A test suite for an Agreement Document Analyser (v 1.0)

Carlos Müller, Sergio Segura, A. Ruiz–Cortés

{cmuller,sergiosegura,aruiz}@us.es

Applied Software Engineering Research Group
University of Seville, Spain
March 2012

Technical Report ISA–12–TR–03

This report was prepared by the

Applied Software Engineering Research Group (ISA)
Department of computer languages and systems
Av/ Reina Mercedes S/N, 41012 Seville, Spain
http://www.isa.us.es/

Support: This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project SETI (TIN2009-07366) and by the Andalusian Government under projects ISABEL (TIC-2533) and THEOS (TIC-5906).

List of changes

Version	**Date**	**Description**
1.0	March 2012	testing of ADA without table of all test cases

Contents

Abstract

The automated analysis operations of WS–Agreement documents proposed in recent works [1, 2, 3, 4] uses complex techniques of constraint programming as well as auxiliary code for agreement documents handling (XML documents mappings to constraint satisfaction problems (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 detection 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 technical report, we present a test suite for an Agreement Document Analyser (ADA) framework. For the test suite design several popular techniques from the software testing community were used to develop a set of implementation–independent test cases to validate the functionality of the eight most significant ADA analysis operations. Specifically, we used three black-box testing techniques to design our test cases, namely: equivalence partitioning, pairwise testing, and error guessing. 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.

Chapter 1

Introduction

The automated analysis operations of WS–Agreement documents proposed in recent works [1, 2, 3, 4] uses complex techniques of constraint programming as well as auxiliary code for agreement documents handling (XML documents mappings to constraint satisfaction problems (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 detection 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 technical report, we present a test suite for an Agreement Document Analyser (ADA)[1] [5]. The test suite comprises a set of implementation–independent test cases to validate the functionality of the eight most significant ADA analysis operations.

For the test suite design we inspired in [6] in which popular techniques from the software testing community were used to develop a representative set of input-output combinations were used to test FAMA a feature model analysis tool. Specifically, we used three black-box testing techniques to design our test cases, namely: equivalence partitioning [7, 8], pairwise testing [9, 10], and error guessing [9]. 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 1 shows the general inputs and outputs for each ADA analysis operation under test[2].

[1]`http://www.isa.us.es/ada`

[2]for more information about the inputs and outputs see the wsdl interface at `http://www.isa.us.es:8081/ADAService?wsdl`

Table 1.1: Inputs and outputs for ADA analysis operations addressed in the suite

Operation ID	**ADA operation**	**Input**	**Output**
incs	inconsistencies detection	WS-Ag doc	Boolean value
incsExp	inconsistencies explanation	WS-Ag doc	Collection of (term, agreement elements)
deadTerms	dead terms detection	WS-Ag doc	Collection of terms
deadTermsExp	dead terms explanation	WS-Ag doc	Collection of (term, agreement elements)
ludTerms	ludicrous terms detection	WS-Ag doc	Collection of terms
ludTermsExp	ludicrous terms explanation	WS-Ag doc	Collection of (term, agreement elements)
compliance	compliance detection	Template, Offer	Boolean value
nonComplExp	non-compliance explanation	Template, Offer	Collection of (Offer term, conflicting template elements)

Chapter 2

Inputs selection

We found two testing techniques to be helpful for the selection of a suitable set of input WS–Agreement document, namely: equivalence partitioning and error guessing. Next, we explain these techniques and how we used them.

2.1 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) [7, 8]. 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 behavior of the program for the corresponding 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 [11]) 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.

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.

