Generating Service Models by Trace Subsequence Substitution

Miao Du
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, VIC 3122, Australia
miaodu@swin.edu.au

Jean-Guy Schneider
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, VIC 3122, Australia
jschneider@swin.edu.au

Cameron Hine
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, VIC 3122, Australia
chine@swin.edu.au

John Grundy
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, VIC 3122, Australia
jgrundy@swin.edu.au

Steve Versteeg
CA Labs
Level 2, 380 St. Kilda Rd
Melbourne, VIC 3004,
Australia
steven.versteeg@ca.com

ABSTRACT

Software service emulation is an emerging technique for creating realistic executable models of server-side behaviour and is particularly useful in quality assurance: replicating production-like conditions for large-scale enterprise software systems. This allows performance engineers to mimic very large numbers of servers and/or provide a means of controlling dependencies on diverse third-party systems. Previous approaches to service emulation rely on manual definition of interaction behaviour requiring significant human effort. They also rely on either a system expert or documentation of system protocol and behaviour, neither of which are necessarily available. We present a novel method of automatically building client-server and server-server interaction models of complex software systems directly from interaction trace data, utilising longest common subsequence matching and field substitution algorithms. We evaluate our method against two common application-layer protocols: LDAP and SOAP. The results show that without explicit knowledge of the protocol specifications, our generated service models can produce well-formed responses for interactions. These responses can then be used within an emulation framework for large-scale enterprise system quality assurance purposes.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*; D.2.11 [**Software Engineering**]: Software Architectures—*Languages, Patterns*

General Terms

Design, Measurement, Verification

Keywords

Service emulation; Interaction emulation; Automatic modelling; Quality assurance

QoSA'13, June 17–21, 2013, Vancouver, BC, Canada.

1. INTRODUCTION

Modern enterprise software environments integrate a large number of software systems to facilitate complex business processes. Many of these software systems are part of "systems of systems" and, consequently, need to interact with services provided by other systems in order to fulfill their responsibilities. CA IdentityMinder (IM) [8], for instance, is an enterprise-grade identity management suite supporting management and provisioning of users, identities and roles in large organisations across a spectrum of different endpoint systems. It is typically deployed into large corporations, such as banks and telecommunications providers, who use it to manage the digital identities of personnel and to control access of their vast and distributed computational resources and services. A significant and non-trivial engineering challenge is how to assure, before live deployment, the quality of such software systems that are to be able to interoperate across the often heterogeneous services provided by a large-scale environments and to investigate the effects of different environment configurations on their operational behaviour.

Due to the often non-trivial interaction patterns between a software system and its operating environment, traditional standalone-system-oriented testing techniques are inadequate and ineffective in assuring quality of such systems (*cf.* [12] for a detailed discussion). Enterprise software environment *emulation* [13, 14] has been postulated as an alternative approach to providing executable, interactive representations

of operating environments. By *modelling* the *interaction behaviour* of individual systems in an environment and subsequently simultaneously executing a number of those models, an enterprise software environment *emulator* provides an interactive representation of an environment which, from the perspective of an external software system, appears to be a real operating environment. Hence, the emulation approach allows for a more systematic approach to assure quality of enterprise systems in a "system of systems" context [12].

Being able to create light-weight executable models is pivotal to the emulation approach. The most common approach is to manually define interaction models with the use of available knowledge about the underlying interaction protocol(s) and system behaviour(s), respectively. This entails defining sometimes complex sequences of request/response patterns between elements of the system including suitable parameter values. However, in a realistic environment, neither of these are necessarily available at the required level of detail (if at all), a scenario not uncommon when third-party, legacy and/or mainframe systems are involved. Additionally, the large number of components and component interactions in such systems makes manual approaches very time-consuming and error-prone. If the environment changes with new enterprise elements or communication between elements, these manual protocol specifications must be further updated.

In order to address this problem, we describe a new approach that infers enterprise system element interaction behaviour via mining interaction recordings (henceforth referred to as *interaction traces*). These interactions are between an *endpoint system* and elements in its deployment *environment*. More specifically, we introduce a framework that, given an incoming request to a modelled system, our technique (i) searches for a suitably "similar" request in a database of previously recorded interaction traces, (ii) identifies the commonalities and differences between the incoming request and the recorded request, and (iii) generates a response based on the identified commonalties and differences and pre-recorded responses. In a proof of concept implementation, we use longest common subsequence matching and field substitution to realise these message processing steps. To evaluate our approach, we report on its applicability using the common enterprise application-layer protocols LDAP [23] and SOAP [1], respectively.

The rest of this paper is organised as follows: Section 2 introduces the key motivation for this work using a concrete enterprise system emulation example. Section 3 discusses related work, followed by the presentation of the various components of the response-generation framework in Section 4. In Section 5, we present the results of our evaluation and discuss the relevant findings as well as identified limitations. Section 6 concludes the paper and gives directions for future work.

2. MOTIVATION

Consider the IdentityMinder (IM) enterprise system, our *system-under-test*, in a proposed deployment environment, as shown in Figure 1. Engineers want to evaluate whether the IM can scale to handling the number of likely *endpoints* in the deployment scenario, given (i) likely maximum number of endpoints; (ii) likely maximum number of messages between endpoint and system; (iii) likely frequency of message sends/receives needed for the system to respond in acceptable timeframe; (iv) likely size of message payloads given deployment network latency and bandwidth; and (v) the system's robustness in the presence of invalid messages, too-slow response from end-points, or no-response from endpoints. Messages being exchanged between the system and endpoints adhere to various protocols. For example, an LDAP message sent by an endpoint to the IM needs to be responded to with a suitable response message sent in reply by the IM, in an acceptable timeframe and with acceptable message payload. Subsequent messages sent by the endpoint to the IM using the LDAP response message payload need to utilise the previous response information.

