Using a Semantic Knowledge Base for Communication Service Quality Management in Home Area Networks

Liam Fallon  
Network Management Lab, LM Ericsson, Athlone, Co. Westmeath, Ireland  
Email: liam.fallon(at)ericsson.com

Declan O’Sullivan  
Knowledge & Data Engineering Group (KDEG)  
Trinity College Dublin, Ireland  
Email: declan.osullivan(at)cs.tcd.ie

Abstract—The data required for automatic optimization of user services usually exists in current systems, but that data is not modelled or linked in a way that facilitates automation. Knowledge engineering is a promising approach for managing the disparate communication service quality management information data sets and the links across those data sets. Once a knowledge base is in place, semantic techniques can be used to analyse and suggest optimizations to service quality. This paper describes our work in building, populating and evaluating a knowledge base for an IPTV service in Home Area Networks. Population of the knowledge base was implemented using terminal reports. The characteristics of the approach were evaluated through experimentation and the evaluation results are presented in this paper.

I. INTRODUCTION

The NGMN[1] and the TM-Forum[2] have both highlighted the importance of automated optimization of communication service delivery. Automatically optimizing end user service delivery presents three challenges[1]. Firstly, the service expectations for a set of services at the service consumption point must be set, agreed and actively managed. Secondly, once those expectations have been agreed, the actual service experience and service context that users are experiencing must be monitored. Thirdly, the service context must be adjusted to optimize service delivery. Any system that addresses these challenges must be adaptable, highly flexible, and operate with minimal human intervention[2].

Service expectations are a balance struck between quality, cost, and resource efficiency for the delivery of services. Contracts such as Service Level Agreements (SLAs)[3] are often used to express service expectations where service delivery crosses organizational boundaries. Service expectations are not formally set with service providers such as YouTube that utilise pure Internet connectivity, end users usually subscribe for a certain bandwidth level or bundle that implicitly sets a service expectation. The service experience is the quality of service (QoS) of a service measured at the point of delivery and the quality of experience (QoE) of the service as perceived by the end user[3][4]. The service context is that set of factors which can affect service quality. Toutain et al.[5] categorize five dimensions of service context: User context, Network context, Social context, Physical context and Device context.

The data required for automatic optimization of user services usually exists in current systems, but that data is not modelled or linked in a way that facilitates automation. While collection of measurements and deployment of configuration changes is automated, analysis and optimization of end user services is largely executed as a manual task with support from tooling[1][6]. Where automation exists, it is implemented in very specific network domains and applications[7][8].

Managing end user service expectations, experience, and context for automatic optimization can be seen as a knowledge engineering problem. Using ontologies and vocabularies to express service expectations, experience, and context knowledge allows the relationships between service knowledge from these sources to be mapped. Once this knowledge has been captured, queries, rules and reasoning can be used to analyse service experience in terms of the expectations on those services and the context in which those services are being delivered and to suggest optimizations for those services. Those optimizations can be applied to the network delivering the services using policies or other means.

This paper describes our work in building and populating a knowledge base for communication service quality management. In order to apply knowledge engineering to manage end user services, a knowledge base to hold service expectations, experience and context must be constructed, and mechanisms to populate that knowledge base with knowledge must be put in place. Building a knowledge base means selecting and designing a set of ontologies and vocabularies and specifying semantic references across them to represent links between knowledge in the knowledge base. The populating mechanisms insert knowledge into the knowledge base using ontologies and vocabularies and build the specific references between individual pieces of knowledge in the knowledge base.

We focus on building a knowledge base for end user service experience and context because they change dynamically during operation and service expectations do not. In building our knowledge base, we used existing ontologies and vocabularies where possible, specifically FOAF[9] for user...
context and WGS84 [10] for positional context. Other aspects were represented with newly developed ontologies.

As terminal reporting [11] [12] is increasingly being used to monitor the QoS and QoE of end user services as well as the service context at the terminal, we used terminal reports as a data source for knowledge base population. SAWSDL [13] was selected as a mechanism to extract and encode semantic knowledge from terminal reports. Mappings were developed from XML elements in terminal reports to create RDF [14] class individuals and references between those individuals.

Any automated service analysis and optimization system must be able to react in near real time to changes in user service experience and context; a system managing an IPTV end user service must be able to put optimizations in place in a time frame of 5-10 seconds to avoid end user complaints [15]. Where knowledge engineering is used for analysis and optimization, it is crucial that the portion of this time budget used for populating the knowledge base be as small as possible.

