Research Note
RN/11/17

Generating Realistic Test Input Using Web Services

26 July 2011

Mustafa Bozkurt

Mark Harman

Abstract

Generating realistic test data is a major problem for software testers. Realistic test data generation for certain input types is hard to automate and therefore laborious. We propose a novel automated solution to test data generation that exploits existing web services as sources of realistic test data. Our approach is capable of generating realistic test data and also generating data based on tester-specified constraints. In experimental analysis, our prototype tool achieved between 93% and 100% success rates in generating realistic data using service compositions while random test data generation achieved only between 2% and 34%.

RN/11/17
26 July 2011

Research
Note

Generating Realistic Test Input Using Web Services

Mustafa Bozkurt and Mark Harman

Telephone: +44 (0)20 7679 0332
Fax: +44 (0)171 387 1397
Electronic Mail: {m.bozkurt, m.harman}@cs.ucl.ac.uk
URL: http://www.cs.ucl.ac.uk/staff/M.Bozkurt/,
http://www.cs.ucl.ac.uk/staff/M.Harman/

Abstract

Generating realistic test data is a major problem for software testers. Realistic test data generation for certain input types is hard to automate and therefore laborious. We propose a novel automated solution to test data generation that exploits existing web services as sources of realistic test data. Our approach is capable of generating realistic test data and also generating data based on tester-specified constraints. In experimental analysis, our prototype tool achieved between 93% and 100% success rates in generating realistic data using service compositions while random test data generation achieved only between 2% and 34%.

Keywords

web service testing, service-oriented test data generation, realistic test data

Department of Computer Science
University College London
Gower Street
London WC1E 6BT, UK

1 Introduction

The topic of testing web services gaining momentum. Evidence for this momentum can be found in the number of papers published on this subject, depicted in Figure 1. This figure presents the total number of research papers published on web service testing between 2002 and 2010 [9]. As can be seen from the figure, there is a good fit to a quadratic growth trend.

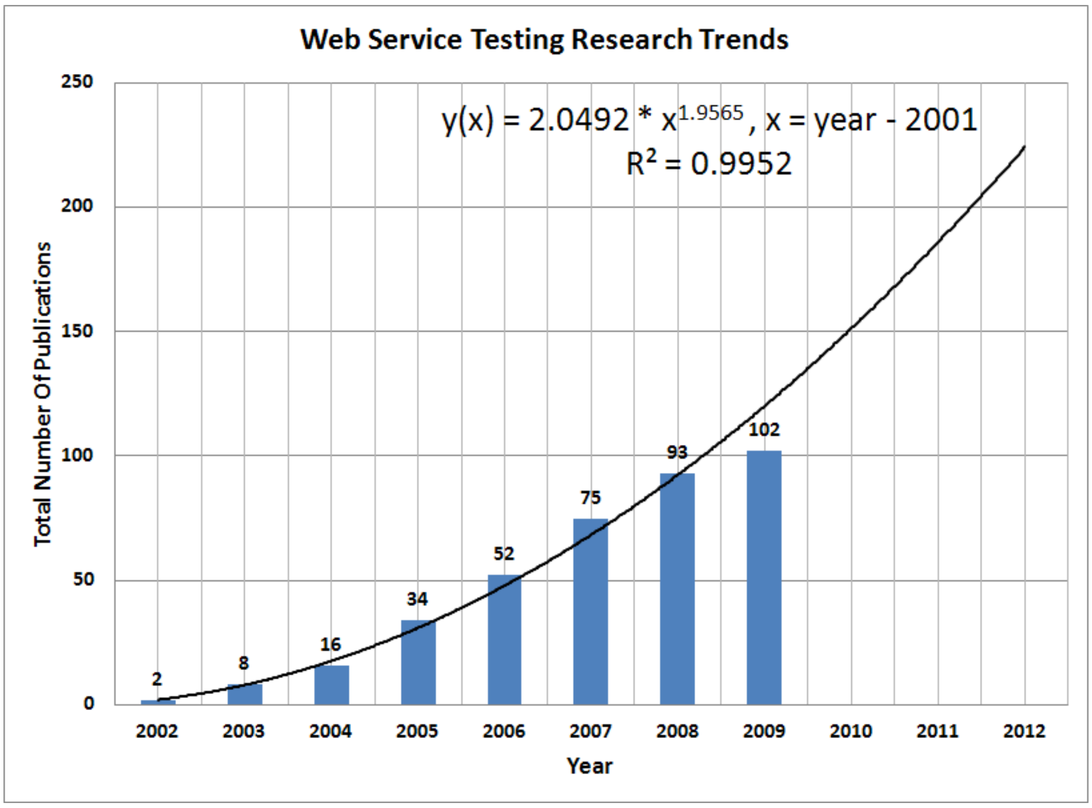


FIGURE 1: Growth in papers on web service testing [9]

A recent survey by Bozkurt et al. [8] summarised the issues regarding testing of web services and highlighted the reasons why service-centric systems require more effective and efficient testing/monitoring than traditional software systems.

According to the literature, one of the most expensive activities in a software project is the construction of test data [14]. This problem might be more severe in environments like Service Oriented Architecture (SOA) where testing frequency is high, binding times can be late and services composed of systems provided by many different third parties. One way of reducing this cost is automation of the test data generation process.

Unfortunately, automated test data generation for most online systems is not as straightforward as it is for traditional systems. The problems in test data generation for online systems are mainly caused by the type of required test data: most online systems require data that cannot be effectively generated by traditional automated test data generation algorithms. For example, the existing semantic web service (SWS) examples from Mindswap [20] require realistic test data, such as ISBNs, United States Zone Improvement Plan (ZIP) codes and English and French words. It is a considerable challenge for any automated test data generation technique to generate realistic test data, not merely 'structurally valid' test data. Realistic data must not only be correctly formed, according to structural rules (which can be automated), it must also have a semantics that ties it to real-world entities. This latter property poses challenges to automated testing. More precisely, realistic test data, in this context, is realistic in two respects:

1. Structural Validity: Realistic data must conform to syntactic constraints. For example, a structurally

valid ISBN consists of 10 digits $x_1, ..., x_{10}$, such that:

$$x_{10} = 11 - (10x_1 + 9x_2 + ... + 2x_9)\ mod\ 10.$$

(This algorithm will be referred as check-digit algorithm in the rest of the present paper.)

2. Semantic validity: Realistic test data must represent a real-world entity. For example, a realistic ISBN must correspond to a real-world book not merely a structurally valid ISBN.

Traditional automated testing can generate tests that achieve coverage, while manual testing can generate realistic test cases. The former is efficient yet unrealistic, while the latter is laborious yet realistic. What is needed is a technique that is both realistic *and* automated.

In this paper we introduce a novel service-oriented test data generation approach that addresses this problem of effectiveness in automated generation of realistic test data. In our approach, the need for previous data and manual tester input are minimized by leveraging the data that can be acquired from compositions of many existing web services.

The advantages of the proposed approach are:

1. **Automated**: Improved effectiveness in automated test data generation for the datatypes that can not be effectively generated.
2. **Tailored**: Test data generation/selection based on the tester criteria and optimized test data generation based on the data source.
3. **Applicability**: Ability to generate test data to test any system with semantic information.
4. **Minimized**: Minimal dependence on the existing data sources such as databases and session data.

We named our tool ATAM by combining the initial letters of these four advantages.

The primary contributions of this paper as follows:

1. We introduce an automated test data generation approach capable of generating realistic test data effectively.
2. We describe a prototype tool that supports our test data generation approach.
3. We present the results from two real-world case studies to evaluate our approach against the most readily available alternative for automated service testing.

