement, this may become a challenging task. For instance, in a scenario in which a provider offers computing services to other organizations, an SLA could be agreed that includes non-functional attributes such as: the *availability* -in percentage-, the *mean time between two consecutive requests of the service* (MTBR) -in seconds-, and the

* This work has been partially supported by the European Commission (FEDER), Spanish Government under CICYT project Web-Factories (TIN2006-00472), and project P07-TIC-2533 funded by the Andalusian local Government.

efficiency. Assuming that: Availability ranges between [90..100] , MTBR ranges between ∈ [5..60] and Efficiency = Availability/MTBR. If the SLA includes a term obligating to guarantee Efficiency > 20, at first sight the SLA is consistent. However the highest valid value for efficiency is 100/5=20, so this term cannot be satisfied. Therefore, a consistency checker that automatically checks the SLA for inconsistencies between its terms would be very appealing from a practical point of view.

Furthermore, it is of interest not only to obtain an automated way of checking the consistency of an SLA document, but also to return an explanation if the document is inconsistent. This explanation is the set of inconsistent terms of the agreement. So, in the previous scenario we would obtain as debugging information that the inconsistent terms are: [(Efficiency > 20), (Availability ∈ [90..100]), (MTBR ∈ [5..60]), (Efficiency = Availability/MTBR)], because they are inconsistent terms. This automated consistency checking enables the implementation of a software tool which makes the creation of correct SLAs and the consistency checking of imported SLAs easier for users.

Nevertheless, as far as we know, the consistency of SLAs has been taken for granted by most authors. In this paper, we describe a mechanism to check and explain SLAs specified with WS-Agreement [2], which is a proposed recommendation of the Open Grid Forum (OGF) which provides a schema for defining SLAs and a protocol for creating them. To this end, we map the terms of a subset of the WS-Agreement document into a constraint satisfaction problem (CSP). Then we use the CSP as an input for a CSP solver with an explanation engine, which will return the set of inconsistent constraints. Finally, we trace back the constraints to the original SLA terms in order to give useful debugging information to users.

As a proof-of-concept, we have developed a prototype of our consistency analyser using Choco solver [1] in conjunction with the Palm explanation engine [3]. This prototype is available for testing at `http://www.isa.us.es` in the tools section.

This paper is structured as follows. Section 2 presents some background on constraint satisfaction problems and WS-Agreement. Section 3 details the subset of WS-Agreement which is used to explain the SLA inconsistencies in 3.1 and its mapping to CSP in 3.2. Section 4 describes our process to explain the WS-Agreement* iconsistencies. Section 5 shows our proof-of-concept. Section 6 reports on the related proposals. Finally, Section 7 details our conclusions and future work.

2 Preliminaries

2.1 Constraint Satisfaction Problems

Constraint Satisfaction Problems [8] have been the object of research in Artificial Intelligence over the last few decades. A Constraint Satisfaction Problem (CSP) is defined as a set of variables, each ranging on a finite domain, and a set of

constraints restricting all the values that these variables can take simultaneously. A solution to a CSP is an assignment of a value from its domain to every variable, in such a way that all constraints are satisfied simultaneously. These are some basic definitions of what a CSP is.

Definition 1 (CSP). *A CSP is a three–tuple of the form* (V, D, C) *where* $V \neq \emptyset$ *is a finite set of variables,* $D \neq \emptyset$ *is a finite set of domains (one for each variable) and* C *is a constraint defined on* V*.*

Consider, for instance, the CSP: $(\{a, b\}, \{[0..2], [0..2]\}, \{a + b < 4\})$

Definition 2 (Solution). *Let* ψ *be a CSP, a solution of* ψ *is whatever valid assignment of all elements in* V *that satisfies* C*.*

In the previous example, a possible solution is $(2, 0)$ since it verifies that $2 + 0 < 4$.

Definition 3 (Solution space). *Let* ψ *be a CSP of the form* (V, D, C)*, its solution space denoted as* $sol(\psi)$ *is made up of all its possible solutions. A CSP is satisfiable if its solution space is not empty.*

$$sol(\psi) = \{S \mid \forall s_i \cdot s_i \in S \Rightarrow C(s_i) = true\}$$

In the previous example there are eight solutions. The only assignment that does not satisfy $a + b < 4$ is $(2, 2)$. Nevertheless, if we replace the constraint with $a + b < -1$, then the CSP is not satisfiable.