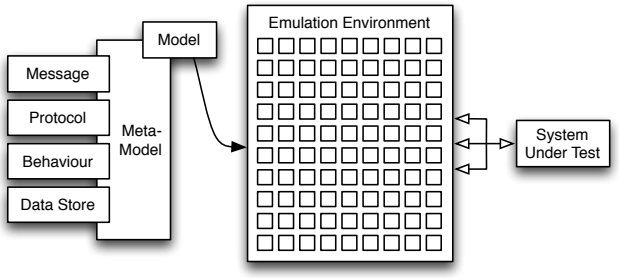

Figure 1: Service Emulation Approach.

In previous work we developed an emulator that could scale to thousands of virtual endpoints to validate such deployment scenarios [12, 25]. Endpoint *emulations* would send and receive messages to the system under test, generating suitable messages to confirm to its protocol and expectations. Unfortunately, considerable manual effort is required to define protocols, such as LDAP and SOAP messaging, and also, considerable effort to implement suitable endpoint behavioural emulation within the emulator. This also required considerable expert knowledge of the protocol.

Key requirements that we have identified and that we want to address in our current work include:

- we need to be able to record and then analyse real system and real endpoint interactions, to enable us to synthesise a protocol definition from these observed interaction traces;
- when our emulation environment is emulating an endpoint, upon the endpoint receiving a message, we need to be able to deduce a best-matching response message and suitable payload;
- we need to reply to the sending system with appropriate message and payload synthesised via the analysis and matching process.

3. RELATED WORK

A major challenge when attempting to assure the quality of large enterprise systems is producing a suitable test-bed environment. To this end, physical replication of a real-world deployment environments very quickly becomes difficult to effectively manage, or even achieve. Recreating the heterogeneity and massive scale of typical production environments is, in many cases, simply impossible given QA team's resources. As outlined above, in our domain, we need a real system-under-test - such as IM - to be able

to communicate with literally thousands of endpoints, behaving as they would in a real production environment. Provisioning such a testing environment with thousands of real client and server hardware platforms, suitably configured networks, and appropriately configured software applications for our IM system-under-test to communicate with, is near-impossible.

Over the years, a number of approaches have been proposed that aim to provision testing environments suitable for quality assurance activities required to test enterprise software systems. Hardware virtualization tools, such as VMWare [24] and VirtualBox [16], are capable of replicating specific facets of deployment environments. However, a general rule of thumb states that a virtual CPU to physical core ratio in the order of ten to one as the practical upper limit [22]. Hence hardware virtualization tools suffer similar scalability limitations as physical recreation of deployment environments. Mock objects [7] mitigate some of the scalability concerns. However, they are often too language specific and require the re-implementation of some of an environment's functionality, resulting in testing environment configuration and maintenance problems and requiring detailed knowledge of environment components.

Performance and load testing tools, such as the ones proposed in [6, 10, 11, 21], provide a means to emulate many thousands of software system clients with limited resources. However, they are designed to generate scalable client *load* towards a target system, rather than the opposite direction needed for our problem situation. Thus while they provide a suitable platform to scale client-to-server load, enterprise systems like CA IM require a test environment with system-under-test to environment load scaling, a related but fundamentally different challenge.

In order to overcome shortcomings with these existing enterprise system QA approaches and tools, the creation of "virtual", or emulated, deployment environments has been proposed. Ghosh and Mathur [9] state that "an emulation environment, which is able to provision representations of diverse components, enables tests of components that interact with a heterogeneous environment, while scalable models put scalability and performance testing for components into practice." In our prior work [12, 13, 14], we proposed an enterprise software environment emulator, called Kaluta. It provides a large-scale and heterogeneous emulation environment capable of simultaneously emulating thousands of endpoint systems on one or a few physical machines. We have shown this scales very well to the needs of enterprise system QA as outlined above. However, the creation of executable endpoint models relies on the availability of a precise specification of the interaction protocols used. It is also time consuming, error-prone and subject to considerable maintenance effort in heterogeneous deployment environments.

ITKO LISA [18] is a commercial software tool which aims to emulate the behaviour of services which a system under test interacts with in its deployment environment. It does this by 'mimicking' responses that a real service would produce when sent a request by the enterprise system under test. One of the key features of LISA is that, after recording a set of real interactions between an enterprise system and an endpoint, it uses these to produce responses to further requests, thus behaving like a 'virtual' service. LISA is able to consider the interaction state when sending a response, and uses field substitution (called *magic strings*) in the responses for fields it detects are identical in the request and response. LISA requires the transport protocol and the service protocol to be known in advance of the recording for the modelling to be effective.

Research in protocol reverse engineering is an important reference to our work. Early effort in reverse engineering was for protocol determination. By analysing a large amount of packets and traces captured on networks, researchers are able to obtain structure information of the target protocol for network analysis and even automatically reverse engineering the state-machine model of network protocols [3]. Cui *et al.* [4], for example, proposed an emulator aiming to mimic both client and server side behaviours. With the emulator, they can record/replay the interactions of web applications for checking conformance of web server behaviours. Although the proposed approach deals with the emulation of interaction process, they essentially intend to test conformance of the client-side systems rather than work in the opposite direction as we do.

