Software Performance Evaluation of a Web Services-Based Clinical Decision Support Infrastructure

Christina Catley  
Carleton University  
1125 Colonel By Drive  
Ottawa, ON K1S 5B6  
1 613 235-1982  
cc@engsoc.org

Dorina C. Petriu  
Carleton University  
1125 Colonel By Drive  
Ottawa, ON K1S 5B6  
1 613 520-5652  
petriu@sce.carleton.ca

Monique Frize  
Carleton/University of Ottawa  
1125 Colonel By Drive  
Ottawa, ON K1S 5B6  
1 613 520-2600 ext. 8229  
mfrize@connect.carleton.ca

ABSTRACT
This paper has two contributions: a) it proposes a web-services based infrastructure to support Clinical Decision Support Systems (CDSSs) for processing multi-domain medical data from the obstetrical, perinatal and neonatal care domains, and b) applies the Software Performance Engineering (SPE) technique to the proposed infrastructure. This extends a XML-based framework for medical data interoperability and integration of CDSSs into the Neonatal Intensive Care Unit, developed previously by the authors. The framework integrates CDSSs, such as Artificial Neural Networks (ANNs), Case-Based Reasoning (CBR) tools, and alert detection systems. The goal is to reduce medical errors, to support the physician’s decision-making process and to improve ultimately patient care. We applied SPE from the early design stages in order to ensure that the system will meet its performance requirements, and to identify possible solutions for relieving the performance limitations of this prototype system. The software performance evaluation is based on a layered queuing network model of the proposed web services-based infrastructure.

Categories and Subject Descriptors
J.3 [Medical Information Systems]
C.4 [Performance of Systems]: Measurement techniques, Modeling techniques, Performance attributes

General Terms
Performance, Design, Experimentation, Management.

Keywords

1. INTRODUCTION
The use of artificial intelligence-based clinical decision support systems (CDSSs) to provide clinical support in high-risk medical environments, such as neonatal care, is becoming increasingly common. As a growing body of research supports the use of CDSSs, such as Artificial Neural Networks (ANNs), Case-Based Reasoning (CBR) tools, and alert detection systems, in order to reduce medical errors and support the physician’s decision-making process [5], the relevance of offering such CDSSs as services within the Hospital Information System (HIS) becomes apparent. Such services will be accessed at first locally, and eventually from remote locations via wireless communication.

1.1 Medical Perspective
With the goal of offering physicians a service-oriented architecture for clinical decision support, our team is currently designing a web services-based infrastructure for integrating and accessing CDSSs and medical databases from distributed medical domains. The rationale for integrating distributed databases is that different medical domains often exhibit complementary abilities in terms of predicting medical outcomes. Obstetrical, perinatal and neonatal databases are examples of complementary predictive medical domains; where perinatal care is the care of a fetus or newborn given before, during, and after delivery. From the physician’s perspective medical outcomes of interest include: identifying factors predicting pre-term labour; determining necessity of neonatal intensive care; predicting outcomes such as mortality in the Neonatal Intensive Care Unit (NICU); and exploring indicators of cesarean birth.

Previously, our group developed an XML-based framework for medical data interoperability and the integration of CDSSs and of customizable XML-based clinical alerts into the Neonatal Intensive Care Unit [3], [4]. The predictive outcomes of our ANN and CBR tools have also been extensively investigated [5]. Current research extends the earlier XML integration framework, to support web service-based clinical decision support systems applied to distributed medical domains within the HIS. In this instance, the web services have multiple purposes: a) an application integration technology (for hybrid ANN and CBR systems), b) a means of linking distributed databases running on different underlying systems, and c) a means of making CDSS applications available locally through the HIS and remotely to authenticated wireless users. Presently, a prototype web services infrastructure called OPNI-Web is being developed. It will be implemented at the Children’s Hospital of Eastern Ontario (CHEO), integrating obstetrical, perinatal and neonatal domains.

In general, web services are categorized as being either core or composite web services. A core web service offers basic functionality that will be invoked by multiple higher-level applications. Composite web services represent higher-level applications, which combine two or more core services to offer a complete system composition scenario as seen from the user’s perspective. Determining appropriate service composition scenarios for each individual patient case and physician’s application selection is essential to OPNI-Web’s development. From the physician’s perspective three kinds of major composite web services can be invoked via the OPNI-Web user interface,
and applied to any of the three distributed patient data domains. These services include:

1. **Outcome Prediction**: After selecting an outcome for which the physician desires a prediction, the system invokes the appropriate trained ANN for this outcome of interest. Outcome Prediction invokes two core web services: **Trained AutoANN Processing** processes the data based on the retrieved optimal minimum data set, and **Replace Missing Values** uses a hybrid CBR-ANN system to replace missing patient data variables required for effective ANN outcome prediction. **Trained AutoANN Processing**, will invoke **Alert Detection** if the ANN output predictions exceed pre-specified physician-set thresholds.