2.2 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 [9]. 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 using conflicting WS–Agreement documents including one or more conflicts inside as suitable inputs to check whether inconsistencies, dead terms, ludicrous terms, and non-compliance are correctly detected 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

[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.

```
Template – Translate it! (1.0)
  Context:
   Responder: Translator, as ServiceProvider
All
  SDT1: Service Description for TranslationService //by default values
    InputFile =                                    //modifiable in offers
    SourceLang =
    TargetLang =
    InputFileSize = 10

  SP1: Service Property for TranslationService
    TranslationTime – measured by metric:Time – related to SDT1
    //Description: % of source text typos
    InputErrors – measured by metric:Percent – related to SDT1/InputFile

  ...

  GTInputFileSize: Guaranteed by ServiceConsumer:
   SLO: InputFileSize < 10

  GTInputFileSizeQC: Guaranteed by ServiceProvider
   QC: NOT ImageTranslation
   SLO: InputFileSize > 10

  ...

Creation Constraints:
  Items:
    ...

  Constraints:
    ...
    MinSize: InputFileSize < 30 => NOT ImageTranslation
```

Figure 2.1: Template got from error guessing with an inconsistency between three agreement elements that make conflicting the document

and 7 agreement offers for conflicts of one WS–Agreement document; and 7 template-offer pairs for compliance conflicts.

For instance the template of Fig. 2.1 was included to check if ADA realized about conflicts involving several agreement elements. Specifically, the template has an inconsistency between `GTInputFileSize` and `GTInputFileSizeQC` by `MinSize` constraint. The reason for that inconsistency is as follows: as the `GTInputFileSize` SLO makes the `MinSize` implication antecedent be fulfilled, the implication consequent is taken into account. Such a consequent satisfies the qualifying condition of `GTInputFileSizeQC` and so its SLO is enabled being inconsistent with `GTInputFileSize`. Note that although `SDT1` description term includes the inconsistent constraint `InputFileSize=10` it is not considered because service description terms values in templates are simply by default values.

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:

1. 7 pairs of templates with different terms, but fixed context and creation constraints; against

C. Müller *et al.* Seville, March 2012

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.

Chapter 3

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.

3.1 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 [9, 10]. 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).

Chapter 4

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 4 shows. For their specification, we followed the guidelines of the IEEE Standard for Software Testing Documentation [12]. As an example, Table 4.2 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 dependencies refer to the identifiers of test cases that must be executed prior to a given test case. The four samples included in Table 4.2 requires the previous execution of a detection 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 an item of the template. Note that test case *IC-105* should be executed beforehand to assure that there exist a non-compliance conflict in such an input. Moreover, *IC-105* requires a previous execution of *CDC-13*, and *CDC-24* test cases to assure each input were conflict-free, and so on.

Table 4.1: Number of inputs got from each testing technique for the ADA analysis operations

Operation ID	Inputs by equivalence partitioning	Inputs by error guessing	Inputs by pairwise	Total test cases
incs	31	18	81	130
incsExp				130
deadTerms		18		130
deadTermsExp				130
ludTerms		18		130
ludTermsExp				130
compliance	0	7+20	120	147
nonComplExp				147
Total	each number is a different set of inputs			**1074**

Table 4.2: Four of the test cases included in the suite

TestCase ID	Description	Input	Expected Output	Deps
EI-31	Detecting errors while explaining inconsistencies between a guarantee term of a nested term compositor with a service description term	Agreement Offer with OneOrMore term compositor with a nested ExactlyOne of 2 guarantee terms	SDT1_Image-Translation inconsistent by GTTranslationTime2	I-31
EDT-124	Detecting errors while explaining the origin of two dead terms by the use of a contradictory predicates in qualifying conditions	Agreement Offer with two guarantee terms with contradictory qualifying conditions	GTInputErrors1 is dead by GTInputErrors1; GTInputErrors2 is dead by GTInputErrors2	DT-124
ELT-18	Detecting errors while explaining the origin of two ludicrous terms by the same item od creation constraints	Template with 2 items, a general constraint with an integer predicate, and a another constraint with an string predicate	GTTranslationTime1 is ludicrous by Item_TranslationTime; GTTranslationTime2 is ludicrous by Item_TranslationTime	LT-18
IC-105	Detecting errors while checking the compliance between a guarantee term of the offer and an item of the template creation constraints	Template with a service description term and 2 items; and an Offer with guarantee term with qualifying condition	The Agreement Offer is non-compliant with the Template	CDC-13, CDC-24
ENC-105	Detecting errors while explaining the origin of non-compliance between a guarantee term of the offer and an item of the template creation constraints	Template with a service description term and 2 items; and an Offer with guarantee term with qualifying condition	GTTranslationTime1 of the Offer is non-compliant with Item_TranslationTime of the Template	IC-105

Chapter 5

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 [13] 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.