In many real-life applications, if a CSP has no solution, we would like to know which set of constraints are responsible for this situation. This can be done by interpreting the CSP as an explanation problem.

Definition 4 (Explanation problem). *Let* ϵ *be a CSP of the form* (V, D, C) *with an empty solution space:* $sol(\epsilon) = \emptyset$*. It is considered to be an explanation problem if its objective is to find a set of constraints* $C' \subset C$ *that cannot be satisfied.*

Definition 5 (Explanations). *Let* ϵ *be an explanation problem, the resulting set of inconsistent constraints* C' *are known as the explanations for the problem. They are divided into two parts: a subset of the original set of constraints* $C' \subset C$ *and a subset of decision constraints introduced so far in the search of solutions* $(dc_1, ..., dc_k)$*.*

As defined in [3], a contradiction explanation, also known as "nogood" [7], is a subset of the constraints of the problem that, left alone, leads to a contradiction (no feasible solution contains a nogood).

The previous CSP example, with the constraint replaced with $a + b < -1$, is not satisfiable, and by interpreting it as an explanation problem the explanation engine should obtain as the set of inconsistent constraints: $[(a + b < -1), (a >= 0), (b >= 0)]$

2.2 WS-Agreement in a Nutshell

WS-Agreement specifies an XML-based language and a protocol for advertising the capabilities and preferences of service providers, and creating agreements based on agreement offers. The structure of an agreement in WS-Agreement comprises:

- **Name**: it identifies the agreement and can be used for reference.
- **Context**: it includes information such as the name of the parties and their roles as initiator or responder in the agreement. Additionally, it can include other important information for the agreement.
- **Terms**: they are grouped by the following term compositors: `ExactlyOne`, `OneOrMore`, or `All`. The two main types of terms are:
  - *Service terms*: they provide service information by means of:
    - Service description terms and service references, which includes information to instantiate or identify the services and operations involved in the agreement.
    - Service properties, which includes the measurable properties that are used in expressing guarantee terms. They consist of a set of variables whose values can be established inside the service description term, and whose domain can be established by the `metric` attribute pointing to an external XML document.
  - *Guarantee terms*: they describe the service level objectives (SLO) agreed by a specific obligated party, either using a free-form element or using a key performance indicator. It also includes the scope of the term (e.g. if it applies to a certain operation of a service or the whole service itself), and a qualifying condition that specifies the validity condition under which the term is applied.

Figure 1 depicts an example of a WS-Agreement between a computing services provider and a consumer. It defines several service properties whose domain is specified in an external XML document (depicted in Figure 2). Note that other XML documents, such as an XML Schema definition, could have been used instead. The service properties defined in the WS-Agreement document of Figure 1 are the following ones:

- the *availability* -integer form 1 to 100-
- the *mean time between two consecutive requests of the service* (MTBR) -integer greater than 1-
- the *mean time to response* (MTTR) -integer greater than 1-
- the *initial cost for the service* (InitCost) -integer greater than 1-
- the *final cost for the service* (Cost) -integer greater than 1-
- the *increase of the cost if the MTBR* < 10 (ExtraMTBRCost) -integer greater than 1-
- the *increase of the cost if the MTTR* < 05 (ExtraMTTRCost) -integer greater than 1-

We have extracted the following information from the SLA:

- MTBR $\in$ [5..60].
- MTTR $\in$ [1..10].
- If MTBR >= 10 Then Availability $\in$ [90..100].
- If MTBR < 10 Then Availability $\in$ [95..100].
- Cost = InitCost + ExtraMTBRCost + ExtraMTTRCost.
- If MTBR < 10 Then ExtraMTBRCost = 15.
- If MTTR < 05 Then ExtraMTTRCost = 15.
- If MTBR >= 10 and MTTR >= 05 Then ExtraMTBRCost = 0 and ExtraMTTRCost = 0.