The rest of this paper is organized as follows. Section 2 summarises existing research on test data generation for web services and approaches similar to the proposed approach in other testing application domains. Section 3 discusses the importance of realistic test data. Section 4 explains data categorisation for our approach. Section 5 introduces our approach in more detail. Section 6 briefly overviews our implementation and technical issues. Section 7 presents the case studies used in the experiments. Section 8 describes our experimental methodology and presents discussion on the results from the experiments. Section 9 presents future work and concludes the paper.

2 Background

This section is divided into two categories: test data generation for web services and test data generation approaches that reuse existing resources.

2.1 Test Data Generation for Web Services

Test case generation for web services is usually based on the web service specification. Traditional web services provide specifications that include abstract information on the available operations and their parameters. Information from these specifications can be used to support test data generation for black-box testing techniques.

Existing WSDL-based test data generation approaches [4, 6, 7, 13, 16, 18, 21, 23] depend on XML Schema datatype information. The datatype information, which defines various constraints for each datatype, allows data generation for each simple datatype. Since it is possible to define complex datatypes, test data generation for these datatypes simply requires decomposition of the complex type into simple datatypes and application of data generation for each simple datatype. This approach can generate structurally valid test inputs, but may not generate *realistic* test inputs.

Fault-based test data generation tests for prescribed faults in web services. Unlike other testing approaches, in fault-based test data generation erroneous test data is generated intentionally. Proposed fault-based testing approaches [13, 21, 27, 28, 31, 32] generate test data from communications among services by inserting erroneous data. The focus of fault-based testing is often the generation of tests able to reveal the fault in question. However, there is no guarantee that the tests so-generated will be realistic.

Test data generation from a WSDL definition is limited to generating structurally valid test data but this cannot be guaranteed to be realistic because WSDL contains no behavioural information about the service under test. This limitation is reduced with the introduction of semantic web services. The use of semantic models such as OWL-S for test data generation is proposed [5, 11, 27, 29] not only because of the behavioural information they provide but also because of the semantic information on the datatypes. This semantic information (in the form of an ontology) allows ontology-based test data generation. However, this semantic information does not necessarily code *realism*, nor can it always guarantee to do this.

2.2 Test Data Generation Approaches Using Existing Resources

Test cases can be generated using existing data, such as existing test cases and recorded user session data. The use of session data for web application testing has been suggested by several researchers [2, 12, 17].

Thummalapenta et al. [26] propose an approach that aids test generation approaches in reaching desired states. The proposed approach mines code bases and extracts sequences that are related to the object types of the method under test.

However, the only approach that aims to generate test data for service-oriented systems was proposed by Conroy et al. [10]. In this approach test cases are generated using the data from applications with Graphical User Interfaces (GUIs). The approach harnesses data from GUI elements and uses the harnessed data to generate test cases for service-oriented systems. Compared to approaches presented in Section 2.1 this approach might expected to be more effective in generating realistic test data. Unfortunately, however, the proposed approach remains unautomated.

Mcminn et al. [19] also proposed automatic generation of realistic test data, aiming to reduce the human oracle cost. The authors propose the use of genetic algorithms combined with data and input domain knowledge to increase the fault-finding capability and the branch coverage probability of the generated test data. The approach also includes re-using of test data among related units of the system under test. Even though this approach is capable of generating 'realistic' test data as with most of the automated approaches it can only provide structural realism.

Alshahwan and Harman [3] use search based testing to generate branch adequate test data for server side code in PHP. Their approach was shown to be effective at branch coverage. The proposed approach could be used to seed our approach for MID test data as a base case. This remains a topic for future work.

3 Why Is Realistic Test Data Important?

Realism is very important in testing for at least the following three reasons:

1) **Faults found by realistic test cases are more likely to be prioritised for fixing**: A test case that reveals a fault but uses special values that seldom or never occur in practice is unlikely to attract the attention of a tester who is preoccupied with many other such 'bug reports' from the field. Finding a test input that convinces the tester of a problem is important. If an automated tool generates many unusual and peculiar inputs that a user is unlikely to use then the tester will soon abandon the tool or disregard its findings.

2) **Realistic Test Inputs May Denote Important 'Corner Cases'**: There is a role to be played in searching for corner cases that are, in some sense, atypical (and therefore *less* realistic). Such cases as invalid data values are important to test the robustness of the system under test. However, realistic test cases can often denote a kind of 'corner case' too. Suppose there is an input that requires a domain specific value, selected from a narrow potential range of values that lie within a wider input type. For instance a ZIP code consists of a five digit number, so there are potentially 10^5 correctly typed candidates. However, only very few of these candidate five-digit numbers will correspond to a location in Huntsville, Alabama (these are the 15 codes from 35801 to 35816). Suppose a large part of the system under test is concerned with this particular geographic region and there is a predicate that checks, early on in any user session, whether the address is valid. All code beyond such a predicate test will be inaccessible to (and therefore untested by) any automated testing approach that is unable to focus on the generation of valid Huntsville ZIP codes. We believe that these situations are far from atypical, particularly in service-oriented systems, where the services provided carry with them a great deal of inherent domain-specific properties. By generating realistic test data we are simultaneously providing a mechanism for addressing the problem of generating domain specific data.

3) **Realism is Important for Testing Service Compositions**: When two services are composed, the interface through which they communicate is likely to specialise the kinds of values that may pass between the services. For instance, a generic train booking service might be composed with a service offering a specific holiday package. Suppose that the holiday package may offer only a very limited set of possible train bookings. This restriction creates another kind of 'realistic value'. That is, though the train service must be tested for any possible seat on any train, on any line, at any time and date, the composition that provides the holiday package requires a specific set of seats on certain trains at restricted times. In generating test cases for such a service composition, a tool that is able to generate realistic test cases for this scenario will only generate correctly restricted journeys. For the same number of test cases, such a tool will be able test the many of the various aspects of this restricted functionality intensively, while a more general tool will simply repeatedly test the 'invalid journey' functionality.

4 Data Categorisation

In our approach, each input type lies in one of three categories:

1. *Method Independent Data (MID)* is the type of data which can be systematically generated. In order to be identified as MID, a data generation method for this datatype must be known to ATAM prior to the test data generation process. MID data can be classified into two groups based on the structural validity criterion:

    Formed data is a type of data where a random but structurally valid instance of the data can be generated.

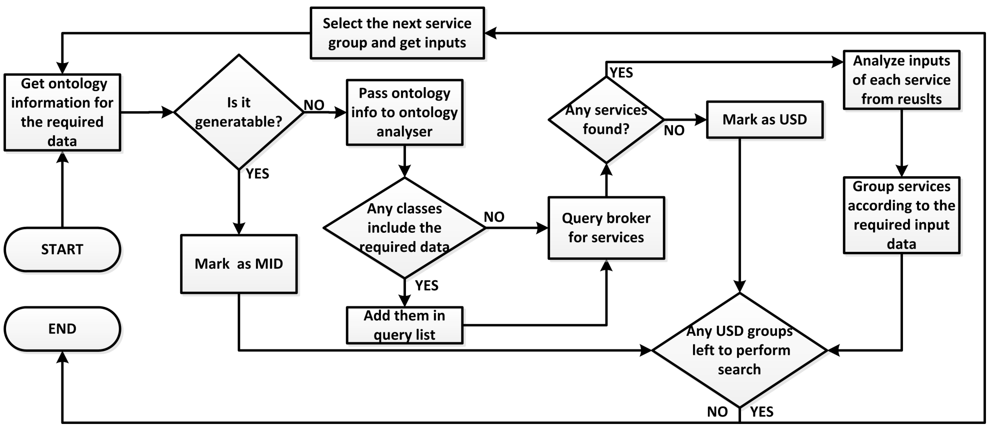
   An example of this category is an ISBN generated using the check-digit algorithm.