5.1 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 with an adequate exception treatment.

5.2 Errors detected while analysing documents.

We found several errors while analysing WS–Agreement documents with specific combinations of agreement elements[1]:

- Attribute-value pairs of service description terms in templates are by default values but it were considered as value assignments.

[1]Errors found while analysing IsCompliant and ExplainNonCompliance operations are not included here yet.

- In WS–Agreement documents with variants inside, dead and ludicrous terms were returned by `deadTerms` and `ludTerms` operations in the same number of variants that include them.
- When `deadTerms` or `ludTerms` operation were used with a conflicting document a Null-PointerException were returned.
- When a term were dead or ludicrous by an attribute-value pair assignment of a service description term, the responsible value assignment were not explained by `deadTermsExp` or `ludTermsExp` operation.
- When the cause of a term to be dead or ludicrous were not found in a variant, the term were classified as non dead or ludicrous by `deadTermsExp` or `ludTermsExp` operation before studying the other variants.
- When several dead or ludicrous 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 `deadTerms` or `ludTerms` operation.
- When the cause of a term to be dead or ludicrous is itself, the explanation were not correctly returned by `deadTermsExp` or `ludTermsExp` operation.

Acknowledgments

We would like to thank Antonio Jurado, as member of our technical staff for his work and dedication to this ADA test suite.

Bibliography

[1] Martín-Díaz O, Ruiz-Cortés A, Durán A, Müller C. An Approach to Temporal-Aware Procurement of Web Services. In: Proc. of The 3rd International Conference on Service-Oriented Computing (ICSOC); 2005. p. 170–184.

[2] Müller C, Ruiz-Cortés A, Resinas M. An Initial Approach to Explaining SLA Inconsistencies. In: Proc. of the 6th Int. Conf. on Service-Oriented Computing (ICSOC). vol. 5364 of LNCS. Sydney, Australia: Springer Verlag; 2008. p. 394–406.

[3] Müller C, Resinas M, Ruiz-Cortés A. Explaining the Non-Compliance between Templates and Agreement Offers in WS-Agreement*. In: Proc. of the 7th International Conference on Service-Oriented Computing (ICSOC). vol. 5900 of LNCS. Sweden, Stockholm: Springer Verlag; 2009. p. 237–252.

[4] Müller C, Resinas M, Ruiz-Cortés A. Conflict Taxonomy and Explanation Mechanisms for WS–Agreement Documents. Submitted to ACM Transactions on the Web (TWEB). 2011;.

[5] Müller C, Resinas M, Ruiz-Cortés A. A Framework to Analyse WS–Agreement Documents. In: Proc. of The 4th Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC'10). Ayia Napa, Cyprus: Springer Verlag; 2010. .

[6] Segura S, Benavides D, Ruiz-Cortés A. Functional Testing of Feature Model Analysis Tools: A Test Suite. IET Software. 2011;5:70–82.

[7] Myers GJ, Sandler C. The art of software testing. Wiley & Sons; 2004.

[8] Pressman RS. Software engineering: a practitioner's approach. McGrap-Hill; 2001.

[9] Copeland L. A practitioner's guide to software test design. Artech House, Inc., Norwood, MA, USA; 2003.

[10] Grindal M, Offutt J, Andler SF. Combination testing strategies: a survey. Software Testing, Verification and Reliability. 2005;15(3):167–199. Available from: `http://dx.doi.org/10.1002/stvr.319`.

[11] Andrieux et al . Web Services Agreement Specification (WS-Agreement) (v. GFD.107); 2007. OGF - Grid Resource Allocation Agreement Protocol WG.

[12] Draft IEEE Standard for software and system test documentation (Revision of IEEE 829-1998); 2007. Available from: `http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=4432350`.

[13] DeMillo RA, Lipton RJ, Sayward FG. Hints on Test Data Selection: Help for the Practicing Programmer. Computer. 1978 april;11(4):34 –41.