3 Mapping WS-Agreement onto CSP

3.1 WS-Agreement* as a subset of WS-Agreement

Due to the flexibility and extensibility of WS-Agreement, we focus in this paper on a subset of WS-Agreement that is a bit less expressive than WS-Agreement, but still useful for our purpose. The subset of WS-Agreement is called WS-Agreement*. A WS-Agreement* document (α) is composed of the following elements:

- Service properties must define all variables that are used in the guarantee terms. In addition, attribute `metric` of the variables is mandatory. This attribute must point to another XML document or schema that provides the data type and a general range of values (δ) for each service property. Figure 2 shows an example of an ad-hoc XML document that includes the mentioned information for service properties of the example of Figure 1, although other XML documents could be used. XML node `Location` (λ_υ) of each variable establishes the specific XML node inside the service description term where it is defined the value for such variable (value(λ_υ)).
- Terms (T) can be composed using the three term compositors defined in WS-Agreement: `All` ($\odot$), `ExactlyOne` ($\otimes$), and `OneOrMore` ($\oplus$).
- Service description terms can be included but only to impose specific value definitions for each variable (υ) of service properties. Other service description terms could be added but they are ignored when checking the SLA consistency.
- Guarantee terms (γ) can be included. Both qualifying condition (κ) and the SLOs (σ) must be defined as constraints on the variables defined in the service properties, and only to those variables (i.e. $\kappa \in C$ and $\sigma \in C$). The scope of a guarantee term cannot be established, but all guarantee terms apply to the whole service.

Note that, although WS-Agreement* is not as expressive as WS-Agreement, it does allow for the expression of complex agreements. For instance, the agreement depicted in Figure 1 is compatible with WS-Agreement*.

Thus, we can formally define an agreement specified with WS-Agreement* as follows:

```
<Agreement ''id=exampleScenario''>
     <Context>   ...    </...>
     <All>
          <ServiceDescriptionTerm Name=''ComputingService''>
               <...>
                    ...              </...>
                    <InitCost>      20   </...>
                    ...              </...>
               </...>
          </ServiceDescriptionTerm>
          <ServiceProperties>
            ...<Variable name=''Availability'' metric=''metricXML:Availability''>
                    <Location> \\Availability </Location>
               </Variable>
               <Variable name=''MTBR'' metric=''metricXML:MTBR''>
                    <Location> \\MTBR </Location>
               </Variable>
               <Variable name=''MTTR'' metric=''metricXML:MTTR''>
                    <Location> \\MTTR </Location>
               </Variable>
               <Variable name=''InitCost'' metric=''metricXML:InitCost''>
                    <Location> \\InitCost </Location>
               </Variable>
               <Variable name=''Cost'' metric=''metricXML:Cost''>
                    <Location> \\Cost </Location>
               </Variable>

            <Variable name=''ExtraMTBRCost'' metric=''metricXML:ExtraMTBRCost''>
                    <Location> \\ExtraMTBRCost </Location>
               </Variable>

            <Variable name=''ExtraMTTRCost'' metric=''metricXML:ExtraMTTRCost''>
                    <Location> \\ExtraMTTRCost </Location>
               </Variable>...
          </ServiceProperties>
          <GuaranteeTerm Name=''MTBRDomain''>
               <SLO>   MTBR >= 5 and MTBR <= 60  </...>
          </GuaranteeTerm>
          <GuaranteeTerm Name=''MTTRDomain''>
               <SLO>   MTTR >= 1 and MTTR <= 10  </...>
          </GuaranteeTerm>
          <GuaranteeTerm Name=''CostDefinition''>
               <SLO>   Cost = InitCost + ExtraMTBRCost + ExtraMTTRCost  </...>
          </GuaranteeTerm>
          <ExactlyOne>
               <GuaranteeTerm Name=''LowerAvailabilityDomain''>
                    <QualifyingCondition> MTBR >= 10 </...>
                    <SLO>   Availability >= 90 and Availability <= 100  </...>
               </GuaranteeTerm>
               <GuaranteeTerm Name=''HigherAvailabilityDomain''>
                    <QualifyingCondition> MTBR < 10 </...>
                    <SLO>   Availability >= 95 and Availability <= 100  </...>
               </GuaranteeTerm>
          </ExactlyOne>
          <OneOrMore>
               <GuaranteeTerm Name=''MTBRIncrement''>
                    <QualifyingCondition> MTBR < 10 </...>
                    <SLO>  ExtraMTBRCost = 15  </...>
               </GuaranteeTerm>
               <GuaranteeTerm Name=''MTTRIncrement''>
                    <QualifyingCondition> MTTR < 05 </...>
                    <SLO>  ExtraMTTRCost = 15  </...>
               </GuaranteeTerm>
               <GuaranteeTerm Name=''CheaperCost''>
                    <QualifyingCondition> MTBR >= 10 and MTTR >= 05 </...>
                    <SLO>   ExtraMTBRCost = 0 and ExtraMTTRCost = 0  </...>
               </GuaranteeTerm>
          </OneOrMore>
      </All>
</Agreement>
```