4. APPROACH

In essence, our approach works as follows:

- A large enterprise system-under-test, such as IM, is observed communicating with endpoint(s) in its deployment environment using a tool such as *Wireshark*.
- The emulation environment uses this trace recording as a source for protocol analysis and response generation, storing it in a repository of request/response pairs.
- When running QA tests against the system-under-test, the emulation environment receives a request from the system-under-test and uses the trace history to identify potential valid response messages.
- We use a set of algorithms to compare the current request, previous request/response pairs, set of historical sequences of request/response pairs, and the set of values in the request to synthesise a response message.
- The emulation environment returns the generated response message to the system-under-test, fulfilling its expectations of a response message.
- The system-under-test consumes the generated response message and continues running.

This is in contrast to existing emulation approaches where requests received by the emulation environment are processed using (typically) manually-specified scripts to generate a response.

4.1 Preliminaries

For the purpose of the proposed technique, we assume that for a given protocol under investigation, we are able to record a sufficiently large number of interactions between two (or more) software endpoints. Tools like Wireshark [15] have the functionality to filter network traffic and record messages of interest in a suitable format for further processing. We also assume that these recordings are "valid", that is, that the sequence of recorded interactions are (i) correct with regards to the temporal properties of the underlying protocol and that (ii) each request and response message is well-formed.

Without loss of generality, we assume that each request is always followed by a single response. If a request does not generate a response, we insert a dedicated "no-response" message into the recorded interaction traces. If, on the other hand, a request leads to multiple responses, these are concatenated into a single response. We use such an approach in our evaluation (*cf.* Section 5) to merge multiple LDAP search result entries into a single response.

Given these assumptions, we define a number of constructs needed to express our framework more formally. We start with the notion of the most basic building block, the set of *message characters*, denoted by $\mathcal{C}$. We require equality and inequality to be defined for the elements of $\mathcal{C}$. For the purpose of our study, $\mathcal{C}$ will most likely comprise of the set of valid Bytes that can be transmitted over a network or the set of printable Characters as a dedicated subset. Furthermore, we define $\mathcal{M}$ to be the set of all (possibly) empty *messages* that can be defined using the message characters. A message $m \in \mathcal{M}$ is a non-empty, finite sequence of message characters $c_1c_2c_3 \ldots c_n$ with $c_i \in \mathcal{C}, 1 \leq i \leq n$. We consider two messages $m_1 = c_{1,1}c_{1,2} \ldots c_{1,l}$ and $m_2 = c_{2,1}c_{1,2} \ldots c_{2,n}$ to be equal if $l = n$ and $c_{1,i} = c_{2,i}, 1 \leq i \leq n$.

A single interaction I consists of a request, denoted by *Req*, as well as the corresponding response, denoted by *Res*. Both *Req* and *Res* are elements of $\mathcal{M}$ and we write (*Req*, *Res*) to denote the corresponding request/response pair. An *interaction trace* is defined as a finite, non-empty sequence of interactions, that is, $I_1I_2I_3 \ldots I_n$. Finally, we define the set of interactions $\mathcal{I}$ as a non-empty set of interaction traces.

4.2 Processing Requests

The motivation behind our approach is that if an incoming request is very similar to one of the recorded requests (having a suitable notion of "similarity"), then the response should also be similar to the corresponding previously recorded response. Hence, identifying the differences between the incoming and previously recorded requests should give us a good indication how the corresponding recorded response can be altered in order to synthesize a matching response.

For example, assume that the recorded interaction traces between an LDAP client and server contain a search request for all entries with the name *Baker*. If an incoming request defines a search for all entries with the name *Parker*, then the two requests can be considered to be similar (both are search requests; only the name is different). Hence, if we replace all occurrences of 'Baker' with 'Parker' and the adjust the LDAP message-id accordingly, then the altered response to the recorded search for Baker is probably a "good enough" response to the search for Parker for emulation purposes.

Consequently, our proposed framework consists of two main processing steps: (i) given an incoming request to an emulated enterprise system endpoint from the system under test, we search for a suitably "similar" request in the previously recorded interaction traces. (ii) Our system then synthesizes a response for the incoming request based on the similarities in the request itself and the "similar" request identified in the interaction traces, as well as the recorded response of the "similar" request.

Using the definitions introduced above, our framework can thus be formalized as below. To facilitate the presentation, we denote Req_{in} as the incoming request and $I^*(\mathcal{I})$ as the set of all interactions in $\mathcal{I}$.

$$Res_{out} = trans\ (Req_{in}, Req_{sim}, Res_{sim})$$

with

- $(Req_{sim}, Res_{sim}) \in I^*(\mathcal{I})$; and
- $\forall\ (Req_i, Res_i) : dist(Req_{in}, Req_{sim}) \leq dist(Req_{in}, Req_i)$

where *dist* and *trans* denote user-defined *distance* and *translation* functions, respectively, allowing the framework to be tailored for the specific needs of given context.

The distance function *dist* is used to compute the distance between two requests. We require (i) the distance of a message m with itself to be zero, that is $dist(m, m) = 0$, and (ii) the distance between two non-identical messages m_1 and m_2 to be greater than zero. Depending on what kind of distance function is used, a different pre-recorded request will be chosen to be the most "similar" to the incoming request.

The *translation* function's responsibility is to synthesize a response for the incoming request. As a simplification of our work, we made the decision to *ignore* temporal properties in our framework, that is, the synthesized response solely depends on the incoming request and the recorded interaction traces, but not on any previously received or transmitted requests or responses, respectively, Adding a temporal dimension to the framework is part of our future work.