2. **Matching Cases**: When the physician invokes **Matching Cases**, the case-based reasoning tool matches an individual patient’s condition, often of a newly arrived patient, to the most similar past cases.

3. **Alert Generation**: If desired, the physician may invoke the **Alert Generation** service that activates the core service, **Alert Detection**, associated with the ANN outcome prediction functionality. Because web services can be accessed from remote devices this means that once physicians are registered in the OPNI-Web system, they can receive web service-based remote alerts, possibly via wireless devices.

### 1.2 Performance Perspective

Potential performance limitations in providing real-time CDSS output to physicians constitute a limiting factor to OPNI-Web’s successful implementation. Therefore, studying these possible limitations prior to implementation is necessary. One of the main motivations for studying the performance characteristics of such a web services-based infrastructure as a function of workload is due to the fact that the number of physicians using hand-held devices, such as Personal Digital Assistants and web-enabled cell phones, is increasing. This work offers a case study applying SPE [11], [12] to a web service-based infrastructure offering clinical decision support (OPNI-Web), describes our experience in employing layered queuing networks [15] to perform predictive software performance evaluation of this prototype system, and suggests possible solutions to relieve identified performance limitations of the system.

An important objective of this case study is to use realistic performance measures for the demands for different system resources made by the tasks being performed, in order to predict the overall capacity and performance of an actual implementation of this web services-based system, and to identify and mitigate potential bottlenecks. The paper also discusses how the resource demands estimates were obtained. The performance model presented in this work incorporates both our XML-based health care integration framework [3] and our established ANN and CBR clinical decision support tools [4]. What is new is that these CDSSs are offered as web services. In the paper, only a representative subset of the complete web services infrastructure will be modeled. The subset models a key system scenario invoking one CDSS as a core web service and accessing the patient’s Electronic Patient Record (EPR). In terms of scalability, future analysis will evaluate more complex scenarios including complex web service composition interactions and accessing multiple databases distributed across the HIS.

### 2. THE CASE-STUDY SYSTEM

The system analysis considers both functional and non-functional requirements (such as performance). Before analyzing these requirements in more detail, a brief introduction to web services is presented.

#### 2.1 Web Services Overview

Web services constitute an emerging technology for which potential applications are unlimited. In simplest terms: “if a provider can imagine a way of delivering something of value to a customer to provide some usefulness… they have a viable web service” [14]. Web services are accessed via ubiquitous web protocols and data formats, such as Hyper Text Transfer Protocol and XML, respectively, rather than via object model specific protocols such as Distributed Component Object Model (DCOM), Remote Method Invocation (RMI), or Internet Inter-ORB Protocol (IIOP) [6]- from a performance perspective this changes the view of ‘middleware’ in the system. Web services are invoked over a network, however they do not have to reside on the World Wide Web; they can be located on an Intranet, or anywhere on the network. One of the key ideas is that a web service’s implementation and deployment platform are not relevant to the application that is invoking the service. There are three key components of web service systems:

- **Wire**: Comprises all technologies required to transport a service request from client to server; including XML for message encoding, and the Simple Object Access Protocol (SOAP) for handling data transmission capabilities.

- **Description**: A web service interface provides a collection of operations accessible through standardized XML messaging. This interface is described using the Web Services Description Language (WSDL), which specifies the operations provided by a web service, including the kinds of objects that are expected as input and the output of the operations [6].

- **Discovery**: The service requestor discovers the web service via discovery agencies that allow service descriptions to be published and discovered. From a performance perspective this involves the time to look up the service in the web services directory using Universal Description Discovery and Integration (UDDI). It is also possible to assume a more simplistic approach, where the client has prior knowledge of the service address. Since the system only allows access to a predetermined pool of authenticated users, this seems reasonable at the prototype level.