Therefore, we carried out a thorough evaluation of our approach which measured the characteristics of our populating mechanism with various terminal reporting loads.

The network domain used in the evaluation scenario is a Home Area Network (HAN) federation [16] and the service being monitored is an IPTV service. Terminals report end user service experience and context at session start, session shutdown and periodically every minute during a session.

This paper is organised as follows. Section II gives a summary of related work. Section III describes the ontologies and vocabularies used to hold end user service experience and context, and outlines the semantic lifting mechanism used. Section IV presents the results of our evaluation of the approach, and Section V summarises the paper.

II. BACKGROUND AND RELATED WORK

This section summarises related work. The first sub-section examines the Service Quality Monitoring function of Service Quality Management described in the TM-Forum’s Telecom Applications Map (TAM) [17]. The following sub-section examines models, vocabularies, and ontologies that are applicable to capture service quality management knowledge. The next sub-section examines the state of art in applying semantic techniques to Service Management. The final sub-section summarises the state of the art.

A. Service Quality Monitoring

Metrics from network nodes are a highly standardised [18] [19] data source for assessing the quality of delivery of resources carrying services. Metrics such as packet loss, delay, and jitter are available as is information on events such as equipment failures or overloads. Metrics on logical entities such as VLANs and MPLS tunnels may also be available.

Reporting directly from service terminals is the most accurate way of evaluating end user service context. Service experience metrics available in terminal reports include QoS metrics such as packet loss and latency, and QoE estimations made by algorithms running in the terminal. Terminal reports also report context information such as the service user, the location of service delivery, and device information such as CPU load, memory usage, disk space, and remaining energy. Standardization activities for terminal reporting are specific to particular service or networking domains. RTCP [11] and RTCP XR [20] is used for quality reporting on RTP based streaming services. The MS-QoE [21] protocol publishes QoE metrics from clients to a monitoring server. The 3GPP use terminal reports to monitor Packet-switched Streaming Service (PSS) [12], Multimedia Broadcast Multicast Service (MBMS) [22] and Multimedia Telephony (MMTel) [23] sessions. As there is no common standard for terminal reporting, GSQR [24] was proposed as a unified approach for terminal reporting in previous work.

B. Applicable Models, Vocabularies, and Ontologies

Although there is a long history of modelling in the telecommunication domain, the focus has been largely on network equipment and resource models rather than on services being carried by networks. SNMP [25] MIBs described in SMI [26] and the DMTF’s CIM [27] are typical of such models.

Some work has been undertaken in modelling telecommunication services. Garschhammer et al. [28] describe a generic telecommunication service model that identifies service relationships such as user, functionality, and QoS parameters. Rodosek [29] uses service template models to describe a service in terms of a generic service model. The TM-Forum’s SID [30] is a generic service model with relationships and mappings between service entities described conceptually as uses and requires relationships. SALmon [31] is a domain specific language for service modelling, allowing specification of service model instances. Another approach is to model telecommunication services as policies. In DEN-ng [32], services sit in the Business View of the Policy Continuum. The Ontology for Support and Management [33] captures changes to user, location, device, and service context as events that trigger policies associated with entities modelled in DEN-ng. Sheridan-Smith et al. [34] describe a policy based service definition language that captures the static and dynamic aspects of services. The BREIN ontology [35] is an OWL ontology that defines basic QoS concepts for telecommunication services.

Toutain et al. [5] make the case for modelling the context of services and state that for contextual service models “the most promising approach uses semantic models based on ontologies”. They describe five dimensions of context for end user services: User context, Network context, Social context, Physical context, and Device context.

The context of some service aspects can be captured with a general ontology or vocabulary. The Friend of a Friend vocabulary [9] can represent service users. The Semantically Interlinked Online Communities (SIOC) [36] ontology captures semantic information of online communities. The WGS84 Geo positioning vocabulary [10] allows the geographical location of a user to be represented. Traffic measurements may be captured with the Monitoring Ontology for IP Traffic (MOI) [37].
C. Semantic Techniques for Service Management

Ontology mapping is used to map relationships between concepts across ontologies. O’Sullivan et al.[38] describe the OISIN process for developing and deploying such mappings. Keeney et al.[39] use ontology mappings in a Knowledge Based Network to enable semantic interoperability between producers and consumers of network information. The Knowledge Based Network allows producers to publish and consumers to subscribe to knowledge. Information models describing knowledge to be published into the system are analysed off line to produce deployable run time mappings. This means that information in any form, once mapped, can be published into or consumed from the system. Reasoning is used to determine which set of mappings should be used for a particular event forwarding operation.