4.3 Common Subsequence Alignment

What kind of distance measure(s) should we choose in order to best express our intention of similarity as discussed in the previous section? A widely used notion of similarity is the *edit distance* [20] between two sequences s_1 and s_2, indicating the minimum number of modifications (insertions, deletions, and substitutions) in order to obtain s_2 from s_1. A very similar problem has also been identified in the area of bioinformatics in order to determine the similarities in the amino acide sequences of proteins [19]. For the purpose of this work, we are using a modified version of the solution presented by Needleman and Wunsch [19] as our distance measure. This is because as the stochastic approach presented by Ristad and Yianilos [20] relies on a suitably configured benchmark corpus, we may not always be able to generate this from the recorded interaction traces.

The basic idea of the sequence alignment is *align* all common subsequences of two sequences under comparison and insert *gaps* into either of the sequences when they differ. In order to avoid "random" alignments of a small size, we modified the algorithm in such a way that a minimum length is required in order to identify common subsequences as such.

The following example briefly shows how our message alignment process works. Consider the following two text sequences:

- Where is my computer book?
- Where is your computer magazine?

The common subsequences are "Where is ", " computer ", and "?". (Note the spaces in around " computer ".) "my" versus "your" and "book" versus "magazine" are the two differing parts of the two sequences. The standard Needleman-Wunsch algorithm would align the character 'y' common to "my" and "your", although it probably makes more sense not to identify 'y' as a common subsequence, hence the need for a minimum length of common subsequence. The fully aligned sequences will be as follows (we use the character '$\star$' to denote an inserted gap):

- Where is my⋆⋆⋆⋆ computer book⋆⋆⋆⋆⋆⋆⋆?
- Where is ⋆⋆ your computer ⋆⋆⋆⋆magazine?

The distance between two sequences is defined by the number of gaps inserted to both sequences in the alignment process – 18 in the example above. In order to allow for a better comparison of similarity across multiple protocols and/or scenarios, we define the *dissimilarity ratio* as the ratio of the "raw" edit distance divided by the length (*i.e.* number of elements) of both sequences, *e.g.* $18/(26+32) = 0.31$ in the example given above. The dissimilarity ratio, as illustrated in this section, was used as the distance measure for the evaluation of our approach (*cf.* Section 5). Two identical sequences will have a dissimilarity ratio of 0 and the bigger the ratio, the more *dissimilar* two sequences are.

4.4 Symmetric Field Identification

The second step in our approach is to synthesize a response for the incoming request, exploiting the commonalities between this request, its best match, as well as the associated recorded response. In order to do so, we again rely on common subsequence identification.

Many protocols encode information in request messages that are subsequently used in the corresponding responses. For example, application-level protocols such as LDAP add a unique message identifier to each request message. The corresponding response message must contain the same message identifier in order to be seen as a valid response. Therefore, any approach that attempts to synthesize responses for LDAP must "copy" the message-id into the corresponding response message. Similarly, information associated with a specific request operation (*e.g.*, a search pattern for a search request) will often also be "copied" across from the request to its response. We will refer to such information as *symmetric fields* for the rest of this work.

We use the common subsequence algorithm described in the previous section in order to identify symmetric fields: they are the common subsequences of a request and its associated response. However, as the symmetric fields of a request may not appear in the same order and/or cardinality, we cannot rely on a "simple" sequence alignment. Instead we have to compute the entire alignment matrix (as defined by [19]) to identify common subsequences. Again, in order to avoid small, "random" common subsequences, a threshold has to be defined as to when a common sequence of characters is considered a symmetric field.[1]

Once the symmetric fields between Req_{sim} and Res_{sim} are determined, the corresponding field information has to be identified in the incoming request Req_{in} and substituted in Res_{sim} in order to synthesise the final response Res_{out}.

The following example will help illustrating the identification of symmetric fields and how symmetric fields are used in the response generation process. Consider the following LDAP search request[2]

```
Message ID: 18
ProtocolOp: searchRequest
  ObjectName: cn=Mal BAIL,ou=Administration,
              ou=Corporate,o=DEMOCORP,c=AU
  Scope: 0 ( baseObject )
```

we are looking to generate a response for. The search for the most similar request in the available interaction traces returns the following request

```
Message ID: 37
ProtocolOp: searchRequest
  ObjectName: cn=Miao DU,ou=Administration,
              ou=Corporate,o=DEMOCORP,c=AU
  Scope: 0 ( baseObject )
```

that is paired-up with the following response:

```
Message ID: 37
ProtocolOp: searchResEntry
  ObjectName: cn=Miao DU,ou=Administration,
              ou=Corporate,o=DEMOCORP,c=AU
  Scope: 0 ( baseObject )
Message ID: 37
ProtocolOp: searchResDone
  resultCode: success
```

Symmetric field identification results in two substrings that are identical across request and response:

```
Message ID: 37
ProtocolOp:
```

and

```
  ObjectName: cn=Miao DU,ou=Administration,
              ou=Corporate,o=DEMOCORP,c=AU
  Scope: 0 ( baseObject )
```

Substituting the corresponding values from the incoming request, we synthesize the following response:

```
Message ID: 18
ProtocolOp: searchResEntry
  ObjectName: cn=Mal BAIL,ou=Administration,
              ou=Corporate,o=DEMOCORP,c=AU
  Scope: 0 ( baseObject )
Message ID: 18
ProtocolOp: searchResDone
  resultCode: success
```

4.5 Implementation

We have developed a proof of concept realisation of our framework, including the sequence alignment, the symmetric field identification and substitution algorithms, as well as the underlying modified Needleman-Wunsch algorithm. This was proof of concept prototype was implemented in the Java programming language. Both Wireshark and LISA were used to capture network traffic and exported into a format suitable for input into our Java implementation. At the time of writing, the implementation was not specifically optimized (both from a performance and memory consumption perspective). This is a task we intend to cover in future work.