#### 2.2 Functional Requirements

From a functional perspective, the system must complete the following main tasks:

1. Invocation of clinical decision support systems by web service client (includes exchange of Web Service Definition Language document with web service provider)

2. Transmission of web service requests to the web service provider
3. Data validation (to ensure medical data to be processed by
the CDSSs is within acceptable ranges) using XML parsers
and predefined XML schemas for medical data
4. Pre-processing of XML data based on eXtended Stylesheet
Language (XSL) transformations, to transform data into a
format suitable for clinical decision support system
processing
5. Processing of data by clinical decision support systems,
either ANN or CBR applications
6. Transmission of result to web service client and display

2.3 Non-Functional Requirements
In terms of non-functional requirements, quantitative performance
requirements are defined for the system. These are user-oriented
and based on physician’s requirements. The defined performance
targets include the system capacity in terms of the number of
simultaneous users, and the average response time for a particular
user. Of these requirements the average response time is
paramount. Since physicians require real-time outputs,
unacceptable delays will hamper the decision-making process and
render the inclusion of CDSSs detrimental rather than beneficial.

When specifying the system capacity, we consider a scenario
where users from different medical domains within the HIS
(potentially scalable to include more than the three medical
domains originally considered in this work) would be
simultaneously accessing the web service-based CDSS. These
users are highly specialized, and even considering the possibility
of remote wireless users, will likely remain a select group.
However, it must be remembered that in the case of the Alert
Generation web service, other allied health care workers, such as
nurses, would also be end users of the system. In this case the
system capacity would need to be increased. To meet these
demands an initial conservative estimate of 50 simultaneous users
appears reasonable. Response times for these users, for basic
CDSS invocation and generation of a result should not exceed 10
seconds.

In web service performance modeling, the following operations
have to be taken into account, as they may have high resource
demands:

- Extracting the XML-defined web service request from the
  Simple Object Access Protocol (SOAP) envelope (conversely, at the client side this involves packing the XML
  request in the SOAP envelope); also have to consider
  packing and unpacking the output response.

- Parsing the XML document. There are different types of
  parsing models and they differ in computational efficiency
  [6]. In pull parsers the application has to ask the parser to
give it the next piece of document information; this style of
parser has the highest efficiency. One-step parser
applications, where the parser reads the whole XML
document and generates a parse-tree), are the least efficient
parsing option.

2.4 Deployment View
The deployment diagram illustrated in figure 1 depicts the system
architecture. All nodes are connected through the HIS Intranet.
Initially, the assumption is made that each server runs on a
separate processor; multi-processor usage will be considered
during the performance analysis.

Figure 1. Deployment of the web services infrastructure (OPNI-Web)

3. APPLYING SPE TO OPNI-WEB
A growing body of research advocates performing Predictive
Software Performance Evaluation (SPE) early in the design phase;
the idea being that even when performance metrics are based on
high-level designs and rough estimates of CPU demand
parameters, potential problems are identifiable [8], [9]. In this
sense, SPE can be thought as an indicative tool to point to
performance problems within the system’s architectural design,
prior to actual implementation. Although the initial system
architecture may not present a completely accurate picture of the
performance of the implemented system, the incomplete
performance model provides enough information to draw some
initial and useful conclusions. In this case study, these
conclusions will be used to evaluate the feasibility, from a performance standpoint, of implementing web service-based CDSSs in the NICU.

3.1 Model Parameter Estimation

The workload parameters are the attributes of the system that define the load on its resources [9]; these parameters will be used to create the Layered Queuing Network model used to evaluate the system’s performance.

3.1.1 Determining the System Workload

Real-time requests for CDSS web services can arrive from multiple allied health-care professionals either locally or remotely. Remote requests may arrive via wireless devices from physicians outside of the hospital information system (HIS). Since in the first development stage of OPNI-web only local users will be allowed, the performance requirements associated with mobile access will not be modeled in this study. However, when estimating the maximum number of users the system should support, we take into account the fact that remote users will be added to the system in the future. As already mentioned, we estimate that the system will be used by at most 50 simultaneous users.

3.1.2 Obtaining Performance Parameters

Currently, there is considerable interest in the literature regarding XML performance benchmarks, with published benchmark results for common manipulations of various sizes and types of XML documents [1], [2], [7], [10], [13]. Based on these benchmarks, table 1 provides response time for the different tasks participating in figure 2’s sequence diagram. The benchmark values for XML encoded SOAP documents were obtained from [13]. However, the encoded electronic patient record (EPR) transmitted in this study is much smaller in size than the example document studied in the benchmark, so a ratio was used to determine the average response times for an EPR of approximately 5 KBytes as opposed to the average response time of a similarly typed 20 KByte SOAP1.xml file (heavy on namespaces and attributes).