Queries and rules are alternatives to reasoning for using semantic knowledge. SPARQL[40] is a query language for RDF documents. SWRL[41] is a rule language for ontologies. Semantic lifting is used to translate information into a semantic form. SAWSDL[13] is used to annotate element definitions in XML schemas with semantic references and mappings that translate XML element information to semantic knowledge. SAWSDL is primarily used to annotate WSDL[42] web service definitions[43], existing SAWSDL implementations do not support non-WSDL XML schemas. Lehtihet and Agoulmine[44] describe an approach for reusing information from SMI[26] and CIM[27] models by mapping those models to a common UML structure, which is then transformed to XML schemas. Those XML schemas are then annotated with SAWSDL. Frutos et al.[35] describe an approach where semantic annotation is used to annotate SLA templates that are used in automated contract negotiations in service composition. A service composition QoS ontology is used as a model, and different services advertise their capabilities by using SAWSDL annotations to reference the QoS model.

There are advantages and drawbacks to semantic representation of knowledge[45][46]. Ontologies enable model interoperability, facilitate incremental modelling, are expressively rich, and allow the use of models from various sources. Initial costs of building models are substantial, semantic content in models varies in detail, and tooling for ontologies is immature.

D. Summary of Related Work

Monitoring of network metrics on physical and logical network resources is widely deployed[18][19]. Service metrics are increasingly being reported from terminals[11][12].

While there are many models for telecommunication network [25][27] and service[30][31] context, few are semantically enabled[33][34]. Modelling service context semantically is seen as a promising approach[5]. Some common vocabularies and ontologies such as FOAF[9] and WGS84[10] are applicable to represent communication service context. Semantically representing telecommunication knowledge allows true knowledge sharing but the initial modelling costs are substantial[45][46].

III. Description

This section explains the vocabularies and ontologies selected to represent end user service experience and context, and shows the mappings used to encode information from terminal reports into those vocabularies and ontologies. It goes on to describe the SAWSDL framework developed to extract, encode, and persist semantic information from terminal reports.

A. Use Case

An IPTV service is running in a single HAN or in a HAN federation[16], a group of physical home area networks on which services can be managed as a single HAN domain.

Terminal reporting is used to collect the service experience of all end user IPTV sessions in the HAN federation. Terminal reports are issued at session start, periodically during a session, and at session termination. Each terminal report carries quantitative Quality of Service (QoS) metrics among which are throughput and latency, and the Mean Opinion Score (MOS), a qualitative Quality of Experience (QoE) metric for video playout quality calculated by the user terminal. The knowledge in those terminal reports is semantically mapped and stored into a knowledge base. That knowledge will subsequently be used to analyse and optimize the quality of IPTV service delivery in the HAN federation.

B. Representing Service Experience and Context Semantically

The structure of the knowledge base used in this work to store service experience is shown in Fig. 1. The RDF individuals generated from terminal reports contain the data properties shown in Fig. 2. The terminal report data that has been identified as being of particular interest for analysis is listed below.

Time: Durations of incidents, terminal report time stamps, start times and duration of measuring periods for metrics are mapped. Understanding how metrics and other parameters change over time is valuable input for analysis and optimization, especially as user terminals and home gateways that are not synchronized to network time.

Physical Equipment: Terminal parameters are mapped because problems such as hardware or software faults on particular types of terminals may be quite easy to infer.

Location: The location of the terminal can be used to determine if problems are localised. Location is especially important in mobile networks where it varies with time.
C. Semantic Lifting of Reports

This section describes the SAWSDL framework that has been developed to extract, encode, and persist semantic information from terminal reports.

A parser reads the XML schema documents for the terminal report and builds an Annotated Element Lookup Reference (AELR) that records each annotated element in the XML report and builds an Annotated Element Lookup Reference (AELR) that records each annotated element in the XML schema. SAWSDL annotations are used because they can be used as a reference to other user and service information, which will be a useful source of knowledge for analysis.