5. EVALUATION

In this section, we present the experiments that we conducted to evaluate the effectiveness of the approach presented in the previous section and discuss the results of our experiments. More specifically, we introduce our experimental setup as well as our evaluation criteria in sections 5.1 and 5.2, respectively. In Section 5.3, we present the results of our cross-validation and illustrate the accuracy of the synthesized responses. Finally, we discuss limitations of our current approach and identify possible areas of future improvements in Section 5.4.

[1]A detailed description of the actual algorithm had to be omitted due to space limitations.

[2]For presentation purposes, we use a more user-friendly layout of the corresponding LDAP messages. For processing, all messages were "normalized", that is, newlines, leading white spaces etc. were removed from the textual representation.

5.1 Experimental Setup

Although one of the aims of our work is to enable emulation for unknown or ill-specified protocols, for evaluation purposes, we used two protocols where the precise message structures as well as the corresponding temporal properties are known: the Simple Object Access Protocol (SOAP) [1] and the Lightweight Directory Access Protocol (LDAP) [23]. Both are commonly used application-layer protocols and hence lend themselves as case studies for our evaluation. SOAP is a light-weight protocol designed for exchanging structured information in a decentralised, distributed environments whereas LDAP is widely used in large enterprises for maintaining and managing directory information.

The interaction trace for SOAP used for our evaluation was generated based on a recording of a banking example using the LISA tool [18]. The protocol consists of 7 different request types, each with a varying number of parameters, encoding "typical" transactions one would expect from a banking service. From a pre-defined set of account id's, account names etc. we then randomly generated an interaction trace containing $1,000$ request/response pairs. Amongst those, we had 548 unique requests (with only 22 requests occurring multiple times), 714 uniqe responses (the replicated ones are predominantly due to the fact that the `deleteTokenResponse` message only had true or false as possible return values), and 23 duplicated request/response pairs. For the purpose of our evaluation, we considered this a sufficiently diverse "population" of message to work with.

The following is one of the recorded requests:[3]

```
<?xml version="1.0"?>
 <S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
   <ns2:getAccount xmlns:ns2="http://bank/">
    <accountId>867-957-31</accountId></ns2:getAccount>
  </S:Body>
 </S:Envelope>
```

with the following the corresponding response:

```
<?xml version="1.0"?>
 <S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
   <ns2:getAccountResponse xmlns:ns2="http://bank/">
    <return>
     <accountId>867-957-31</accountId>
     <fname>Steve</fname>
     <lname>Hine</lname>
    </return>
   </ns2:getAccountResponse>
  </S:Body>
 </S:Envelope>
```

This example illustrates that besides the structural SOAP information encoded in both messages, there is specific information that appears in both, the SOAP request and SOAP response, such as the account-ID in the example above.

LDAP is a binary protocol that uses an ASN.1 encoding to encode and decode text-based message information to and from its binary representation, respectively. For the purpose of our study, we used a corresponding decoder in order to translate recorded LDAP messages into a text format and an encoder to check whether the synthesized responses were well-formed (*cf.* Section 5.2). In future work, we plan to investigate whether we can omit the encoding/decoding steps and directly manipulate the corresponding binary representations.

The LDAP interaction trace used for the evaluation consisted of 498 unique interactions containing the core LDAP operations, such as *adding*, *searching*, *modifying* etc. applied to CA's *DemoCorp* sample directory [2]. The trace did not contain any duplicated requests or responses, and the search responses contained a varying number of matching entries, ranging from zero to 12.

The following briefly illustrates the textual representation of a *search* request:

```
Message ID: 15
ProtocolOp: searchRequest
  ObjectName: cn=Juliet LEVY,ou=Administration,
              ou=Corporate,o=DEMOCORP,c=AU
  Scope: 0 ( baseObject )
```

and the corresponding response, consisting of the merge of a *search result entry* and a *search result done* message:

```
Message ID: 15
ProtocolOp: searchResEntry
  ObjectName: cn=Juliet LEVY,ou=Administration,
              ou=Corporate,o=DEMOCORP,c=AU
  Scope: 0 ( baseObject )
Message ID: 15
ProtocolOp: searchResDone
  resultCode: success
```

This example LDAP request contains a (unique) message identifier (`Message ID: 15`) and a specific object name (`ObjectName: ...`) as the root node for the search to be used. The corresponding responses use the same message identifier (to indicate the request they are in response to) and the `searchResEntry` message refers to the same object name as the request. For our approach to synthesize correct LDAP responses, the corresponding information needs to be copied across from the incoming request to the most similar response to be modified.

5.2 Cross-Validation Approach and Evaluation Criteria

A *cross-validation approach* [5] is one of the most popular methods for assessing how the results of a statistical analysis will generalise to an independent data set. For the purpose of our evaluation, we applied the commonly used 10-fold cross-validation approach [17] to both the recorded SOAP and LDAP messages, respectively.

As shown in Figure 2, we randomly partitioned the original interactions' data set into 10 groups. Of the 10 groups, group i (*cf.* top-left rectangle in Figure 2) is considered to be the *evaluation group* for testing our approach, and the remaining 9 groups constitute the *training set*. The cross-validation process is then repeated 10 times (the same as the number of groups), so that each of the 10 groups will be used as the evaluation group once.

In order to investigate the applicability and effectiveness of our approach, for each message in the evaluation group, we compared the resulting synthesized response with the corresponding recorded response. We defined the following criteria to evaluate the "validity" of synthesized responses:

[3]Similar to the LDAP example given in Section 4.4, we removed any newlines, whitespaces etc. introduced for presentation purposes during processing.

