

XML in a Multi-Tier Java/CORBA Architecture

Mehran Moshfeghi¹, Bart de Greef²

Philips Research¹, Philips Medical Systems², Mountain View, CA

Email: [mmm,degreef]@medgrid.philips.com

Abstract

We demonstrate three different methods of using XML in a multi-tier Java/CORBA system:

- (1) Personalized display of XML content with stylesheets.
- (2) A dynamic XML table-of-contents for an on-line help system.
- (3) Utilizing XML to represent and enforce valid request patterns for performance simulations.

These first two examples deal with the display of well-formed XML content, but do not touch on XML validity. The third example shows how XML representations of usage patterns can be validated against a DTD that specifies allowed usage patterns.

1. Introduction

XML separates structure and content from how it is presented. Once the eXtensible Style Language (XSL) [1] and other stylesheet specifications for XML [2] are finalized and tools become available it will be possible for a user to select different stylesheets in order to change the view on the data. This could be carried out locally on the client without having to go back to the server to download the content again. In section 2 we provide an overview of a Java and Common Object Request Broker Architecture (CORBA) based Electronic Medical Record (EMR) system which we have developed. In section 3 we describe the method and present results for personalizing the display of XML electronic medical record documents with SGML-like stylesheets. In section 4 we present the use of XML for the on-line help system of our EMR system, where the use of XML and Java makes the table-of-contents dynamic.

One of the main benefits of XML over HTML is that XML is extensible and can define new elements (commonly referred to as tags) and syntax rules for the elements. A Document Type Definition (DTD) is a set of rules that specifies which elements are allowed in a document, the order in which elements can appear, which elements can appear inside others, which elements have

attributes, etc. In section 5 we describe a method for using XML to represent system usage files for load testing experiments. We present an XML DTD that we designed to specify allowed usage pattern rules for our Java/CORBA-based EMR system. XML usage pattern files are validated against this DTD before they are allowed to load the EMR system.

2. Overview of EMR System

We have designed a multi-tier client-server system that uses CORBA's distributed object technology to dynamically create views of the multimedia patient record [3], and have called it MIRACLE (Medical Information Retrieval Application for Clinical Enhancement). The three tiers of MIRACLE are the clients, middleware servers, and back-end applications, as shown in Figure 1. The clients, the middleware servers, and some of the back-end systems are completely written in Java and can run on multiple platforms without modification. The first tier is formed by the Java client applications that reside and run on a variety of computers. The clients can also run as applets in Java-enabled browsers.

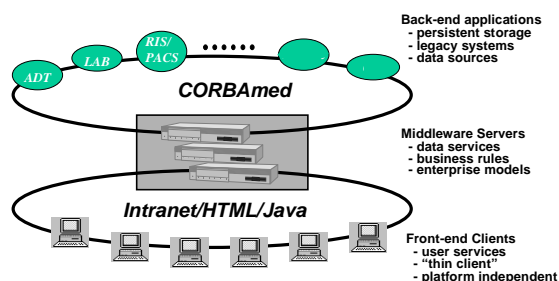


Figure 1. Overview of MIRACLE.

The second tier consists of middleware servers. The server objects implement the business logic. A client sends CORBA requests through the run-time Object Request Broker (ORB) to these servers. The servers connect to persistent back-end data stores in the third-tier and provide the client with an integrated view. We use CORBA's Interface Definition Language (IDL) to define the interfaces between the client and the middleware

servers, and the interfaces between the middleware servers and the back-end applications.

The third tier consists of legacy back-end systems that have been CORBA-wrapped, and new persistent data stores that have been designed with CORBA interfaces from the start. Typical legacy systems include laboratory systems, hospital information systems, radiology information systems and Picture Archiving and Communication Systems (PACS). In the next section we describe how we present XML documents with attached stylesheets inside MIRACLE.

3. Personalization with Stylesheets

Figure 2 demonstrates the structure of an XML document by displaying its tree diagram. The document depicts a patient's medical record and consists of a table of laboratory results, a cardiac x-ray image, and some demographics information. The laboratory results consist of several clinical observations, each of which has a test name, result value, unit of measurement, and lower and upper range. The image is in gif format and is referenced as an external entity. The demographics information consists of the patient's names, address, home telephone, driver license number, social security number, employer information, occupation, insurance information, etc.