Figure 1. Example of a WS-Agreement document with all kinds of compositors.

```
<metricXML>
    <Availability type=''integer'' min=''1'' max=''100''/>
    <MTBR type=''integer'' min=''1'' max=''unbounded''/>
    <MTTR type=''integer'' min=''1'' max=''unbounded''/>
    <InitCost type=''integer'' min=''1'' max=''unbounded''/>
    <Cost type=''integer'' min=''1'' max=''unbounded''/>
    <ExtraMTBRCost type=''integer'' min=''1'' max=''unbounded''/>
    <ExtraMTTRCost type=''integer'' min=''1' max=''unbounded''/>
</metricXML>
```

Figure 2. Ad-hoc XML document for the variable domains of Figure 1.

Definition 6 (A WS-Agreement* document). *A WS-Agreement* document α is a set of variables υ_i, variable domains δ_i, and a set of terms T, including service description terms, guarantee terms and terms compositors as follows:*

$$\alpha = (\upsilon_i, \ldots, \upsilon_n, \delta_i, \ldots, \delta_n, T_1, \ldots, T_m)\,, \text{ where } T_i = \left\{ \begin{array}{l} \lambda_\upsilon \\ \gamma \\ T_{i_1} \odot \ldots \odot T_{i_k} \\ T_{i_1} \otimes \ldots \otimes T_{i_k} \\ T_{i_1} \oplus \ldots \oplus T_{i_k} \end{array} \right\}$$

3.2 Mapping WS-Agreement* onto CSP

Figure 3 depicts the mapping (μ) of a WS-Agreement* document (α) onto an equivalent CSP, (ψ_α). The variables (υ) defined inside the service properties are the CSP variables; the variable domains (δ) included in the document specified by the metric attribute are the CSP variable domains; and the constraints from the service description terms (λ_υ), guarantee terms (γ) and term compositors ($\odot$ as a logic "AND", $\otimes$ as logic "XOR", and $\oplus$ as logic "OR") are the CSP constraints.

Thus, in general, our WS-Agreement* to CSP mapping can be defined as follows:

Definition 7 (Mapping an WS-Agreement* to CSP). *The mapping ($\mu : \alpha \rightarrow \psi$) of a WS-Agreement* document (α) to a CSP (ψ) can be defined as follows:*

$$\mu(\alpha) = \mu\left(\upsilon_1, \ldots, \upsilon_n, \delta_1, \ldots, \delta_n, T_1, \ldots, T_m\right) =$$
$$= \left(\{\upsilon_1, \ldots, \upsilon_n\}, \{\delta_1, \ldots, \delta_n\}, \{\mu_T(T_1, \ldots, T_m)\}\right) = \psi_\alpha$$

where $\mu_T : T \rightarrow C$ is a mapping function of terms into constraints defined as follows:

$$\mu_T(T) \equiv \left\{ \begin{array}{ll} \upsilon = \lambda_\upsilon & \textit{if } T \textit{ is a service description term } (\lambda_\upsilon) \\ \sigma & \textit{if } T \textit{ is a guarantee term without qualifying condition } (\gamma_\sigma) \\ \kappa \Rightarrow \sigma & \textit{if } T \textit{ is a guarantee term with a qualifying condition } (\gamma_{\sigma,\kappa}) \\ \bigwedge_{i=1}^{k} \mu_T(T_i) & \textit{if } T \textit{ is a composite term } (T_1 \odot \ldots \odot T_k) \\ \bigwedge_{i=1}^{k} \mu_T(T_i) \Leftrightarrow (\bigwedge_{j=1 \setminus j \neq i}^{k} \neg \mu_T(T_j)) & \textit{if } T \textit{ is a composite term } (T_1 \otimes \ldots \otimes T_k) \\ \bigvee_{i=1}^{k} \mu_T(T_i) & \textit{if } T \textit{ is a composite term } (T_1 \oplus \ldots \oplus T_k) \end{array} \right\}$$