Listing 1. Abbreviated terminal report XML schema

```xml
<xs:element name="TerminalReport">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="User" type="xsd:complexType"/>
      <xs:element name="Session" type="xsd:complexType"/>
      <xs:element name="Metrics" type="xsd:complexType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 2. User terminal report XML element to RDF XSL transformation

```xml
<xsl:template match="/">
  <xsl:text>
    @prefix foaf: <http://xmlns.com/foaf/0.1/> .
    @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
    @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
    @prefix term: <http://fame/TerminalReports/TerminalReport#> .
    @prefix user: <http://fame/TerminalReports/User#> .
    @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

    user:User<xsl:value-of select="User/ID"/>
    a user:User;
    user:ID <xsd:value-of select="TR/Terminal/User/ID"/>
    .
    user:Person foaf:Person<xsl:value-of select="User/ID"/>
    .
    user:Address <xsd:value-of select="User/Address"/>
    .
    foaf:Person foaf:Person <xsd:value-of select="TR/Terminal/User/ID"/>
    .
    foaf:Person foaf:Person <xsd:value-of select="TR/Terminal/User/Name"/>
    .
    term:Terminal<xsl:value-of select="TR/Terminal/ID"/>
    .
    term:User user:xsl:value-of select="User/ID"/>
  </xsl:template>
</xsl:transform>
```

User: Information about the user and their type of service are mapped. The user ID and the service type of the user can be used as a reference to other user and service information, which will be a useful source of knowledge for analysis.

Session: Sessions are used to keep track of communication instances and for mapping those instances across a network. Sessions have end points, paths, communication characteristics, and payloads.

Metrics: QoS and QoE metrics reported on terminals and sessions represent the service experience of the end user.

The format of terminal reports is specified in an XML schema, with terminal reports complying with the restrictions specified in that schema. SAWSDL annotations are used to annotate schema elements from which semantic information is to be extracted. XSL transformations are used to extract semantic information from elements in terminal reports; triggered using SAWSDL liftingSchemaMapping annotations on elements. The mappings used are listed in Table I. As an example, the XML schema annotation for a User XML element in a terminal report is shown in Listing 1 and its XSL transformation is shown in Listing 2.

As well as carrying out the mappings from XML elements to RDF, the transformations build references between RDFS class individuals. The references between TerminalReport, Terminal, User, Session, and Metrics RDFS class individuals are essential to allow analysis of the extracted semantic knowledge. In Listing 2, the Terminal User reference is set to point at the instantiated User individual, thus building a semantic link from the terminal to the user of that terminal.

C. Semantic Lifting of Reports

This section describes the SAWSDL framework that has been developed to extract, encode, and persist semantic information from terminal reports.

A parser reads the XML schema documents for the terminal report and builds an Annotated Element Lookup Reference (AELR) that records each annotated element in the XML schema and the mappings associated with that element. SAWSDL liftingSchemaMapping annotations are used because
TABLE II
AN AELR ENTRY

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Element or attribute name</td>
</tr>
<tr>
<td>Type</td>
<td>Element or attribute</td>
</tr>
<tr>
<td>XPath Statement</td>
<td>XPath expression identifying element</td>
</tr>
<tr>
<td>Namespace Map</td>
<td>List of URIs to name spaces in XPath expression</td>
</tr>
<tr>
<td>Annotation Type</td>
<td>modelReference, liftingSchemaMapping,</td>
</tr>
<tr>
<td></td>
<td>or loweringSchemaMapping</td>
</tr>
<tr>
<td>Map References</td>
<td>List of URIs to mappings for this element</td>
</tr>
</tbody>
</table>

XML terminal reports are processed as they stream past; the liftingSchemaMapping annotation indicates which elements in a report are to be extracted and mapped to semantic information, as shown in Fig. 3. The Interceptor uses the AELR produced by the Parser to detect which elements should be extracted to generate semantic information.

The parser finds SAWSDL annotations in XML schemas. An XPath expression identifying the element or attribute position in the schema hierarchy is composed and stored to the AELR together with SAWSDL annotation mappings for that element or attribute. Table II shows an AELR entry structure.

The parser uses a recursive algorithm to find annotations on XML elements and attributes in the schema files loaded into the Parser. It loads each XML schema into memory and iterates over the elements at the highest level of the schema hierarchy, calling a ParseElement routine for each element. It checks for annotations on simple type elements and complex type elements. If complex type elements have XML attribute declarations, these are checked for SAWSDL annotations as well. Complex types can have nested elements. The Parser iterates over nested elements and recursively calls the ParseElement routine to parse each nested element. When the parser finds a SAWSDL annotation on an element or attribute declaration, it composes and stores an AELR entry for that element or attribute.