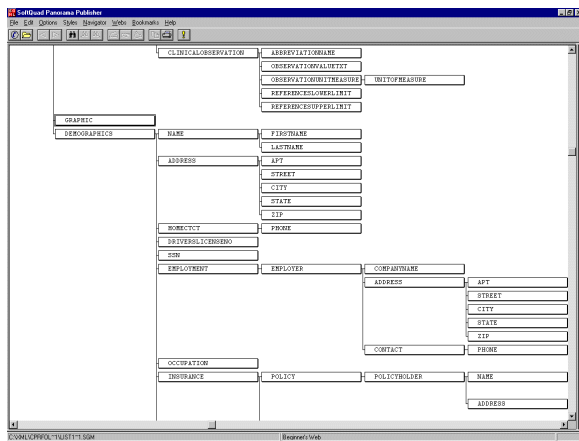


Figure 2. Tree diagram of an XML document.

There are several mechanisms by which the client can obtain such a representation. A server-centric method is that the middleware servers access back-end system to obtain the data, format the data into XML if necessary and provide the XML document to the client for display. Another option for a Java-based system such as ours is for the client to use Java's Reflection APIs to transform Java objects into XML documents at the client-side [4-6].

We have managed to display XML documents with an

SGML tool, Panorama Publisher 2.0 from SoftQuad and Interleaf. Suppose that the XML document in Figure 2 is called "record.xml". If the document's name extension is changed from .xml to .sgml (i.e. "record.sgml"), this tool will think that it is about to display an SGML document. It will first display a harmless message that it can not find a DTD, but will continue to display the document. Panorama Publisher can display XML files inside the Netscape browser because it can be configured as a Netscape helper application. The next generation web browsers will have built-in support for XML parsing and display. We had to use Panorama Publisher because of the lack of stylesheet standards for XML and the unavailability of XML tools when this work was carried out.

One of the design issues in tagging a document is to separate the data from presentation aspects. Stylesheets provide mechanisms for assigning display properties to elements, thereby enabling the separation of structure from display. We used the stylesheet language of Panorama Publisher to specify the presentation characteristics of XML documents. Panorama Publisher's stylesheet provides display properties for fonts, paragraphs, text labels and tables.

Figure 3 shows the result of attaching two different stylesheets to the XML document whose tree view is shown in Figure 2. Consider the display in Figure 3(a). The stylesheet for this display is responsible for text labels like "Lab Results", "Cardiac Image" and "Demographics Information". There is no mention of these labels in the XML document itself. The font properties and the section and table formatting are also specified in the stylesheet. For example, the "-" separator between low and high range values in the laboratory results table is specified in the stylesheet.

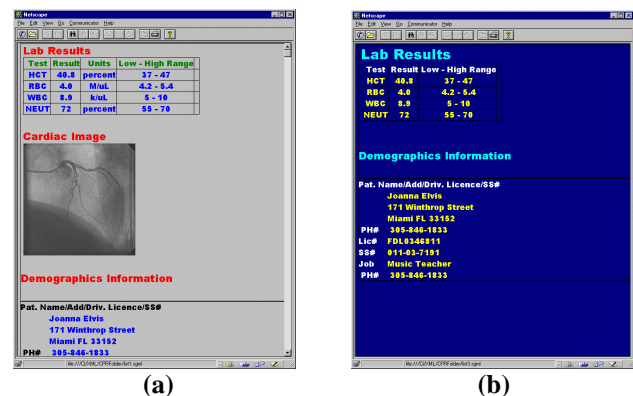


Figure 3. Result of attaching two different stylesheets to the XML document of Figure 2.

Now consider the display in Figure 3(b). The stylesheet for this display has assigned different properties for fonts

and colors. This stylesheet is also using the stylesheet's hide feature to hide the display of the "Units" column of the laboratory table, the cardiac image, and some sections of the demographics information. This demonstrates how stylesheets enable users to personalize the way they view XML documents at the client. Users can dynamically change stylesheets and see a different view without having to go back to the server. This capability will be provided in the next generation web browsers once the stylesheet specifications for XML are finalized [1,2].