    Non-formed data is randomly generated data that lies within a supertype for which even structural validity cannot be guaranteed.

An example of this category is an 'ISBN' constructed merely by generating 10 random digits; the set of all 10 digit numbers is a supertype of the ISBN type.

2. *Data Originated from a Service (DOS)* is the type of data that can be attained from an existing web service.

3. *Tester Specified Data (TSD)* is the type of data that needs to be provided by the tester (such as a login and password). When our tool requires an input data that cannot be generated, it requests assistance from the tester. All non-MID data is initially considered to be TSD data. However, for TSD data that can be provided by a web service composition becomes DOS data within our test data generation process. Our goal is to maximally automate the production of realistic test data by migrating TSD to DOS.



Select the next service group and get inputs
Get ontology information for the required data
Is it generatable?
NO
YES
Pass ontology info to ontology analyser
Any classes include the required data
NO
YES
Query broker for services
Any services found?
YES
NO
Mark as USD
Analyze inputs of each service from reuslts
Group services according to the required input data
START
END
Mark as MID
Add them in query list
Any USD groups left to perform search
NO
YES

FIGURE 2: Flow graph of the overall search process that seeks realistic test data from composition of other services

5 Realistic Test Data Generation

To address this issue of realism in automated test data generation of services, we propose to use services themselves as the solution to the problem. We shall seek compositions of services that are able to build realistic test cases. If Service A is not able to generate the required realistic value then we will seek to compose it with Service B, the output of which, when fed to service A does result in the desired set of possible realistic values. For example, if we have a service that can produce an ISBN given an author and a service that can generate and author name given a keyword, then the composition can generate an ISBN from a keyword. When further composed with a dictionary service, we obtain a service sequence that can produce a valid ISBN with little or no input. In this way, we seek arbitrary service compositions that will progressively build the realism we seek into the test data we seek.

Our approach is limited to semantic systems. The reason for this limitation drives from the need for semantic information concerning input and output messages which are core elements of our approach. Semantic information for messages, not only allows us to automatically discover existing web services with required data, but also allows automatically discover relations among different data structures that might contain the required test data.

ATAM is intended to be used with all semantic systems. As a result, it uses two different types of initial user inputs. If the service/system under test (SUT) is a SWS the tester can provide a semantic service description such as OWL-S description. For any other system, the tester has to provide a semantic description of the required input data in the form of an ontology class. The overall flow graph of our search, analyse and compare process implemented by ATAM, is depicted in Figure 2.

The information on test data to be generated is passed to the ontology analyser to search for possible relations defined within the specified ontologies.

5.1 Ontology Analyser

Automated service discovery is one of the main topics of SOA. In our approach we benefit from the ability provided by SOA to automatically discover services that provide the required test data.

There are several existing tools that provide subsumption based service discovery/matchmaking. Subsumption relation-based matchmaking is capable of finding super or subclass of the provided ontology class. For example, if we take an ontology where *book class* is defined as a subclass of *ISBN class* and try to discover services using this ontology. In this scenario, it is possible to discover the services (among the web services using this ontology) that provide book information while searching for services that provide ISBN.

Discovering the relations between service messages, such as ISBN and book, is one of the most important requirements that needs to be facilitated in our approach. Unfortunately, this direct relation might not be specified in the ontology. We analysed many of the available ontologies that include a definition of ISBN in Swoogle [25], a popular ontology database, and found out that none of the existing ontologies with an ISBN definition contains this relation. Interestingly, in all except one ontology, 'ISBN' is defined as a *DatatypeProperty*, a property used in defining relation between instances of an ontology class and RDF literals and XML Schema datatypes, of book class.

In the light of the results from our analysis, we introduced an ontology analyser that discovers relations between ontology entities. The ontology analyser currently discovers two different possible relations among classes and properties. The discovered relations are, as mentioned earlier, the relation between a DatatypeProperty and its defining classes and relation between two separate classes defined using a DatatypeProperty. The ontology analyser is not only able to process a single ontology, but it is also able to process ontologies where relations to external classes are specified as well.

5.2 Service Discovery

After the ontology analysis stage, ATAM queries the service broker(s) for services that provide any of the discovered ontology classes as output. Search queries used in the experiments include only an output specification.

In order to achieve this functionality, the generated search queries use the Ontology Web Language (OWL) 'Thing' class, a built-in class 'global base class' that denotes superclass of all classes. The reason for using 'Thing' class is to find all available web services with the given output message. Using this class as input with a subclass based subsumption criteria on inputs guarantees a service matching regardless of the service inputs.

5.3 Search Result Processing

After obtaining the search results, services from the results need to be analysed for their inputs. This is to identify whether they require MID or TSD type input data, as explained in Section 4. After the analysis, services with TSD type inputs go trough an additional iteration of the discovery process in order to investigate whether there exist services that provide inputs of this service.

As mentioned in Section 4, some of the TSD can be provided by other web services. A TSD service with an input that can be provided by another service becomes a DOS service. When a web service proving the data for another TSD service is found, these two services are considered to form a 'service sequence'. In service sequences, the service providing the data is considered as higher level than the service that requires the data.

A search result might contain many TSD services with the same functionality. In such a situation, performing the discovery process for all TSD services involves running the same query several times. This is an unwanted side effect that increases the time and cost of service discovery.

In order to avoid this potential overhead, a grouping stage is introduced. During this stage, web services from search results are grouped based on their inputs. Services with same input(s) are grouped together and a single service from each category is used to represent the group. As a result, all the services in a group require a single query for each input data they need.

During the grouping process services also go through an elimination process. At present, we introduced the following two natural elimination criteria:

1. Services that require the same input data as SUT are eliminated.
2. Services that already exist in lower levels in the existing sequences are eliminated.

5.4 Services with multiple inputs

Using services that require multiple inputs is more expensive than those that require a single input. Of course, it may not be possible to avoid services with multiple inputs so we also considered ways to reduce the cost of using these services.

One obvious solution is to reduce the number of services that provide the required inputs. There might be situations where several of the required inputs of a service can be provided by a single web service. In such situations, it is possible to reduce the execution time of the test data generation process by reducing the number of search queries and service invocations. In order to achieve this goal a 'partition stage' is introduced.

In this stage, the ontology analyser finds classes that contain the required data for each input as normal. After the initial analysis the ontology analyser examines the resulting classes for the possibility of containing an input other than that for which the search was conducted. If an ontology class containing more than one TSD input is found, it is marked as a combined solution for all the inputs it contains. As a result, during the next search iteration only one search query is used on combined solutions rather than the number of inputs they provide.

This functionality is only partly developed in the present version of ATAM. Its extension and other efficiency-improvement technique remain topics for future work.

5.5 Test Data Generation

After the completion of the search process, ATAM presents the tester with a graphical view of the possible sequences in an interactive GUI as shown in Figure 5. This window provides the tester with the group view of the sequences and it also allows the tester to see the details of each web service in each group.

In the GUI, nodes with green colour represent MID service groups which can be automatically generated by ATAM. The red nodes represent TSD and DOS services. The node that all sequences are connected to represents the searched data.

The tester starts the data generation process from the node at the end of one sequence. This might be the case for sequences that end with a MID group. On the other hand, for sequences that end with a TSD service the tester might want to start the process from a different group. ATAM allows the tester to start the test data generation process at any point in a sequence.

If the tester decides to start from a TSD group, ATAM asks the tester to provide the input data for the starting group in order to invoke a service from the selected group. Similarly, if the tester selects a DOS group then this group is considered to be a TSD group and the tester is asked for inputs.