However, one of the difficulties in obtaining performance measures pertains to determining metrics for the data retrieval. According to [1]: “We observed large differences in query response times between different systems even in single-user mode, e.g. ranging from 10 ms to several seconds on a small database. In multi-user mode the 90th percentile response time limit, which we set to 3 seconds for most queries, was often missed.” This variability makes it difficult to obtain an accurate metric without testing the native XML document retrieval in a particular implementation. One option would be to function on the worst case metrics, however if these are unrealistically high, a false software bottleneck may be introduced.

Table 1. Performance parameters for XML operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Mean Execution Time/ Document Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Document build time: Time required to parse a text document and construct the document representation</td>
<td>100 ms / 20 K</td>
</tr>
<tr>
<td>Parsing for a SOAP document: packing XML envelope</td>
<td>25 ms / 5 K</td>
</tr>
<tr>
<td>Document Walk Time: Time required to walk the constructed document representation, going through each element, attribute, and text content segment in document order.</td>
<td>20 ms / 20 K</td>
</tr>
<tr>
<td>XML validation: Time required to validate the XML document</td>
<td>5 ms / 5 K</td>
</tr>
<tr>
<td>Document Modify Time: Time to update the XML-based EPR with the new CDSS result</td>
<td>50 ms / 20 K</td>
</tr>
<tr>
<td>XML parsing:</td>
<td>12.5 ms / 5 K</td>
</tr>
<tr>
<td>Text generation Time: Converting the XML document into a format suitable for the ANN</td>
<td>75 ms / 20 K</td>
</tr>
<tr>
<td>XML processing:</td>
<td>18.75 ms / 5 L</td>
</tr>
</tbody>
</table>

Benchmarking values for native XML databases depend on whether an XML document conforms to its schema. If this is not the case, additional error reporting overhead is incurred. Values also depend on whether query operations are expressed using XML language proposals (e.g. XQuery) for the native XML database queries. One of the main performance problems with native XML databases is that “XML data is often highly redundant so that XML data management systems should be able to limit the space requirements for document storage. However in our tests the databases occupied 3-8 times the space of the source data.” [1]. Table 2 provides benchmarking values for a native XML document storage system based loosely on the XOO7 benchmark, “an XML version of the O07 benchmark enriched with relational, document and navigational queries that are specific and critical for XML databases [2]”.

Table 2. Performance parameters for database operations

<table>
<thead>
<tr>
<th>Operation (database retrieval of initial 5 K file)</th>
<th>Mean Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native XML database query: XML databases store data similarly to hierarchical and network-structured databases. For queries that want such a hierarchical structure, native XML databases can be considerably quicker than an RDBMS, which has to construct the XML document as output. The XML-based file will be larger than 5 K in database ~ 15 K</td>
<td>50 ms</td>
</tr>
<tr>
<td>Relational database query (based on mySQL): Retrieval times for RDBMS queries vary drastically depending on the complexity of the query relative to the structure of the database. A simple query on a single table might go as low as 5ms, and less in a highly optimized database once the database connection has been made. If the query results in relational joins between tables, which is quite likely since relational databases are flat structures and XML represents a hierarchical structure, then performance values could be significantly worse than 5ms.</td>
<td>10 ms</td>
</tr>
</tbody>
</table>

3.2 Annotated Sequence Diagram

Combining the performance metrics from the previous section with additional performance information, such as the user think time, the network transmission delays for the HIS Intranet’s...
10Mbps Ethernet over TCP connection, and the CDSS processing times, yields the annotated sequence diagram shown in figure 2. The sequence diagram illustrates a scenario encompassing the entire functionality for web service invocation. This analysis is based on Intranet communication, such as would be present in a Hospital Information System (HIS), where all users have secure and authenticated access to the patient’s Electronic Patient Record (EPR).

4. LAYERED QUEUEING NETWORKS

The resource metrics and architectural design presented in the previous sections are used to build a layered queuing network (LQN) model, solved using the analytic solver, LQNS. Layered queuing networks are extended queuing models adapted for software resources, with queues for concurrent tasks [9]; in a LQN model, a server may become a client to other servers from which it requires nested services, while serving its own clients [8]. Tasks (depicted as bolded rectangles) offer services called entries (depicted as plain rectangles associated with the task), each with its own average execution times and demands for other services (demands are shown on the invocation arrow as the number of visits made to other nested tasks). Hardware devices that act as servers to the tasks are depicted as circles; each has an associated scheduling discipline. Infinite servers model tasks with pure delays, such as networks. Figure 3 presents the LQNS model for the web service invocation scenario, including all required concurrent tasks, entries, service demands and average execution times; note that all entry service times are in milliseconds.
5. PERFORMANCE ANALYSIS

The LQNS analytical model is used to predict the system response time for a single user, the overall capacity the system can achieve while still satisfying individual user’s performance requirements, and software and hardware bottlenecks. The analysis begins by presenting the original performance results, from which bottlenecks are identified. Replication of both threads and processors is introduced to mitigate these bottlenecks.