In this section we presented personalized display of XML documents inside our Java/CORBA application. In the next section we demonstrate the use of XML for a dynamic table-of-contents in our application's on-line help.

4. XML in the On-line Help

Figure 4 shows the on-line help system of MIRACLE. The system uses Sun's Early Access Release 2 of JavaHelp [7]. MIRACLE's help viewer displays a table-of-contents (TOC) on the left and a content pane on the right. The table-of-contents provides collapsible or expandable display of help topics. The file format for the TOC is XML, while the content pane displays HTML 3.2 help topics.

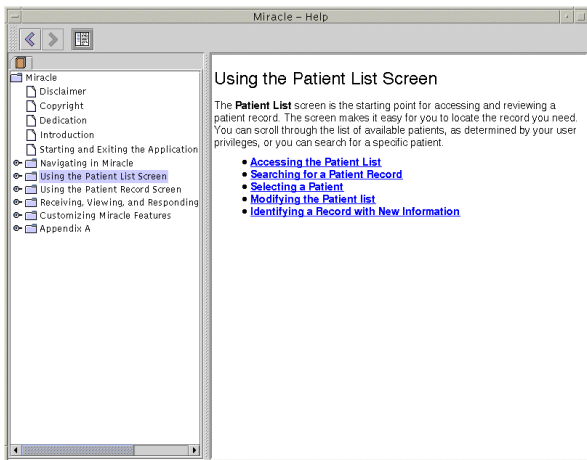


Figure 4. MIRACLE's on-line help, with the table-of-contents on the left, and the content on the right.

MIRACLE's HelpSet file, which is shown in Figure 5, is first read when a user chooses the MIRACLE help menu. The JavaHelp system derives all the information it needs from this HelpSet file. The format of this file is XML.

The <helpset> tag is the root element that defines the HelpSet. The <title> tag names the HelpSet. The <homeID> tag specifies the default home that is displayed when the help viewer is called. The <map> file specifies the name(s) of the map file(s) used in the HelpSet, and the <data> tag specifies the URL of the map file(s). In this case it is a relative local file called Map.map. The map file is used to associate topic IDs with URLs so that the paths to HTML topic files are specified.

```
<?xml version='1.0' encoding='ISO-8859-1'
standalone='yes' ?>
<helpset>
  <title>Miracle - Help</title>
  <homeID>top</homeID>
  <map>
    <data>Map.map</data>
  </map>
  <view>
    <name>TOC</name>
    <label>Table Of Contents</label>
    <type>javax.javahelp.TOCView</type>
    <data>MiracleHelpTOC.xml</data>
  </view>
</helpset>
```

Figure 5. MIRACLE's HelpSet file for on-line help.

The <view> tag defines the navigators used in the HelpSet. The TOCView navigator is used in this example. Other available options are the index or search navigators. The <view> tag contains other tags such as <name> for the name of the view, <label> for the display label on the navigator tab, <type> for the path to the navigator class, and <data> for the path to the data used by the navigator. In this example the data used by the navigator is a local file called MiracleHelpTOC.xml.

Figure 6 shows a section of the MiracleHelpTOC.xml TOC file. The format of this file is also XML. This file specifies the TOC of Figure 4. The <toc> tag defines the TOC and can contain <tocitem> tags. The latter defines a TOC entry. The <tocitem> tags can be nested for hierarchical TOC entries. The <tocitem> tag uses a target attribute to specify the target to display when the user chooses the entry. The target IDs are defined and associated with a URL in the map file. The use of Java and XML means that the client can expand, collapse, and navigate the TOC without going back to the server to generate a new view each time.

```

<?xml version='1.0' encoding='ISO-8859-1'
standalone='yes' ?>
<toc>
<tocitem>Miracle
  <tocitem target="Chapter0.Disclaimer">
    Disclaimer</tocitem>
  <tocitem target="Chapter0.Copyright">
    Copyright</tocitem>
  <tocitem target="Chapter0.Dedication">
    Dedication</tocitem>
  <tocitem target="Chapter1.Intro">
    Introduction</tocitem>
.....
</tocitem>
</toc>

```

Figure 6. A section of the table-of-contents (TOC) file for MIRACLE’s on-line help.