This feature handles those cases where no MID services are found. It provides a 'basecase', allowing the tester to start test data generation process from a service group where he/she has knowledge on the required input data.

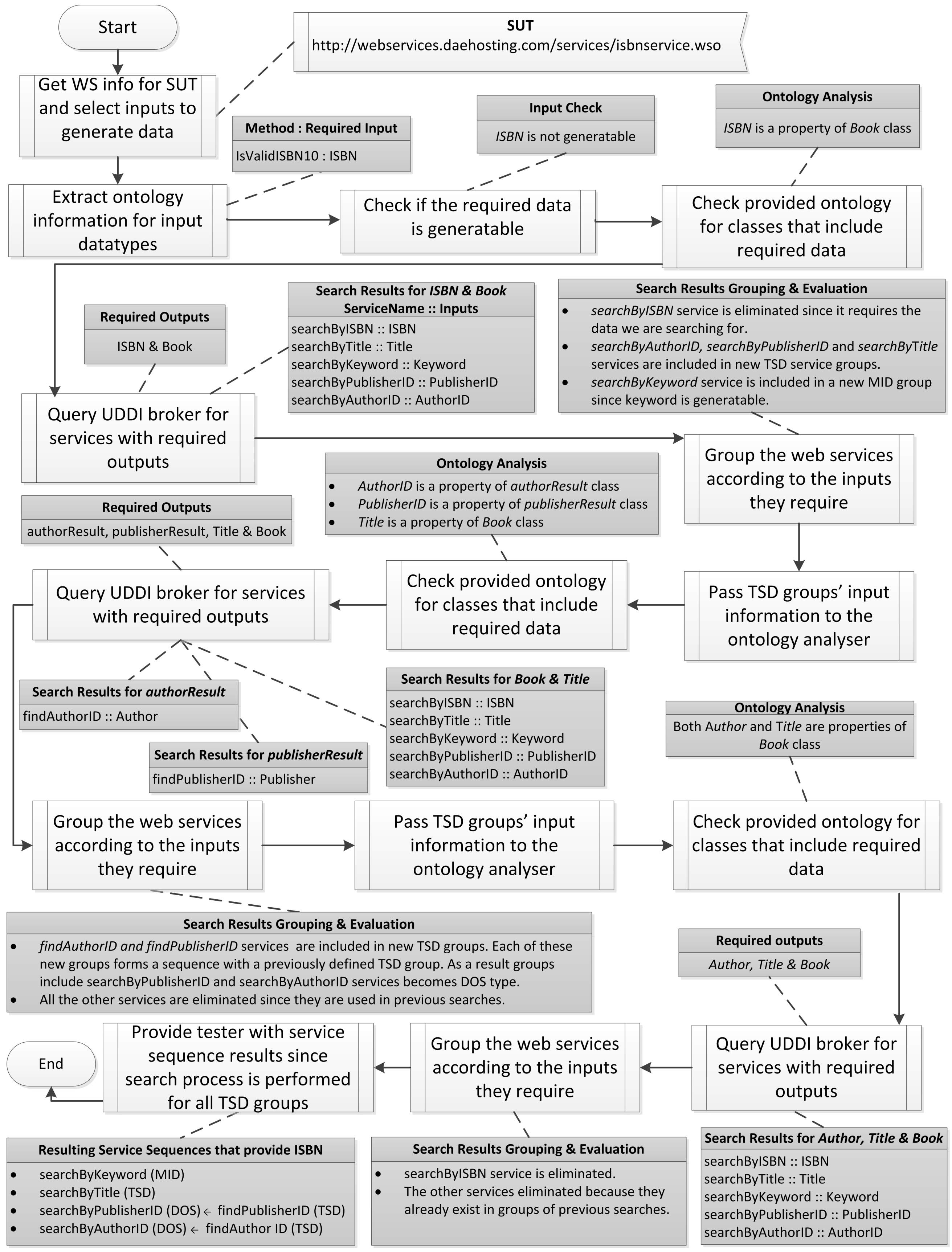

Start
Get WS info for SUT and select inputs to generate data
SUT
http://webservices.daehosting.com/services/isbnservice.wso
Method : Required Input
IsValidISBN10 : ISBN
Input Check
ISBN is not generatable
Ontology Analysis
ISBN is a property of Book class
Extract ontology information for input datatypes
Check if the required data is generatable
Check provided ontology for classes that include required data
Required Outputs
ISBN & Book
Search Results for ISBN & Book
ServiceName :: Inputs
searchByISBN :: ISBN
searchByTitle :: Title
searchByKeyword :: Keyword
searchByPublisherID :: PublisherID
searchByAuthorID :: AuthorID
Search Results Grouping & Evaluation
searchByISBN service is eliminated since it requires the data we are searching for.
searchByAuthorID, searchByPublisherID and searchByTitle services are included in new TSD service groups.
searchByKeyword service is included in a new MID group since keyword is generatable.
Query UDDI broker for services with required outputs
Group the web services according to the inputs they require
Ontology Analysis
AuthorID is a property of authorResult class
PublisherID is a property of publisherResult class
Title is a property of Book class
Required Outputs
authorResult, publisherResult, Title & Book
Query UDDI broker for services with required outputs
Check provided ontology for classes that include required data
Pass TSD groups' input information to the ontology analyser
Search Results for authorResult
findAuthorID :: Author
Search Results for publisherResult
findPublisherID :: Publisher
Search Results for Book & Title
searchByISBN :: ISBN
searchByTitle :: Title
searchByKeyword :: Keyword
searchByPublisherID :: PublisherID
searchByAuthorID :: AuthorID
Ontology Analysis
Both Author and Title are properties of Book class
Group the web services according to the inputs they require
Pass TSD groups' input information to the ontology analyser
Check provided ontology for classes that include required data
Search Results Grouping & Evaluation
findAuthorID and findPublisherID services are included in new TSD groups. Each of these new groups forms a sequence with a previously defined TSD group. As a result groups include searchByPublisherID and searchByAuthorID services becomes DOS type.
All the other services are eliminated since they are used in previous searches.
Required outputs
Author, Title & Book
End
Provide tester with service sequence results since search process is performed for all TSD groups
Group the web services according to the inputs they require
Query UDDI broker for services with required outputs
Resulting Service Sequences that provide ISBN
searchByKeyword (MID)
searchByTitle (TSD)
searchByPublisherID (DOS) ← findPublisherID (TSD)
searchByAuthorID (DOS) ← findAuthor ID (TSD)
Search Results Grouping & Evaluation
searchByISBN service is eliminated.
The other services eliminated because they already exist in groups of previous searches.
Search Results for Author, Title & Book
searchByISBN :: ISBN
searchByTitle :: Title
searchByKeyword :: Keyword
searchByPublisherID :: PublisherID
searchByAuthorID :: AuthorID

FIGURE 3: Complete search process for the first Case Study (CS1), explaining each step of the service sequence generation in detail. The search process for CS1 is important because it illustrates the effectiveness of our grouping mechanism in finding the shortest sequences. The whole search process presented in this figure is automated. As a result, ATAM considerably reduces the manual effort required for service-based realistic test data generation.