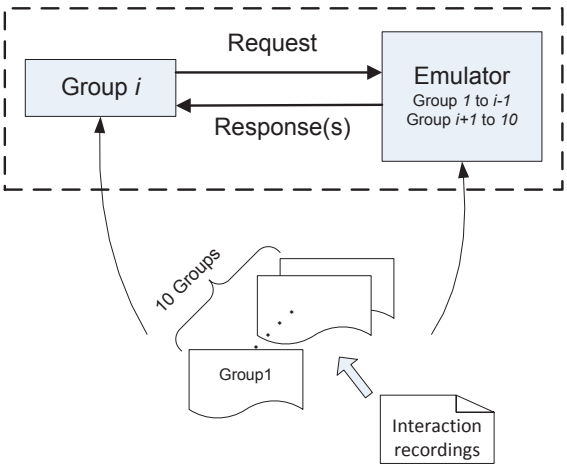

Figure 2: 10-fold Cross Validation Approach

1. **Identical**: the synthesized response is identical to the recorded response if all characters in the synthesized response exactly match those in the recorded response (as per our definition in Section 4.1.)

2. **Well-Formed**: this criterion indicates that the synthesized responses correspond to the structure required for responses as defined by the underlying protocol. Synthesized responses that do not meet this criteria are considered to be **Ill-Formed**.

3. **Protocol Conformant**: this criterion requires that synthesized responses are *well-formed*. On top of that, it requires that the responses conform to the temporal interaction properties of the given protocol, that is, the temporal consistency between request and response is preserved.

For the purpose of our evaluation, we used a weaker notion of *protocol conformance* as the order in which the requests are selected from the evaluation set is *random* and, as a consequence, unlikely to conform to a sequence of protocol conformant requests. Therefore, we consider a synthesized response to be protocol conformant if it conforms to the temporal properties at some point in time.

If a synthesized response is *identical*, then the other two properties (*well-formed* and *protocol conformant*) are implied. We can guarantee this under the assumption that the recorded interaction traces we use are considered to be valid and conform to the temporal interaction properties of the protocol. However, it is very well possible that the response generation process synthesises a well-formed response that is *not* protocol conformant (as we will further discuss in Section 5.4).

For the purpose of emulation, protocol conformance is the most important property a synthesized response needs to exhibit. The aim of an emulatable endpoint model is not necessarily to reproduce the behaviour of a real endpoint to 100% - as long as the responses an emulated endpoint provides are "good enough", this will be sufficient for many quality assurance activities [12].

5.3 Evaluation Results

To benchmark the effectiveness of our approach for synthesizing responses, we used a random selection strategy as baseline where for an "incoming" request, the corresponding response is randomly selected from the responses contained in the training set. All generated responses for both, the approach based on common subsequence alignment (CSA) as well as the random selection strategy, were categorised according to the criteria introduced in Section 5.2.

Table 1 summarizes the result of our experiments. Besides the number of responses falling in each of the four categories, it lists the number of *valid* response messages, that is, the sum of identical and protocol-conformant messages. Please note that the column *Well-form.* does *not* include the number of valid messages, that is, only those well-formed responses that are not protocol conformant are listed. Furthermore, for all *non-identical* responses, Table 1 also lists the median as well as the maximum dissimilarity ratios, respectively.

5.3.1 Evaluation results for SOAP.

Table 1 compares the different outcomes of the random response strategy and our CSA approach. Most importantly, no ill-formed SOAP responses were generated by either the baseline approach or the common subsequence alignment approach. However, our approach outperformed the random selection strategy in a number of aspects. Specifically, (i) *all* $1,000$ synthesized responses using our approach were protocol conformant, compared to only 33 of the randomly selected responses, and (ii) 9.3% of the generated responses were identical to the recorded responses in our approach, compared to 3.3% in the random selection strategy.

Analysing the non-identical responses in more detail, we observed that the worst dissimilarity ratio of the common subsequence alignment approach is 0.046 (all other dissimilarity ratios are smaller). With an average response length of 239 characters, this gives us a maximum edit distance of 24 between the synthesized response and the "expected" response (*i.e.* the response associated with the most similar request). This shows that for the SOAP case study used, our approach was able to synthesize responses significantly more accurately than the random strategy.

5.3.2 LDAP results

A summary of the result of the LDAP experiments are also given in Table 1. For the common subsequence alignment approach, 466 (out of 498) generated response messages were identical to the corresponding recorded responses (89.9%), and an additional 18 of the generated responses met the protocol conformant criterion (3.6%). Therefore, a total of 487 (or 97.8%) of all generated responses were considered to be valid. Of the remaining 14 responses, 9 were well-formed, but had the wrong message type, and 5 responses were *ill-formed*. Both aspects will be discussed further in the following section.

In case of the random selection strategy, all responses were well-formed (as expected), but as many as 438 responses were valid (87.5%), which is not much worse than the CSA approach. This rather surprising result can be explained by the fact that about 90% of all recorded requests are `searchRequest` messages (with different search criteria), and hence the likelihood of randomly choosing another `searchRequest` as the "best" match is rather high.

Experiment	No.	Valid	Ident.	Conf.	Well-form.	Ill-form.	Mean dsim.∗	Max dsim.∗
SOAP Random	1,000	33	33	0	967	0	0.046	0.259
SOAP CSA	1,000	1,000	93	907	0	0	0.020	0.046
LDAP Random	498	438	2	436	39	0	0.067	0.873
LDAP CSA	498	484	466	18	9	5	0.200	0.775

Table 1: Summary of Evaluation Results.