It should be noted that while JavaHelp uses the XML syntax, its element names are pre-specified and fixed at this time. Future versions of JavaHelp will likely support the display of XML files in the content pane. This is currently awaiting the adoption of stylesheet standards.

In the previous two sections we demonstrated the display of XML documents and XML table-of-contents inside our application. In the next section we describe how we use an XML DTD to represent allowed usage patterns, and show an allowed request sequence that is represented as a valid XML file.

5. XML for Usage Patterns

One of our goals is to investigate the scalability of MIRACLE. It is important to measure the performance and scalability of large distributed enterprise systems, such as MIRACLE, and find potential bottlenecks before they occur out in the field. Load testing is the process where users are simulated in order to generate client activity and test the response of the system or application. System designers can use load testing to find out how many concurrent users a server can handle or how configuration changes affect system performance. Load testing requires simulating the usage patterns of typical users of the system.

In this section we describe the user interface and flow chart of MIRACLE. We then present an XML DTD that we designed to specify allowed usage pattern rules. We also present an example of a valid XML usage pattern file.

Figure 7 shows the patient chart interface of MIRACLE. The user has logged in, chosen a patient, and selected three lab results from the “Lab Results” tab pop-up list. The patient chart interface has ten tabs as seen in the left of Figure 7.

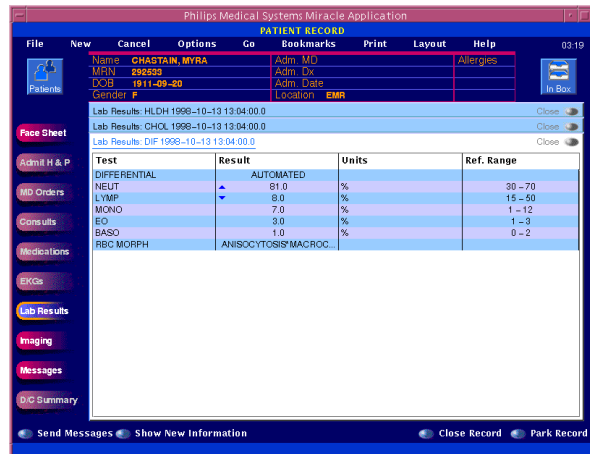


Figure 7. MIRACLE’s patient chart user-interface.

Figure 8 shows the flow chart for most of the features of MIRACLE. A session starts with a user successfully authenticating him/herself to the system and logging in. After login the user is presented with a list of patients that he/she is allowed to access. At this point the user can logout, use the toolbar menu, or enter the chart of a patient. A logout request ends the session. The toolbar menu commands allow the user to blank the application’s screen, customize the application’s settings, and add/remove application bookmarks.

After activating menu commands the user still has the option to use the menus, logout or enter the chart of a patient. If the user selects a patient from the patient list he/she enters the chart of that patient.

Figure 8 shows navigation for four of the 10 patient chart tabs in Figure 8: “Face Sheet” (for demographics), “Lab Results”, “Imaging” and “Messages”. All except the demographics tab present a list in a pop-up window because there can be more than one item of information under these tabs. The “Imaging” tab provides a list of reports as well as images. A user can select one item at a time from the pop-up lists to see the desired information. For example, the user in Figure 7 has selected three lab results from the “Lab Results” pop-up list by making three separate selections. Navigation inside a patient’s chart can continue with the same or other tabs. The user can also use the “Send Messages” icon at the bottom left-hand corner of the interface in Figure 7 to send patient-related messages to other healthcare professionals. The toolbar menu is also accessible from within the patient chart interface. The user can use the “Patients” icon on the top left-hand corner of the interface in Figure 7 to go back to the patient list and select another patient. The session ends with a logout command from the user or

from the application's automatic logout feature in case the user is inactive for a long time.