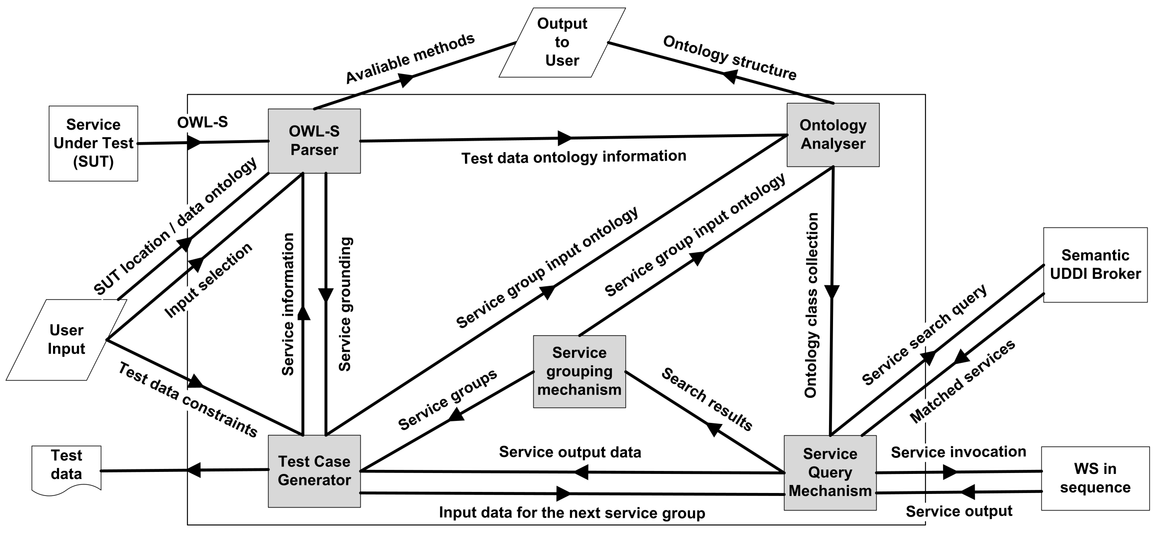


FIGURE 4: Overall architecture of ATAM

5.6 *Tailored Test Data Generation*

When the tester selects a service group to start, ATAM asks the tester for constraints. Constraints are not essential but they do allow tailoring of test data. Using constraints, the tester can refine the outputs of each service group, thereby guiding the test data generation process towards the requirements of the SUT.

To eliminate outputs that do not satisfy the constraints, we use XPath queries in ontology class instances. These queries are provided by the service developer within the *xsltTransformationString* element of the OWL-S grounding. As a result, the tester is not expected to provide a complete query, merely the constraint part of the XPath query such as “contains(hasAuthor,‘rowling’)” or “hasPublisher = ‘Springer’”.

The ontology analyser helps the tester to build constraints by providing the structures of ontologies. The tester is provided with this information through a constraint input window.

6 Implementation Details

The selection of semantic service specifications is based on expedient of tool availability. That is, we choose the OWL-S semantic markup language is chosen due to availability of tools that support OWL-S. WSDL2OWLS [30] is used to automatically generate OWL-S templates from WSDL documents. JXML2OWL [15] is used to generate the transformation strings in the OWL-S grounding. WSDL2OWLS uses OWL-S version 1.1 specifications and generates OWL-S specifications. The architecture of ATAM based on the selected service specifications is depicted in Figure 4.

Even though all the OWL-S specifications used in our experiments are in OWL-S version 1.1 format, the OWL-S parsing mechanism uses the OWL-S API version 2. This was necessitated by a major technical problem faced with OWL-S API 1.1 which was the API’s inability to parse xsltTransformationString elements correctly. Our use of two different OWL-S versions creates issues such as grounding incompatibilities which we had to solve by modifying the OWL-S Grounding structure.

ATAM supports OWL Full; a version of OWL that allows mixing of RDF Schemata with OWL, and which enjoys a high degree of compatibility with RDF. Even though ATAM supports OWL Full, it does not support pure RDF ontologies due to constraint incompatibilities. However, despite these technical details and compatibility issues, we have been able to use ATAM to provide realistic test data, as our two case studies illustrate.

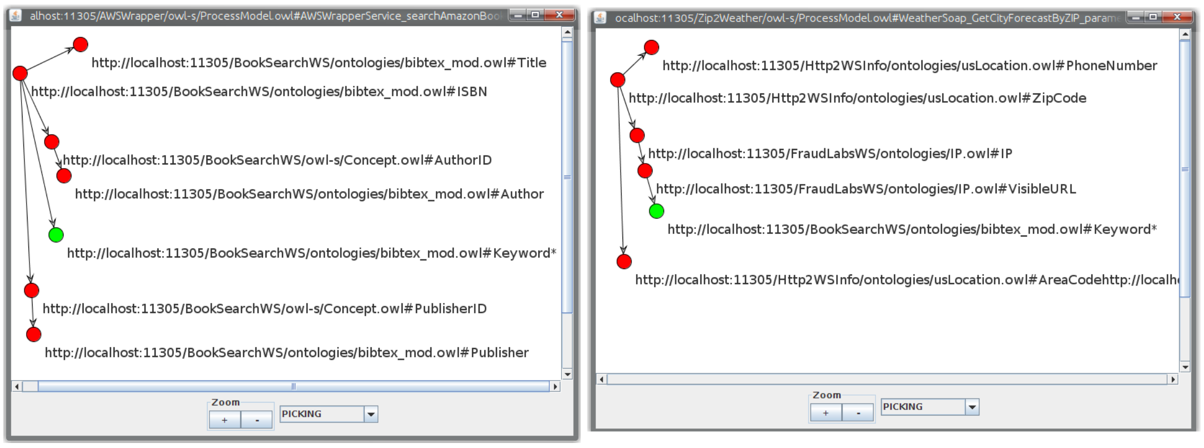

alhost:11305/AWSWrapper/owl-s/ProcessModel.owl#AWSWrapperService_searchAmazonBoo
http://localhost:11305/BookSearchWS/ontologies/bibtex_mod.owl#Title
http://localhost:11305/BookSearchWS/ontologies/bibtex_mod.owl#ISBN
http://localhost:11305/BookSearchWS/owl-s/Concept.owl#AuthorID
http://localhost:11305/BookSearchWS/ontologies/bibtex_mod.owl#Author
http://localhost:11305/BookSearchWS/ontologies/bibtex_mod.owl#Keyword*
http://localhost:11305/BookSearchWS/owl-s/Concept.owl#PublisherID
http://localhost:11305/BookSearchWS/ontologies/bibtex_mod.owl#Publisher
Zoom
PICKING
ocalhost:11305/Zip2Weather/owl-s/ProcessModel.owl#WeatherSoap_GetCityForecastByZIP_param
http://localhost:11305/Http2WSInfo/ontologies/usLocation.owl#PhoneNumber
http://localhost:11305/Http2WSInfo/ontologies/usLocation.owl#ZipCode
http://localhost:11305/FraudLabsWS/ontologies/IP.owl#IP
http://localhost:11305/FraudLabsWS/ontologies/IP.owl#VisibleURL
http://localhost:11305/BookSearchWS/ontologies/bibtex_mod.owl#Keyword*
http://localhost:11305/Http2WSInfo/ontologies/usLocation.owl#AreaCodehttp://localh
Zoom
PICKING

FIGURE 5: Search results for CS1 and CS2 respectively from left to right

The service query mechanism uses the open source version of the soapUI [24] framework to send and receive SOAP messages. The soapUI framework supports multiple messaging protocols for services REST and SOAP. The web service broker/matchmaker selection is based on OWL-S support. The Alive matchmaker [1] was used for simulating a real-world semantic service broker. It is open source and offers flexible search query generation.

7 Case Studies

For the experiments, we selected the set of related services shown in Table 1, most of which are commercial real-world services. We used these services for two reasons:

1. Commercial services are more likely to exhibit issues concerned with ‘realistic data’.

2. Commercial services allow a degree of ‘real-world’ evaluation of our ATAM approach.

Only 6 out of 16 these services are free to use: ABS, vISBN, ZCV, GS, CW and CC. Even though these services are free, most of them are provided by commercial companies such as Google, Amazon and CDYNE. We only used the GS service in test data generation. The rest of the free services were used as services under test in the experiments.