5.1 Initial Results

The system, in its original configuration, with no replication of either threads or processors, is not able to satisfy the system’s performance targets. Analyzing figure 4’s graph indicates that response times of 10 seconds per user can only be achieved for a system capacity of 10 users or less. At 50 users the response time is 39.9 seconds, well above the accepted limit.

To identify system bottlenecks and improve response times, figure 5 presents utilizations for the most saturated tasks and their processors. Both the WSCoordinator and CDSSControl task are highly saturated, with maximum utilizations of 0.98 and 0.92 respectively. This initial result suggests that the bottleneck is caused by contention in the software, rather than the hardware; multithreading is introduced to mitigate the software bottlenecks.
5.2 Replication: Threads and Processors

Expanding on the initial analysis, the bottlenecked WSCoordinator task is replicated and analyzed for the goal capacity of 50 users. However, the addition of WSCoordinator threads alone does not result in significant improvements in response time. The response time stagnates at 36.5 seconds for 10 threads. In this configuration, both the WSCoordinator task and the CDSSControl task are completely saturated, with respective utilizations of 9.999 and 0.996. This result suggests that the CDSSControl task should also be multi-threaded, while the number of WSCoordinator threads is fixed at 10. Figure 6 illustrates that the maximum improvement in response time is obtained using 4 threads, with a response time of 21.65 seconds.

![Figure 6. Replicating CDSSControl tasks](image)

In the above configuration, the CDSSControl task’s processor, Applic_CPU, is completely saturated. However, the WSCoordinator’s processor, WSP, remains lightly loaded. This suggests a possible hardware bottleneck at the level of the Applic_CPU processor. Figure 7 shows the effect of replicating the Applic_CPU, while varying the number of CDSSControl threads. An optimal response time of 10 seconds is obtained using 3 Applic_CPU processors and 10 CDSSControl threads (the WSCoordinator threads remain fixed at 10). The analysis also indicates that regardless of the number of available Applic_CPU processors, the processor utilization never exceeds 2.51, leading us to conclude that configurations with greater than 3 processors are ineffective.

![Figure 7. Replicating Applic_CPU processors](image)

5.3 Sensitivity Analysis

Intuitively, there are two potentially significant factors that may impact the results obtained in the previous section: the web service processing time (e.g. in this case the processCDSS entry’s service time) and the number of visits made to the EPR Database. Both factors are variable and depend respectively on the selected web service’s processing time, and on the size of the XML-based record that must be read and updated before and after web service processing.

As expected, the response time increases with increased service times. The results are not initially dramatic: decreasing the service time of the processCDSS entry from 500 to 100 ms results in a response time of 5.024 seconds, as compared to 5.482 seconds. However, a service time of 2000 ms yields a significantly greater response time of 10.98 seconds.

![Figure 8. Replicating EPRT threads](image)

Varying the number of database reads and writes also effects the response time. Multiplying the original number of database visits (12 for database writes, and 8 for reads) by a multiplicity factor of 0.5, 1, 1.5 and 2, generates the exponential curve shown in figure 10. Figure 10 indicates that when the number of database visits doubles, the response time increases from 5.48 to 9.16 seconds. These results confirm the intuitive hypothesis that the optimal response time is sensitive to both the web services processing time and the number of database visits.

![Figure 9. Impact of web service processing time](image)
6. CONCLUSION

This paper presented a case study, outlining the performance modeling and analysis of a web services-based CDSS infrastructure, based on layered queuing networks. Analyzing the LQNS analytic model, demonstrated that it is possible to achieve the system’s initial performance targets: satisfying a capacity of 50 users with response times of less than 10 seconds per user. Based on this required capacity, an optimal response time of 5.4 seconds was obtained. The optimal response time is sensitive to both the web service processing time and the size of the electronic patient records.

7. ACKNOWLEDGMENTS

The authors wish to acknowledge the support of the Natural Sciences and Engineering Research Council (NSERC) of Canada’s Post Graduate Scholarship program.

8. REFERENCES