The HAN federation issues terminal reports in GSQR[24] format, with reports in other formats such as RTCP or 3GPP being translated into GSQR by adapters. The Interceptor, shown in Fig. 3, reads those terminal reports, extracts semantic information, and stores it to a knowledge base. On start up, the Interceptor opens a connection to a RDF repository and calls the Parser with a set of terminal report XML schemas to generate the AELR. It then waits for terminal reports to appear in a specified file system directory.

When a terminal report appears, the Interceptor reads and validates it against its XML schema; all reports must be valid or they are rejected. The XPath expressions in each liftingSchemaMapping AELR entry are run against the terminal report document, producing a list of matching nodes. The transformations specified in the AELR entry are run on each node in turn producing RDF fragments. These RDF fragments are stored in the RDF store.

D. Knowledge Base Maintenance

It is neither necessary nor practical to store all raw terminal data indefinitely in the knowledge base. Once raw terminal data has been used to analyse service experience and optimize the network delivering those services, that data is retained for reporting purposes and so that detailed drill-down examination of analysis and optimization results can be carried out. Having historical reporting and drill-down is a desirable feature for the system, but a time limit on the retention of raw data must be set in order to keep the knowledge base to a manageable size.

A parameterized aggregation and deletion maintenance procedure that aggregates and removes raw data is one strategy that addresses the need for historical reporting and drill down while managing knowledge base size. Such a procedure aggregates raw data hourly, daily, weekly, monthly, and yearly and allows retention periods to be specified for each type of time aggregate, with retention periods set based on the storage available. The knowledge base maintenance procedure runs periodically to aggregate data and delete data that is older than its retention period.

IV. Evaluation

This section describes how we evaluated our approach. As stated in I, the time budget used for populating the knowledge base should be as small as possible, so the evaluation measured the characteristics of the populating mechanism with various terminal reporting loads.

In the evaluation scenario, terminal reporting is active in a HAN federation where six simultaneous IPTV sessions are active, a reasonably heavy load for a HAN federation. The evaluation set out to investigate if it is possible to populate a knowledge base with knowledge that references a set of ontologies from terminal reports in a time window of a minute when terminal reporting is set up so that each of six terminals report every minute. The evaluation also assessed the performance and scalability of the Interceptor implementation shown in Fig. 3, thereby determining the scale of federation to which the approach can be applied.
A. Scenario

Terminal reporting is activated for IPTV sessions running in a HAN federation. Packets are being dropped at an increasing rate during the session due to a network problem, leading to deterioration in the quality of the IPTV service.

In the scenario, the metrics that vary are the packet loss, a quantitative QoS metric, and the MOS, a qualitative QoE metric. At session start, packet loss in the session is zero. As the session continues, packet loss increases and MOS values deteriorate. Eventually, the packet loss reaches a level where the picture is highly distorted and the session is terminated. Packet loss values are measured in the terminal and MOS values may have been entered by the user using a form or estimated using a MOS estimation algorithm such as those described in [47].

In the scenario, MOS values vary from 5.0 (perfect) to 0.0 (worst). The relationship between packet loss and MOS is approximated using the equation below.

\[
MOS = P - (e^l - 1) \text{ if } (P - (e^l - 1)) > 0, \ 0 \text{ otherwise}
\]

where \(P\) is a perfect MOS score of 5.0 and \(l\) is packet loss expressed as a percentage. The equation gives a reasonable estimation of MOS as packet loss increases because at values of over 2%, media streams become unwatchable[48].

B. Simulation and Execution

We ran a series of 17 tests, varying the number of simultaneous simulated IPTV sessions being monitored 6 to 300 over the test series. We repeated the 17 tests in 10 independent runs. Tests were executed a simulated environment. The goal was to measure the populating mechanism characteristics, so communication costs and application loads were not simulated.

In a single simulated IPTV session, 11 terminal reports were generated to simulate 10 minutes of operation: an initial report, 9 periodic reports at 1-minute intervals, and a terminating report. A simulator running in a Java visual machine (JVM) spawned a thread for each simulated IPTV session, with each session starting at a random time between 0 and 1 minutes in order not to have all terminals reporting at the same instant in a test. The reports generated were created as files in a simulation directory, from where the Intercepter read them.