One side effect of using commercial services is the potentially prohibitive cost involved in running our experiments. Fortunately, for most of the services used in our case studies, we have been given limited but sufficient access which facilitated our experiments. However, we could not get cost-free access for the SIP service. As a result, we had to emulate the functionality of the SIP service by creating a SOAP service that mimics the SIP service’s functionality. This new SOAP service relies on the Selfseo [22] website to provide the necessary data.

Some of the services are wrapped in SOAP web service wrappers in order to overcome two technical problems:

1. Most of these services are REST services. Unfortunately, WSDL2OWLS tool does not support REST services even those with a WSDL description.

2. Some of these services are intended to be used statically. For example, ABS requires a signature with each SOAP message and can not be invoked dynamically.

TABLE 1: Web services used in the experiments

	Service	**Input(s)**	**Output**
Case Study 1	ISBNdb by ISBN (ISBNS)	ISBN	Book
	ISBNdb by keyword (ISBNK)	Keyword	Book
	ISBNdb by title (ISBNT)	Title	Book
	ISBNdb find author ID	Author	AuthorID
	ISBNdb by author ID	AuthorID	Book
	ISBNdb find publisher ID	Publisher	PublisherID
	ISBNdb by publisher ID	PublisherID	Book
	Amazon Book Search (ABS)	ISBN	Item
	Validate ISBN (vISBN)	ISBN	Boolean
Case Study 2	Google Search (GS)	Keyword	Search result
	Strikeiron IP Lookup (SIP)	Http address	IP address
	FraudLabs IP2Location (FIPL)	IP address	US location
	CDYNE CheckPhone (CC)	Phone number	US location
	FraudLabs AreaCodeWorld	NPA + NXX	US location
	ZipCode Validator (ZCV)	ZipCode	Boolean
	CDYNE Weather (CW)	ZipCode	W. forecast

8 Research Questions and Experimental Validation

In this section, we define the four research questions, with which we help to validate our approach. We explain the method we use to investigate those questions and present and discuss the experimental results.

8.1 Research Questions

We ask the following research questions:

RQ1. Is our approach more effective than random test data generation in generating realistic data?

RQ2. Is our approach more effective than random test data generation in situations where formed test data can be generated?

RQ3. Does using formed data rather than non-formed data as MID input data affect the effectiveness of our approach?

RQ4. Can our approach effectively generate tailored test data which fulfils given constraints?

8.2 Method of Investigation

In order to answer the research questions we use two real-world case studies: CS1 and CS2. The services in each case study are listed in Table 1.

In CS1, we seek to generate realistic ISBNs using existing services. In order to evaluate the effectiveness of our approach, we selected vISBN and ABS services for evaluation and the rest of the services for generating ISBNs. vISBN is used to verify the structural validity of ISBNs and ABS is used to check if the ISBNs represent a real-world book. The search process for this case is presented in Figure 3 and the resulting service sequences are presented in Figure 5.

In CS2, we seek to generate realistic ZIP codes. In this case study, we selected ZCV and CW services for evaluation and the rest of the services for generating ZIP codes. ZCV is used to verify the structural

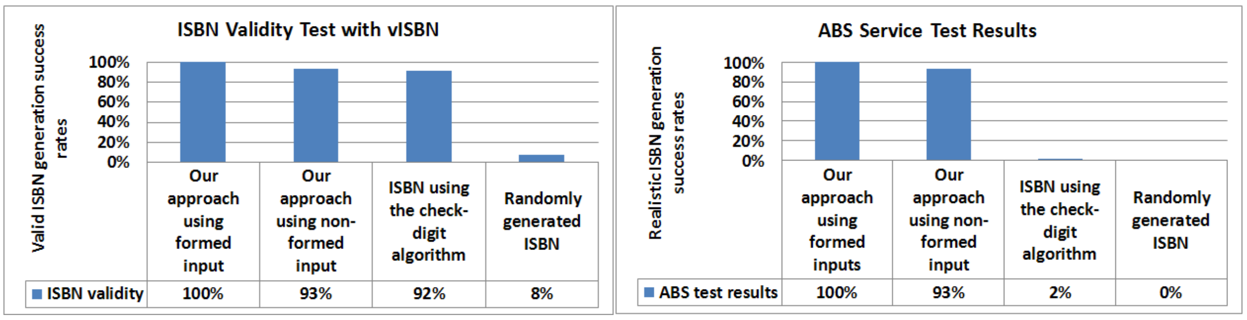


FIGURE 6: Comparison of overall success rates for our approach against random test data generation for the first case study (CS1): the image on the left presents the effectiveness at successfully generating structurally valid test data. In this evaluation, our approach achieved a 100% success rate with formed input data and a 93% success rate using non-formed input data. On the other hand, 92% of the ISBNs generated using the check-digit algorithm and only 2% generated using random generation were found to be structurally valid. The image on the right presents the effectiveness at generating realistic test data. Our approach achieved 100% success rate with formed input data and a 93% success rate using non-formed input data. Whereas only a 2% of the 'ISBNs' generated by the check-digit algorithm and none of the randomly generated ISBNs successfully tested a real-world service.

validity of the generated ZIP codes and ABS is used to check whether the generated ZIP codes represent a real-world location in US. The resulting sequences for this case are presented in Figure 5.

In order to answer RQ1, we generated 100 ISBNs and 100 ZIP codes using both our approach and random test data generation. Then we compared the effectiveness of these approaches in creating structurally valid test cases (valid ISBNs) and in testing real-world services (i.e. generating semantically valid test cases.)

For the answer to RQ2, we generated 100 unique formed ISBNs using the check-digit algorithm. Then we used the same evaluation criteria used for RQ1 and compared ISBNs generated by our approach against the formed ones.

For the answer to RQ3, we introduced a keyword generator that generates valid English words. Using this generator we generated 100 unique formed inputs and also generated unique 2 character long strings as non-formed data. The generated input data was used as input to service sequences that require *keyword* as input; ISBNK and GS. We ran these inputs and compared their effectiveness at test data generation with our approach.

There are two methods for generating tailored test data; the MID-seed method and the TSD-seed method. In the MID-seed method, ATAM generates test data using MID input data while eliminating data that does not fulfil the constraints. The TSD-seed method uses selected TSD input data which intend to direct ATAM towards data that fulfils the constraints. In order to answer to RQ4, we evaluated these two methods using both case studies with five different constraints for each case study.

8.3 Results

In generating valid ISBNs (depicted in Figure 6), our approach achieved a 100% success rate using formed data and 93% using non-formed data. Random test data generation achieves only an 8% success rate in generating valid ISBNs whereas ISBNs generated using the check-digit algorithm achieved a 92% success rate. In testing the ABS service (depicted in Figure 6), our approach achieved a 100% success rate with formed data and a 93% success rate with non-formed data. Only 2 out of 92 valid ISBNs generated with the ISBN validation algorithm successfully tested the ABS service. Valid ISBNs are unlikely to correspond to real books. As expected, none of the randomly generated ISBNs tested ABS successfully.