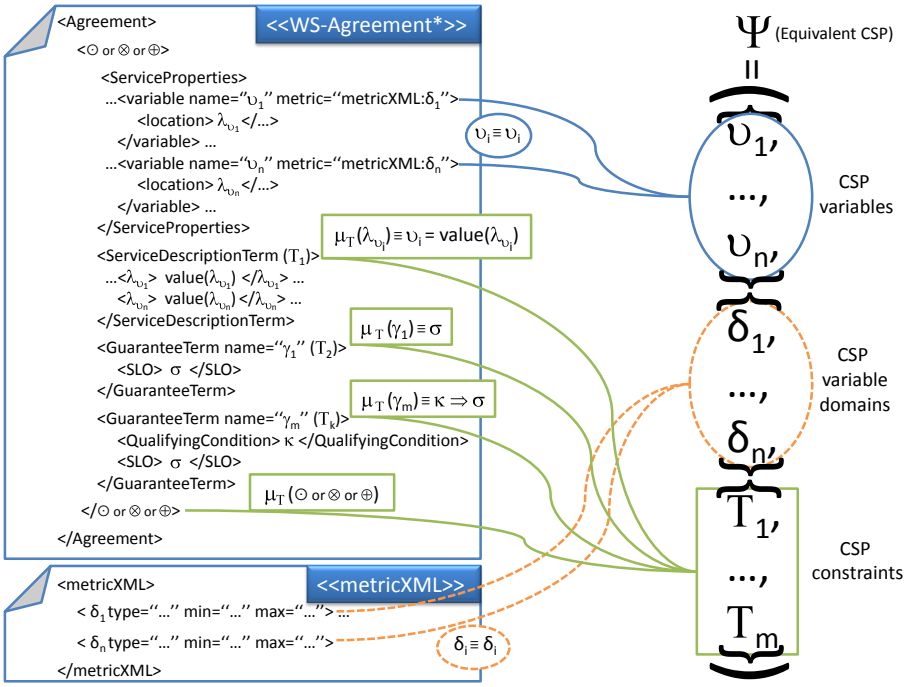

Figure 3. Summary of WS-Agreement* to CSP mapping.

Using this mapping, the ψ_α for the example of Figure 1 is mapped as follows:

ψ_α = ({ Availability, MTBR, MTTR, InitCost, Cost, ExtraMTBRCost, ExtraMTTRCost },
{ [1 . . . 100], [1 . . . ∞), [1 . . . ∞), [1 . . . ∞), [1 . . . ∞), [1 . . . ∞), [1 . . . ∞), },
{ InitCost = 20,
MTBR ≥ 5 and MTBR ≤ 60,
MTTR ≥ 1 and MTBR ≤ 10,
Cost = InitCost + ExtraMTBRCost + ExtraMTTRCost,
((MTBR ≥ 10) ⇒ (Availability ≥ 90 and Availability ≤ 100)) ⇔
⇔ ¬ ((MTBR < 10) ⇒ (Availability ≥ 95 and Availability ≤ 100)),
(MTBR < 10 ⇒ ExtraMTBRCost = 15) ∨
∨ (MTTR < 5 ⇒ ExtraMTTRCost = 15) ∨
∨ ((MTBR ≥ 10 and MTTR ≥ 5) ⇒ ExtraMTBRCost = 0 and ExtraMTTRCost = 0) }
)

The following table denotes constraints mapped from the example of Figure 1.