The Intercepter, running in a separate JVM, called the Parser once at startup to generate the AELR (see Table II) from a terminal report XML schema file. It then read simulated terminal reports from the simulation directory and processed them as they appear. If there were no further terminal reports in the directory, the Intercepter slept for 1 second before checking again for terminal reports.

The simulated environment ran on a HP Z600 workstation (2.4 GHz Intel 4-Core Xeon processor/24GB memory/Fedora 15 Linux). The Sesame\(^1\) RDF store was used, persisting to a local PostgreSQL\(^2\) database over HTTP using a Sesame servlet running on Apache Tomcat\(^3\). The machine was performing no other tasks besides routine operating system housekeeping.

C. Results

Four parameters were measured during each test. The Total Test Time (Fig. 4) is the time taken by the Intercepter from the start of the read of the first terminal report of a test to the time of completion of storage of the last terminal report. The Generation to Store Time (Fig. 5) is the time taken from generation of a terminal report to its semantic information being stored in the RDF store. The Processing Time (Fig. 6) is the time taken to process a terminal report from reading the report from the file system to complete storing its semantic information in the RDF store. The Intercept Time (Fig. 7) is a portion of the Processing Time taken from reading a report from the file system to being ready to store the semantic information in the RDF store; that is, the Processing Time without the RDF store contribution.

D. Interpretation of Results

The experimental results demonstrate that it is possible to populate a knowledge base with knowledge that references a set of ontologies from terminal reports in the time window

---

1Sesame Version 2.6.2, see http://www.openrdf.org
2PostgreSQL Version 9.0, see http://www.postgresql.org
3Apache Tomcat Version 6.0, see http://tomcat.apache.org
of a minute when terminal reporting is set up so that each of six terminals report every minute. At that reporting rate, the average processing time for a terminal report is less than 0.4 seconds. Such a population rate suggests that the system can handle much higher reporting loads.

1) Reporting Limit on Interceptor: Tests should take approximately 11 minutes to complete because each simulated terminal starts reporting after a delay of 0 to 1 minutes and reports for 10 minutes (See IV-B). Total Test Time (Fig. 4) shows the observed length of each test and is highly correlated for all runs. With reporting loads up to 175 terminals, test times are very close to 11 minutes, indicating that the Interceptor is processing terminal reports as they are generated. Total Test Time increases well beyond 11 minutes for higher reporting loads because the Interceptor continued to run after simulation completed to clear the terminal report backlog. Therefore, the observed results indicate that the Interceptor can handle reporting loads of up to 150 terminals.

2) Relationship between Generation to Store and Processing Times: At terminal reporting loads higher than 150 terminals, Generation to Store Time, again highly correlated across runs, increases rapidly as terminal report backlogs build up. A difference between Average Generate to Store and Processing Time is expected at higher reporting loads because terminal reports are more likely to have to wait in the file system for some time after generation before being processed. At lower loads, a 1-2 second delay is caused because the Interceptor sleeps for a period of 1 second if no terminal reports are found on the file system. This delay coupled with the processing time, gives the total Generate to Store time.

3) Average Processing and Interceptor Time: Average Processing Time (Fig. 6) becomes well correlated across runs and exhibits an increasing trend at reporting loads from 60 terminals. This trend is expected because at higher loads, the RDF store performance deteriorates because the amount of links to be built as extra data is added increases. The low near-linear nature of this increase indicates that the RDF store handles increases in reporting load well.

Average Interceptor Time (Fig. 7) is also highly correlated across runs and shows a very desirable decreasing trend with increasing terminal load. Interceptor times are a very small portion of processing time, decreasing from 4.2% to 0.8% with increasing processor load. This decrease probably happens because, at low terminal loads, Java XML handling classes become inactive and must be re-instantiated.

The high Average Processing Time observed in 4 runs at low reporting loads merits examination. The data for those runs showed that 2-5% of the database writes for the tests in question took longer than 1 second, much more that the average of 0.5% of slow writes across all run tests. The high rate of slow writes on tests with low loads probably occurred due to database connections becoming inactive due to low write rates or database maintenance procedures.

The maximum number of terminal reports that can be handled per second by the Interceptor is the inverse of processing time. A processing time of 200 ms per terminal report means that 5 reports can be handled per second. Table III takes the lowest and highest average processing times and sets out what the maximum number of terminal reports per second and maximum reporting load would be for those processing times, excluding all other factors. If each terminal reports every minute, the Interceptor should be able to handle reporting loads of between 148 and 223 terminals.