There are two unexpected results in Figure 6. The first one is the 92% validity score for ISBNs generated using the check-digit algorithm. This result is unexpected because we expected all ISBNs generated using this algorithm to be valid. We verified the 8 invalid ISBNs (4663277421, 2546962401, 8135892901, 5265526961, 7445203631, 1976505021, 3914584191, 6863093121) with other online ISBN validation

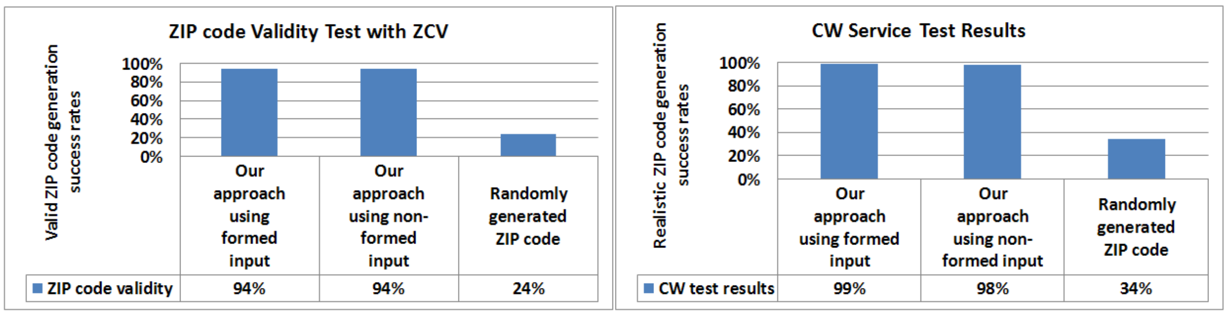


FIGURE 7: Comparison of overall success rates for our approach against random test data generation for the second case study (CS2). The image on the left presents the effectiveness in successfully generating structurally valid test data. In this evaluation, our approach achieved a 94% success rate with both formed and non-formed input data. On the other hand, only 24% of the ZIP codes generated using random generation were found out to be structurally valid. The image on the right presents the effectiveness at generating realistic test data (structurally and semantically valid). In generating realistic data, our approach achieved a 99% success rate with formed input data and a 98% success rate using non-formed input data. Whereas only 34% of the randomly generated ZIP codes successfully tested a real-world service.

tools and some of the tools validated these ISBNs while others did not. This further confirms the need for realistic test data, even in cases where the syntactic structure is well defined. Even in these apparently straightforward cases we cannot be sure to have valid instances without real-world validation of test cases. The second unexpected result is the 93% validity score for our approach with non-formed inputs using ISBNK service. The same inputs achieved 96% validity score when used with ISBNT service. This is unexpected because the ISBNT service returns only a subset of possible ISBNK service results with the same input. According to developer functionality description ISBNK perform searches for books using all details of the indexed book information (which includes the title field) and ISBNT perform searches using only the title field.

To the best of the authors' knowledge, there is no algorithm to generate structurally valid ZIP codes other than requiring that they consist of 5 digits. As a result, in the second case study (CS2) our approach is compared to random test data generation only. For the problem of structurally valid ZIP code generation (depicted in Figure 7), our approach achieved a 94% success rates using both formed and non-formed data whereas random test data generation only achieved a 24% success rate.

In testing the CW service (depicted in Figure 7), our approach achieved a 99% success rate with formed data and a 98% success rate with non-formed data. On the other hand, only 34% of the randomly generated ZIP codes successfully tested the CW service.

The service reliability problem can be observed by comparing the results of CW and ZCV tests with the same ZIP codes. After investigating possible underlying causes of these inconsistencies in results we discovered the following:

1. CW is not 100% reliable and returned unexpected results for 7 valid ZIP codes (20001, 20005, 20212, 03867, 04469, 20420, 97086). This also is the reason for our approach scoring less than 100% using formed input data.
2. ZCV's reliability appears to be lower than of CW. As a result the achieved success rates are lower than expected.
3. The website used by the SIP replacement service also exhibited reliability issues. This caused a lower score than expected in 2 cases.

In order to test the ability to generate tailored test data, we used two different scenarios for the case studies. For CS1, we tried to generate an ISBN which is published by a chosen publisher. For CS2, we tried to generate a ZIP code which belongs to a given state. In this part of the experimental analysis we did not compare our approach to random test data generation due to the expected low success rates it would achieve based on the previous evaluations.

TABLE 2: Success rates for tailored ISBN generation using MID-seed

Publisher	**Constraint**	**Success Rate**
Random House	contains(hasPublisher,'Random House')	5%
Pearson	contains(hasPublisher,'Pearson')	2%
Hachette	contains(hasPublisher,'Hachette')	0%
HarperCollins	contains(hasPublisher,'HarperCollins')	7%
Simon&Schuster	contains(hasPublisher,'Simon&Schuster')	3%

TABLE 3: Success rates for tailored ZIP code generation using MID-seed

State	**Constraint**	**Success Rate**
California	contentsEqual(State,'CA')	93%
Texas	contentsEqual(State,'TX')	29%
New York	contentsEqual(State,'NY')	17%
Floria	contentsEqual(State,'FL')	7%
Illinois	contentsEqual(State,'IL')	5%

As mentioned in Section 8.2, there are two different methods for tailored test data generation depending upon whether MID or TSD is used as a 'seed'. Table 2 and Table 3 present the results for the MID-seed method. By analysing the results, we concluded that in most cases ATAM is able to find a test input that fulfils the given criterion. Unfortunately, the success rates for the MID-seed method vary greatly depending on the selected criterion and the test data domain. The achieved success rates can be as high as 93% (as shown in Table 3) and as low as 0% (as shown in Table 2).

TABLE 4: Success rates for tailored ISBN generation using TSD-seed

Publisher	**Used input**	**Success Rate**
Random House	Random House	100%
Pearson	Pearson	100%
Hachette	Hachette	100%
HarperCollins	HarperCollins	100%
Simon & Schuster	Simon & Schuster	100%

TABLE 5: Success rates for tailored ZIP code generation using TSD-seed

State	**Used input**	**Success Rate**
California	website in California	100%
Texas	website in Texas	100%
New York	website in New York	100%
Florida	website in Florida	100%
Illinois	website in Illinois	100%

Table 4 and Table 5 present the results for the TSD-seed method. Using the TSD-seed method, ATAM successfully generated test data fulfilling the given constraints in both cases studies and achieved a 100% success rate.

8.4 Answers to Research Questions

The results from the experiments provide evidence for the effectiveness of our approach compared to random test data generation and give an answer to RQ1. In all the scenarios, our tool noticeably outperformed random test data generation. On the other hand, results from both case studies indicated that random generation is not effective when it comes to generating realistic test data. However, existing service testing, whether it is automated at all, relies heavily on random testing. The results from CS1 clearly show how ineffective random generation can be in some domains. In CS1, random generation failed to generate a single ISBN that successfully test ABS service. Even the highest success rate achieved by random generation (34% in CS2) is significantly lower than the success rate of our approach in the same case study.

For RQ2, the results shows that even with a valid generation algorithm random test data generation can not guarantee a high success rate for realistic test data generation. Even though the ISBNs generated using ISBN validation algorithm achieved a high structural validity rate, as shown in Figure 6, these ISBNs achieved only 2% success rate when testing the ABS service. On the other hand our approach for the same scenario achieved 100% success rate with formed input data.

The benefits of using formed input data can be observed in the results of both case studies, as shown in Figure 6 and Figure 7. Using formed data achieved 100% success rates in both case studies, while non-formed data achieved 93% and 98% success rates. These results provide evidence that using formed input data allows our approach to achieve better testing (answering RQ3).

In terms of tailored test data generation, as asked in RQ4, the results provide evidence for our approach's ability to generate test data fulfilling given constraints. This functionality has the potential to achieve 100% success rates using TSD-seed method (as shown in Table 4 and Table 5). Unfortunately, the success rates are not as consistent with MID-seed method (as shown in Table 2 and Table 3).