Example term (T_i) name	**Equivalent mapped ($\mu_T(T_i)$)**
InitCost	InitCost = 20
MTBRDomain	MTBR >= 5 and MTBR <= 60
MTTRDomain	MTTR >= 1 and MTTR <= 10
CostDefinition	Cost = InitCost + ExtraMTBRCost + ExtraMTTRCost
LowerAvailabilityDomain	(MTBR $\geq$ 10) $\Rightarrow$ (Availability $\geq$ 90 and Availability $\leq$ 100)
HigherAvailabilityDomain	(MTBR $<$ 10) $\Rightarrow$ (Availability $\geq$ 95 and Availability $\leq$ 100)
MTBRIncrement	(MTBR $<$ 10) $\Rightarrow$ (ExtraMTBRCost = 15)
MTTRIncrement	(MTTR $<$ 5) $\Rightarrow$ (ExtraMTTRCost = 15)
CheaperCost	(MTBR $\geq$ 10 and MTTR $\geq$ 5) $\Rightarrow$ (ExtraMTBRCost = ExtraMTTRCost = 0)
ExactlyOne	(LowerAvailabilityDomain $\Leftrightarrow \neg$ HigherAvailabilityDomain) $\wedge$ (HigherAvailabilityDomain $\Leftrightarrow \neg$ LowerAvailabilityDomain)
OneOrMore	MTBRIncrement $\vee$ MTTRIncrement $\vee$ CheaperCost
All	InitCost $\wedge$ MTBRDomain $\wedge$ MTTRDomain $\wedge$ $\wedge$ CostDefinition $\wedge$ ExactlyOne $\wedge$ OneOrMore

Table 1. Mapping example terms to CSP constraints

4 Checking and Explaining WS-Agreement* Inconsistencies

Checking the consistency of WS-Agreement documents involves both syntactic and semantic checking. The former involves checking the document against the WS-Agreement XML Schema. The latter, however, is harder to check on and is thus the focus of this paper.

Figure 4 depicts our consistency checking process. In this scenario, it is imposed by a human error, an MTBR value definition inside the service description term with a value of 61. Thus, the ψ_α would not be satisfiable by the MTBR domain definition. As depicted in Figure 4, we propose to use the agreement specified with WS-Agreement* in conjunction with the XML document which defines the variables metrics as the two inputs for a mapping component which implements the mapping defined in section 3.2. Once the agreement is mapped to a CSP, the explanation engine of the CSP explainer component will obtain whether the SLA document is consistent or not. In the latter case, the component sends to a tracing component the explanations for the problem. In this case, the explanations for the problem are ϵ_{MTBR}: [MTBR = 61; MTBR $\geq$ 5 and MTBR $\leq$ 60]. The tracing component converts the explanations into the equivalent inconsistent original terms in order to give useful information to users. This is done by naming constraints mapped from an SLO as the name of the guarantee term which includes it; and if the constraint was mapped from a value definition inside a service description term, we name it "SDT", concatenate with the name of the variable. Then, the previous explanation for MTBR would be traced by us to the inconsistent term "SDTMTBR" constraint, showing that it is the MTBR