This also explains the rather low number of only well-formed messages.

With regards to the rather high maximum dissimilarity ratio, there are a number of very similar search requests in our data set, some of them resulting in responses with zero or one search result entries only, others with a large number of entries. Therefore, if a response with a small number of entries is used as the basis to synthesize a response for a request that expects a large number of entries (or vice versa), then the edit difference between the synthesized and expected responses is rather large and, consequently, the dissimilarity ratio as well. However, for the purpose of our overall goal of being able to generate valid responses, this is not a problem as despite a high dissimilarity, a valid response is generated as long as all symmetric fields are replaced correctly.

5.4 Discussion and Limitations

Based on the investigation of both SOAP and LDAP experimental results, we can see that our approach is able to automatically generate valid responses in most situations. However, as illustrated in the results for LDAP, a small proportion of protocol non-conformant or even ill-formed responses were synthesized. In order to better illustrate the underlying reasons, consider the following example where a protocol non-conformant response was synthesized. The following request

```
Message ID: 171
ProtocolOp: addRequest
  ObjectName: cn=Miao DU,ou=Finance,
              ou=Corporate,o=DEMOCORP,c=AU
  Scope: 0 ( baseObject )
```

resulted in the generation of the following response:

```
Message ID: 171
ProtocolOp: modifyResponse
  resultCode: success
```

The response is well-formed and the `Message Id` field has been substituted properly. However, according to the LDAP protocol specification, an `addRequest` adding an extra node to an LDAP directory, must result in an `addResponse`, and not in an `modifyResponse` as given in the example above. The reason for this unexpected response can be explained by the fact that the test set contains a `modifyReqest` with precisely the same `ObjectName` and `Scope` as the `addRequest` above and a `Message ID` of 151. Our distance measure identified this `modifyRequest` as the most similar match and hence, the associated modify response was used as the basis for synthesizing the response.

Most application-level protocols define message structures containing some form of *operation* or service name in their requests, followed by a *payload* on what data this service is expected to operate upon [12]. In the example above, the fact that `addRequest` and `modfiyRequest` denote different operations was not taken into consideration when the most similar request was chosen. In future work we intend to devise suitable heuristics allowing us to (semi-)automatically identify which part(s) of a request message most likely correspond to a service name, use this information to divide the set of interaction traces into clusters containing a single service type only, and restrict the search for the most similar request to one cluster only. This should also improve the run-time performance of our approach.

The following example indicates an ill-formed LDAP response. It is worth noting that the `Message Id` and `ObjectName` fields have been properly substituted from the corresponding request. However, the `protocolOp` values of `addResEntry` and `addResDone` are invalid LDAP operation names and were flagged as such by the LDAP encoder used.

```
Message ID: 154
ProtocolOp: addResEntry
  ObjectName: cn=Miao DU,ou=Legal,
              ou=Corporate,o=DEMOCORP,c=AU
  Scope: 0 ( baseObject )
Message ID: 154
ProtocolOp: addResDone
  resultCode: success
```

Similar to the previous example, there is a mismatch in the operation name of the most similar request: whereas the request message denotes an `addRequest`, the test set contains a `searchRequest` with a very similar message id and an identical `ObjectName`. The message id was substituted correctly, but all occurrences of `search` in the response were substituted to `add`, resulting in an ill-formed LDAP response. Again, clustering the set of interactions according to the service/operation name would have most likely prevented the selection of a `searchRequest` as the most similar request to an `addRequest`.

Comparing the dissimilarity measures of our LDAP and SOAP results (*cf.* the corresponding values in Table 1), we noticed that non-zero SOAP similarities are generally significantly lower than the non-zero LDAP results, indicating that our non-exact matching SOAP responses are typically less dissimilar to the real responses than their LDAP counterparts. This can be attributed to the fact that SOAP messages contain a significant amount of structural information which is easily duplicated in the generated responses. This makes the generated and real SOAP responses *similar* even when there are, perhaps significant, differences in the payload.

This is not a major issue for our approach in general. However, it implies that comparing the effectiveness of various distance and translation functions *across* protocols needs to be done carefully as low(er) dissimilarity ratios in one protocol may be more due to the amount of common *structural* information than the properties of the distance and translation functions used. Similar to the abovementioned

clustering approaches, we intend to use heuristics to (semi-)automatically separate payload and structure in messages and devise similarity measures that give payload information a higher weighting than structural information in order to improve the cross-protocol comparisons.

In our tests we have examined text-based messages with SOAP being a text-based protocol and for LDAP, we used a text representation. Future work will attempt to synthesise responses directly for binary protocols. This will bring extra challenges. In order to give one example, binary packets often contain the packet length as part of the encoding. Our field substitution method could change the length of packets and, therefore easily produce an ill-formed response. In order to address this issue, without using explicit knowledge of the message structure, we will need to devise methods to automatically identify fields such as the packet length.

6. CONCLUSIONS AND FUTURE WORK

We have demonstrated that it is feasible to create a system which automatically builds executable interactive models of software service behaviour from recorded message traces, without requiring explicit knowledge of the internals of the service or of the protocols the service uses to communicate. This eliminates the human effort of manually specifying models and furthermore reduces reliance on a system expert or the need for extensive documentation of the service protocol and behaviour, respectively.

Our approach is to build models directly from interaction traces, recorded between a system-under-test and a software service which it depends upon. The interaction traces are used as a library with which to compare new requests from a system-under-test. The Needleman-Wunsch longest common subsequence alignment method is used to calculate the distances between a new request and requests in the interaction traces. In our initial approach, we assume that the response corresponding to closest matching request, is the best response to send back to a system-under-test. Symmetric field substitution is used to modify the sent response so that it is tailored to the new request.