9 Conclusion and Future Work

One of the most expensive activities in testing is test data generation. Unfortunately, for most existing services test data is not currently generated automatically. We introduced a novel approach that can effectively generate test data for such services. The proposed approach not only effectively generates test data it also minimises the dependence on existing data. The major benefit of this approach is its ability to produce customised and realistic test data.

The test results presented in Section 8 provide evidence for the effectiveness of our approach compared to the state of the art automated test data generation approach. In both case studies our approach comfortably outperformed random test data generation.

The two major problems identified during experimental analysis that need to be addressed in the future work are:

1. The ability to discover and use more reliable services to increase the overall effectiveness of the approach.
2. The ability to discover and use services with lower cost in order to reduce the cost of test data generation.

References

[1] Alive Matchmaker. [Online]. Available: http://wiki.bath.ac.uk/display/alivedocs/

[2] N. Alshahwan and M. Harman, "Automated session data repair for web application regression testing." in *ICST'08: Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation.* Lillehammer, Norway: IEEE Computer Society, April 2008, pp. 298–307.

[3] N. Alshahwan and M. Harman, "Automated web application testing using search based software engineering," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, Lawrence, Kansas, USA, November 2011, to Appear.

[4] X. Bai, W. Dong, W.-T. Tsai, and Y. Chen, "WSDL-based automatic test case generation for web services testing," in *SOSE 2005: Proceedings of the IEEE International Workshop on Service-Oriented System Engineering*. Beijing, China: IEEE Computer Society, October 2005, pp. 207–212.

[5] X. Bai, S. Lee, W.-T. Tsai, and Y. Chen, "Ontology-based test modeling and partition testing of web services," in *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services*. Beijing, China: IEEE Computer Society, September 2008, pp. 465–472.

[6] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini, "WS-TAXI: A WSDL-based testing tool for web services," in *ICST '09: Proceedings of the International Conference on Software Testing Verification and Validation*. Denver, Colorado, USA: IEEE Computer Society, April 2009, pp. 326–335.

[7] A. Bertolino, J. Gao, E. Marchetti, and A. Polini, "Automatic test data generation for XML Schema-based partition testing," in *AST '07: Proceedings of the 2nd International Workshop on Automation of Software Test*. Minneapolis, Minnesota, USA: IEEE Computer Society, May 2007, pp. 4–4.

[8] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing web services: A survey," Department of Computer Science, King's College London, Tech. Rep. TR-10-01, January 2010.

[9] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing in service oriented architecture: A survey," *Software Testing, Verification and Reliability (STVR)*, Submitted for publication, An earlier version is avaliable as a technical report TR-10-01.

[10] K. Conroy, M. Grechanik, M. Hellige, E. Liongosari, and Q. Xie, "Automatic test generation from GUI applications for testing web services," in *ICSM 2007: Proceedings of the 23rd IEEE International Conference on Software Maintenance*. Paris, France: IEEE Computer Society, October 2007, pp. 345–354.

[11] G. Dai, X. Bai, Y. Wang, and F. Dai, "Contract-based testing for web services," in *COMPSAC 2007: Proceedings of the 31st Annual International Computer Software and Applications Conference*, vol. 1. Beijing, China: IEEE Computer Society, July 2007, pp. 517–526.

[12] S. Elbaum, G. Rothermel, S. Karre, and M. Fisher II, "Leveraging user-session data to support web application testing," *IEEE Transactions on Software Engineering*, vol. 31, no. 3, pp. 187–202, 2005.

[13] S. Hanna and M. Munro, "Fault-based web services testing," in *ITGN 2008: 5th International Conference on Information Technology: New Generations*. Las Vegas, Nevada, USA: IEEE Computer Society, April 2008, pp. 471–476.

[14] D. C. Ince, "The automatic generation of test data," *The Computer Journal*, vol. 30, no. 1, pp. 63–69, 1987.

[15] JXML2OWL Project. [Online]. Available: http://jxml2owl.projects.semwebcentral.org/

[16] Z. J. Li, J. Zhu, L.-J. Zhang, and N. Mitsumori, "Towards a practical and effective method for web services test case generation," in *AST'09: Proceedings of the ICSE Workshop on Automation of Software Test*. Vancouver, Canada: IEEE Computer Society, May 2009, pp. 106–114.

[17] X. Luo, F. Ping, and M.-H. Chen, "Clustering and tailoring user session data for testing web applications," in *ICST '09: Proceedings of the 2009 International Conference on Software Testing Verification and Validation*. Denver, Colorado, USA: IEEE Computer Society, April 2009, pp. 336–345.

[18] C. Ma, C. Du, T. Zhang, F. Hu, and X. Cai, "WSDL-based automated test data generation for web service," in *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering*. Wuhan, China: IEEE Computer Society, December 2008, pp. 731–737.

[19] P. McMinn, M. Stevenson, and M. Harman, "Reducing qualitative human oracle costs associated with automatically generated test data," in *STOV'10: Proceedings of the 1st International Workshop on Software Test Output Validation*. Trento, Italy: ACM Press., July 2010, pp. 1–4.

[20] MINDSWAP: Maryland information and network dynamics lab semantic web agents project. OWL-S services. [Online]. Available: http://www.mindswap.org/2004/owl-s/services.shtml

[21] J. Offutt and W. Xu, "Generating test cases for web services using data perturbation," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 5, pp. 1–10, 2004.

[22] Selfseo. Find ip address of a website. [Online]. Available: http://www.selfseo.com/find_ip_address_of_a_website.php

[23] H. M. Sneed and S. Huang, "WSDLTest - a tool for testing web services," in *WSE '06: Proceedings of the 8th IEEE International Symposium on Web Site Evolution*. Philadelphia, Pennsylvania, USA: IEEE Computer Society, September 2006, pp. 14–21.

[24] SoapUI. [Online]. Available: http://www.soapui.org/

[25] Swoogle. Semantic web search. [Online]. Available: http://swoogle.umbc.edu/

[26] S. Thummalapenta, T. Xie, N. Tillmann, P. de Halleux, and W. Schulte, "MSeqGen: Object-oriented unit-test generation via mining source code," in *ESEC/FSE '09: Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering on European Software Engineering Conference and Foundations of Software Engineering Symposium*. Amsterdam, The Netherlands: ACM Press., August 2009, pp. 193–202.

[27] W. T. Tsai, X. WEI, Y. Chen, R. Paul, and B. Xiao, "Swiss cheese test case generation for web services testing," *IEICE - Transactions on Information and Systems*, vol. E88-D, no. 12, pp. 2691–2698, 2005.

[28] M. Vieira, N. Laranjeiro, and H. Madeira, "Benchmarking the robustness of web services," in *PRDC '07: Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing*. Melbourne, Australia: IEEE Computer Society, December 2007, pp. 322–329.

[29] Y. Wang, X. Bai, J. Li, and R. Huang, "Ontology-based test case generation for testing web services," in *ISADS '07: Proceedings of the 8th International Symposium on Autonomous Decentralized Systems*. Sedona, Arizona, USA: IEEE Computer Society, March 2007, pp. 43–50.

[30] WSDL2OWL-S Project. [Online]. Available: http://www.semwebcentral.org/projects/wsdl2owl-s/

[31] W. Xu, J. Offutt, and J. Luo, "Testing web services by XML perturbation," in *ISSRE '05: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*. Chicago, Illinois, USA: IEEE Computer Society, November 2005, pp. 257–266.

[32] J. Zhang and L.-J. Zhang, "Criteria analysis and validation of the reliability of web services-oriented systems," in *ICWS '05: Proceedings of the 2005 IEEE International Conference on Web Services*. Orlando, Florida, USA: IEEE Computer Society, July 2005, pp. 621–628.