4) Spread of Processing Time and Interceptor Time Data: Using average values does not always give an accurate view of a set of measurements so examined the data spread for Processing Time and Interceptor time, the two most interesting measurements. The combined spread of Processing Time and Interceptor Time data for all runs is presented for each test in Fig. 8 and Fig. 9 in the form of box and whisker plots. As an example, the leftmost plot, the data spread for 660 tests; 6 terminals produce 11 reports each in 10 runs. High outliers, data points 1.5 times the Inter Quartile Range (IQR) beyond the upper quartile were eliminated. There were no low outliers. The percentage of data points identified as outliers is indicated in parenthesis on the x-axis labels.

The spread in Processing Time is skewed to the right, with skew tending to increase in a non-uniform manner as terminal load increases. The spread in Interceptor Time is also skewed, but skew decreases as terminal report load increases.

The box and whisker plots identified a relatively high number of outliers. The plots in Fig. 10 and Fig. 11 show data points including outliers in Processing and Interceptor Time results for tests with 300 terminals reporting for all runs, and were produced to show the influence of outliers. There are a significant number of outliers in the early measurements, probably due to system initiation costs. The Processing Time plot shows that outliers occur frequently during the run, whereas the Interceptor Time plot shows that outliers are extremely rare once the system has initiated.

This analysis shows that Interceptor time, the time taken to process a terminal report semantically, is stable and predictable in the system. Processing Time, which includes the time taken

<table>
<thead>
<tr>
<th>Run</th>
<th>Test</th>
<th>Avg. Processing Time (ms)</th>
<th>Max. Terminal Reports/s</th>
<th>Max. Reporting Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>5</td>
<td>4</td>
<td>269</td>
<td>3.72</td>
</tr>
<tr>
<td>High</td>
<td>2</td>
<td>1</td>
<td>405</td>
<td>2.47</td>
</tr>
</tbody>
</table>
to store the semantic information to the RDF store, is less stable and predictable.

V. Summary and Future Work

The evaluation results demonstrate that populating a knowledge base with semantic information from terminal reports is possible at a reporting load of 6 terminals. The results further demonstrate that, if the overall time budget for putting optimizations in place is 5-10 seconds, knowledge base population will use much less than a second of that budget.

The experimental results show a flat or decreasing trend in average processing (Fig. 6) and intercept times (Fig. 7) up to the maximum measured reporting load of 300 terminal reports per minute. It shows that semantic lifting of data from terminal reports or other XML streams is feasible and implementable at least up to that level of load.

Semantic lifting from XML streams is not a bottleneck when used as a technique to obtain data to which to apply semantic analysis. The main bottleneck in the system is the time taken to store knowledge persistently to the RDF store. A system where raw terminal report knowledge is held in memory for the duration of the semantic analysis and optimization process with only results and aggregations being stored persistently to the RDF store will execute much more quickly. The time taken to lift terminal reports is also more predictable than the time taken to store lifted terminal reports in the RDF store.

The linear trend in processing time with load is a very desirable property because the amount of processing time used by the Interpreter determines the minimum time in which a terminal report can be processed and, as the number of terminal reports being processed increases, the time taken to process each individual terminal report does not increase significantly. This indicates that the Interpreter can scale to handle large loads and is capable of handling reporting loads likely to be generated by small businesses.

It would be interesting to explore if the relationship between processing time and reporting load continues beyond a load of 300 terminal reports per minute. Another topic for future work is the investigation of alternative Interpreter deployments. A threaded Interpreter implementation is an obvious improvement, but complicates construction of references in the knowledge base. A pool of intercepters deployed in a Network Management System (NMS) may be able to handle the terminal report load of an entire network such as a mobile access network. An Interpreter may be implementable on a machine such as a Home Gateway or Set Top Box with much lower capabilities than that used in this work.

The work described in this paper has provided us with a populated knowledge base of service experience for ongoing service analysis and optimization work. In that work, we are examining the applicability and characteristics of using semantic techniques such as rules and inference to the MOS scores and other service experience metrics in the knowledge base to automatically assess and improve service quality. This work will be the subject of future publication submissions.

Acknowledgment

This work was partially supported by Science Foundation Ireland under the FAME SRC Grant No 08/SRC/11403.
REFERENCES