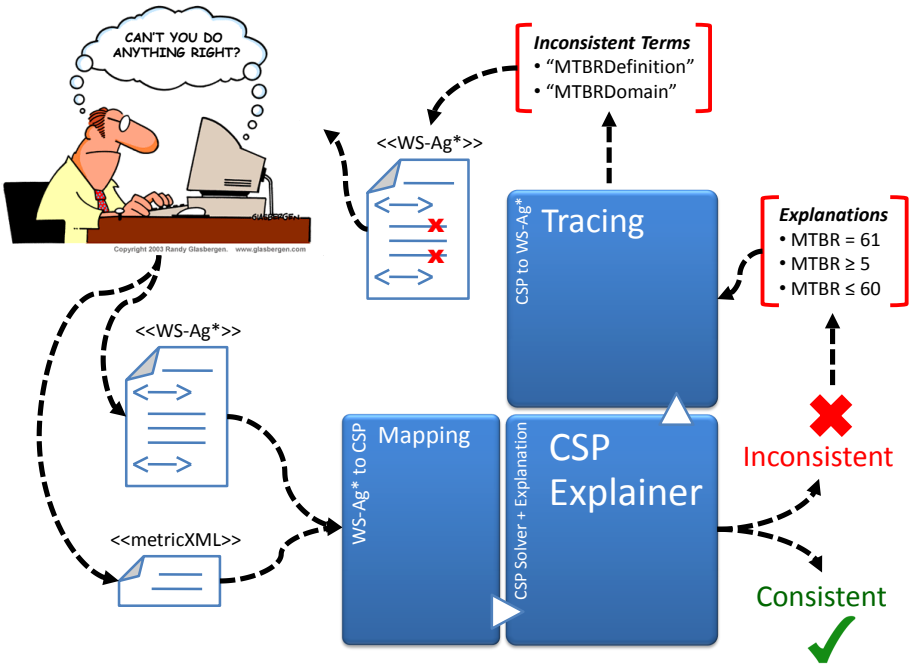

Figure 4. WS-Agreement compliant process for explaining SLA inconsistencies.

value assignment and the domain restriction of the "MTBRDomain" guarantee term. The user should then grasp that the MTBR value definition is inconsistent with the MTBR domain.

5 A proof-of-concept

We have developed a proof-of-concept with the Choco constraint solver [1] and the Palm explanation engine [3]. This proof-of-concept receives two XML documents as input: the SLA WS-Agreement* document and the metric XML document. After processing the inputs, our consistency checker returns whether the document is consistent or not and its explanation in the latter case. We have excluded the syntax consistency checking for simplicity.

The current Palm library included in Choco v.1.2.03 only gives complete support to integer variables and constraints with logical operators like $\geq, \leq, =, \neq$, ... Boolean operators and implications like *if constraint1 then constraint2* and assignments like var1 = var2 × var3 are excluded. Thus, to test our proof-of-concept, we have simplified our previous SLA example as follows:

– Variables remain equal, unlike in the previous example, because they are all integer variables. But we choose only three of them: availability, cost and

increment, and we have added a new initial cost. The unbounded integer domains must be bounded and we assign a maximum value of 10.000 by default as Figure 5 depicts.

```
<metricXML>
    <Availability type=''integer'' min=''1'' max=''100''/>
    <Cost type=''integer'' min=''1'' max=''10000''/>
    <Increment type=''integer'' min=''1'' max=''10000''/>
    <InitCost type=''integer'' min=''1'' max=''10000''/>
</metricXML>
```

Figure 5. Ad-hoc XML document for the variable domains of Figure 6.

– We have removed the qualifying conditions from the guarantee terms because implications are not supported currently. As a consequence, the term compositor elements `OneOrMore` and `ExactlyOne` do not make sense in the simplified example, so an unique `All` term compositor element is included in the new SLA. Finally we have included an inconsistency in the definition of the value of the `InitCost` property. The new SLA is shown in Figure 6.

After mapping the example of Figure 6 to the equivalent CSP ψ_α, our proof-of-concept processes the explanation problem and returns a minimal subset of inconsistent constraints of the ψ_α. Then, it traces these constraints to the inconsistent agreement terms. For this example, the proof-of-concept consistency checker returns the following subset *[CostLET15, CostDefinition, SDTInitCost]*, because the `InitCost` definition inside the `ServiceDescriptionTerm` is inconsistent according to the `CostDefinition` and `CostLET15` terms. If the user solves the inconsistency of these terms, the checker would check again whether the new agreement document is consistent or not and it would return the minimal subset of inconsistent constraints in the second case.

6 Related Work

As far as we know, there are no proposals that deal with providing explanations for the SLA inconsistencies of WS-Agreement documents. Previously, in [6], we proposed mapping SLAs to CSPs, aimed at checking their consistency and conformance. However, in that paper no explanation about the inconsistency of the terms was provided. In addition, [6] dealt with its own SLA specification instead of using a standard format such as WS-Agreement.