The Needleman-Wunsch longest common subsequence distance measure combined with symmetric field substitution produces promising results for the two protocols tested. For LDAP, 94% of synthesised responses were identical to that of the real service, and 98% of generated responses were protocol conformant. For the more complexly structured SOAP, while only 9% of synthesised responses were identical, 100% were protocol conformant.

Future work will refine our request matching algorithms. For example, some fields in the request (such as the operation name) are more critical for identifying which response should be sent back. We will explore methods for automatically identifying critical fields. Possible approaches include reverse engineering the protocol structure, or using clustering to group responses and requests and then infer the critical junctures at which different types of responses are sent for similar looking requests. Utilising conversation state information may also improve the accuracy of synthesised responses. Finally, there is a need to test our methods on a wider range of protocols. Proprietary mainframe protocols, which are often poorly documented, are a particularly interesting category to test.

Acknowledgements

This work is supported by ARC Linkage Project LP100100622 *Large-Scale Emulation for Enterprise Software Systems.*

7. REFERENCES

[1] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple Object Access Protocol (SOAP) 1.1,. W3C Note 8, W3C, May 2000. `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`.

[2] CA Technologies. *CA Directory Administration Guide (r12.0 SP11)*, 2012.

[3] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. Prospex: Protocol Specification Extraction. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (SP '09)*, pages 110–125. IEEE, 2009.

[4] Weidong Cui, Vern Paxson, Nicholas C. Weaver, and Randy .H. Katz. Protocol-independent Adaptive Replay of Application Dialog. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium*, February 2006.

[5] P.A. Devijver and J. Kittler. *Pattern recognition: A statistical approach*. Prentice/Hall International, 1982.

[6] Sebastian Elbaum, Gregg Rothermel, Srikanth Karre, and Marc Fisher II. Leveraging user-session data to support web application testing. *IEEE Transactions on Software Engineering*, 31:187–202, 2005.

[7] Steve Freeman, Tim Mackinnon, Nat Pryce, and Joe Walnes. Mock roles, objects. In *Companion to the 19th annual ACM SIGPLAN conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 236–246, New York, NY, USA, 2004.

[8] Matthew Gardiner. CA Identity Manager, November 2006. White Paper on CA Identity Manager.

[9] Sudipto Ghosh and Aditya P. Mathur. Issues in Testing Distributed Component-Based Systems. In *In First International ICSE Workshop on Testing Distributed Component-Based Systems*, 1999.

[10] John Grundy, Yuhong Cai, and Anna Liu. Generation of Distributed System Test-Beds from High-Level software Architecture Descriptions. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE '01)*, pages 193–200, San Diego, November 2001.

[11] John Grundy, Yuhong Cai, and Anna Liu. SoftArch/MTE: Generating Distributed System Test-Beds from High-Level Software Architecture Descriptions. *Automated Software Engineering*, 12(1):5–39, January 2005.

[12] Cameron Hine. *Emulating Enterprise Software Environments*. Phd thesis, Swinburne University of Technology, Faculty of Information and Communication Technologies, 2012.

[13] Cameron Hine, Jean-Guy Schneider, Jun Han, and Steve Versteeg. Scalable Emulation of Enterprise Systems. In *Proceedings of the 20th Australian Software Engineering Conference (ASWEC 2009)*, pages 142–151, Gold Coast, Australia, April 2009. IEEE Computer Society Press.

[14] Cameron Hine, Jean-Guy Schneider, and Steve

Versteeg. Reac2o: a runtime for enterprise system models. In Jamie Andrews and Elisabetta Di Nitto, editors, *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 1́0)*, pages 177–178, Antwerp, Belgium, September 2010. ACM.

[15] Ulf Lamping, Richard Sharpe, and Ed Warnicke. *Wireshark Users's Guide*, 2012.

[16] Pen Li. Selecting and Using Virtualization Solutions: our Experiences with VMware and VirtualBox. *Journal of Computing Sciences in Colleges*, 25(3):11–17, January 2010.

[17] Geoffrey J. McLachlan, Kim-Anne Do, and Christophe Ambroise. *Analyzing Microarray Gene Expression Data*. Wiley-Interscience, 2004.

[18] John Michelsen. Key Capabilities of a Service Virtualization Solution, October 2011. ITKO White Paper. Available at: `http://www.itko.com/resources/service_virtualization_capabilities.jsp`.

[19] Saul B. Needleman and Christian D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[20] Eric Sven Ristad and Peter N. Yianilos. Learning String-Edit Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532, May 1998.

[21] Sreedevi Sampath, Sara Sprenkle, Emily Gibson, Lori Pollock, and Amie Souter Greenwald. Applying Concept Analysis to User-Session-Based Testing of Web Applications. *IEEE Transactions on Software Engineering*, 33(10):643–658, 2007.

[22] Joe Sanchez. Squeezing virtual machines out [of] CPU cores, 2011. VM Install.

[23] J. Sermersheim. Lightweight Directory Access Protocol (LDAP): The Protocol. RFC 4511 (Proposed Standard), June 2006.

[24] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2001. USENIX Association.

[25] Steve Versteeg, Cameron Hine, Jean-Guy Schneider, and Jun Han. Emulation of Cloud-Scale Environments for Scalability Testing. In *Proceedings of the 12th International Conference on Quality Software (QSIC '12)*, pages 201–209, Xi'an, China, August 2012. IEEE.