Other similar work is [5], in which Oldham et al. create a description logic-based ontology of WS-Agreement that could be used to check consistency and conformance of SLAs using a description logic reasoner. However, the authors do not detail what the consistency checking process is. Furthermore, they do not support the explanations for the inconsistent terms.

```
<Agreement ‘‘id=simplifiedExampleScenario’’>

     <Context>   ...    </...>

     <All>
          <ServiceDescriptionTerm Name=’’ComputingService’’>
               <...>
                    ...
                    <InitCost>      20   </...>
                    <Availability>  95   </...>
                    <Increment>     15   </...>
                    ...
               <...>
          </ServiceDescriptionTerm>

          <ServiceProperties>
          ...
               <Variable name=’’Availability’’ metric=’’metricXML:Availability’’>
                    <Location> \\Availability </Location>
               </Variable>
               <Variable name=’’Cost’’ metric=’’metricXML:Cost’’>
                    <Location> \\Cost </Location>
               </Variable>
               <Variable name=’’Increment’’ metric=’’metricXML:Increment’’>
                    <Location> \\Increment </Location>
               </Variable>
               <Variable name=’’InitCost’’ metric=’’metricXML:InitCost’’>
                    <Location> \\InitCost </Location>
               </Variable>
          ...
          </ServiceProperties>

          <GuaranteeTerm Name=’’CostDefinition’’>
               <SLO>   Cost = InitCost + Increment  </...>
          </GuaranteeTerm>

          <GuaranteeTerm Name=’’InitCostLET15’’>
               <SLO>   InitCost <= 15  </...>
          </GuaranteeTerm>
          <GuaranteeTerm Name=’’CostGET20’’>
               <SLO>   Cost >= 20  </...>
          </GuaranteeTerm>

          <GuaranteeTerm Name=’’CostLET30’’>
               <SLO>   Cost <= 30  </...>
          </GuaranteeTerm>
      </All>

</Agreement>
```

Figure 6. Simplified example of a WS-Agreement* document for the proof-of-concept.

7 Conclusions and Future Work

In this paper we have motivated the need for explaining inconsistencies of WS-Agreement documents and we have presented a first approach to reach this goal. More specifically, we present the problem of explaining WS-Agreement inconsistencies as a constraint satisfaction problem (CSP), and then we use a CSP solver together with an explanation engine to check the consistency and return the inconsistent terms.

To this end, we have defined WS-Agreement*, which is a subset of WS-Agreement that limits the expressiveness of WS-Agreement, but still allows defining complex SLAs such as the one depicted in Figure 1. Then, we have detailed the mapping of WS-Agreement* to a CSP and we have described the steps that involve the process of checking and explaining WS-Agreement inconsistencies. Finally, we have presented a proof-of-concept implementation that uses the Choco solver and the Palm explanation engine to perform the explanation of SLA inconsistences on a simple example.

In summary, the main contributions of this paper are the following:

1. we define a subset of WS-Agreement that can be useful for implementations that do not require the full expressiveness of WS-Agreement;
2. we define a mapping of WS-Agreement* to CSPs that enables the use of CSP solvers to check the consistency of SLAs;
3. we describe a CSP solver-independent process to check and explain inconsistencies of SLAs.

However, there are still some open issues that require further research: first, extending the mapping to CSPs to full WS-Agreement; second, checking the consistency of WS-Agreement with the temporal extension we detailed in [4], and third, using the CSP solver to check not only the consistency, but also the conformance of an agreement offer with an agreement template.

References

1. Choco constraint solver web site, 2008. `http://choco-solver.net/`.
2. OGF Grid Resource Allocation Agreement Protocol WG (GRAAP-WG). Web Services Agreement Specification (WS-Agreement) (v. gfd.107), 2007.
3. Narendra Jussien and Vincent Barichard. The PaLM system: explanation-based constraint programming. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, pages 118–133, Singapore, September 2000.
4. C. Müller, O. Martín-Díaz, A. Ruiz-Cortés, M. Resinas, and P. Fernández. Improving Temporal-Awareness of WS-Agreement. In *Proc. of the 5^{th} International Conference on Service Oriented Computing (ICSOC)*, pages 193–206, Vienna, Austria, Sept 2007. Springer Verlag.
5. N. Oldham, K. Verma, A. Sheth, and F. Hakimpour. Semantic WS-Agreement Partner Selection. In *15^{th} International WWW Conf., 697–706.* ACM Press, 2006.

6. A. Ruiz-Cortés, O. Martín-Díaz, A. Durán, and M. Toro. Improving the Automatic Procurement of Web Services using Constraint Programming. *Int. Journal on Cooperative Information Systems*, 14(4), 2005.
7. T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *Tools with Artificial Intelligence, 1993. TAI '93. Proceedings., Fifth International Conference on*, pages 48–55, 8-11 Nov 1993.
8. E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1995.