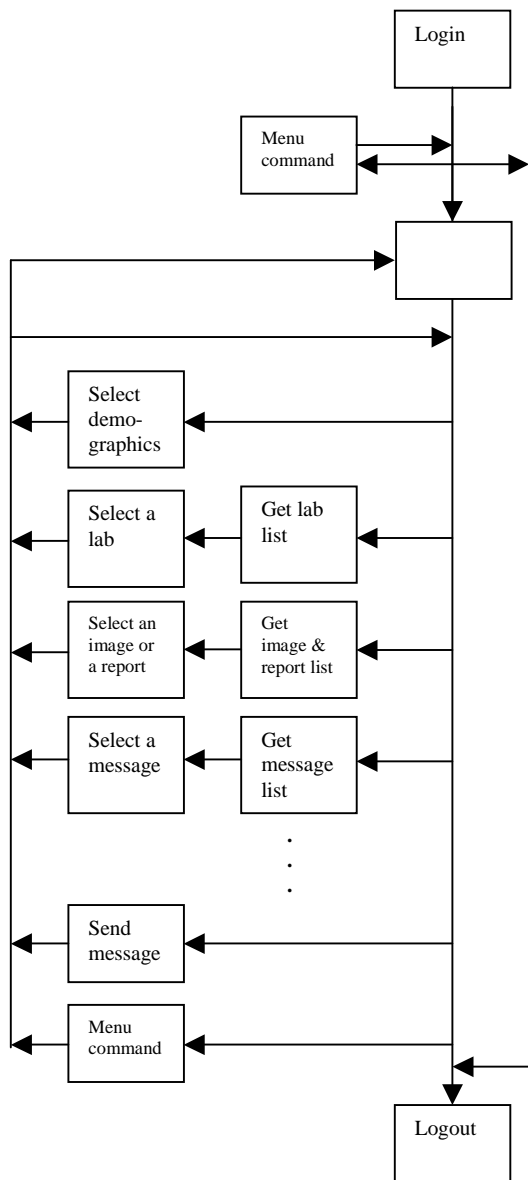


Figure 8. Flow chart of MIRACLE's navigation rules.

5.1. XML DTD for Usage Pattern Rules

Figure 8 illustrates navigation options inside MIRACLE. Some of the blocks in Figure 8 generate multiple requests to the middleware servers, while others do not generate any requests and merely change the presentation states of the GUI. We will now describe the blocks of Figure 8 in more detail and use the XML DTD shown in Figure 9 to formally state the request syntax rules.

In Figure 9 elements with upper-case names represent client requests made to the middleware servers, while elements with lower-case letters represent convenience or container tags. For example, the `<sleep>` element with the required `duration` attribute is a tag that is used to indicate that the client is sleeping for `duration` seconds before sending the next request. This way, user think time" can be included in the simulation.

The `<session>` element is a tag that represents a session as a `<login>` element, followed by zero or more occurrences of `<menu>` and/or `<inside_chart>` elements, followed by a `<logout>` element. After a user is logged in five requests are automatically generated and sent to the middleware one after another: a login request, a request to get the user's profile, a request to get the user's bookmarks, a request to get the user's list of new information, and a request to get the user's list of patients. Thus, the `<login>` element in the DTD of Figure 9 contains these five request elements. The `<logout>` element has only one logout request.

The menu commands allow a user to blank the application screen, change the user's profile, and get/add/delete bookmarks. Thus, the `<menu>` element contains one of these requests. The `<inside_chart>` element signifies that the user is inside a patient's chart. This element can contain zero or more occurrences of `<menu>`, `<get_demographics>`, `<get_labs>`, `<get_image_report>`, `<get_messages>`, and `<send_messages>` elements.

The `<get_demographics>` element contains one request for getting the patient's demographics. The `<get_labs>` element can contain an optional request for getting the list of lab procedure items, followed by a request for a lab result observation item. The first request for the list is optional because after a user clicks on any of the tabs with a pop-up list, the list is cached. Thus, the next time the user clicks that tab the request for the list is unnecessary.

The `<get_image_report>` element contains the optional request for the list of images and reports, followed by either a report request or a number of requests for images (one for the header and one for every image in the series). The `<get_messages>` element contains an optional request for the list of messages, followed by a request for a message item from the list. The `<send_messages>` element contains two requests, one for getting the pick-list of doctors that are allowed to receive messages, and one send message request.

```

<!ELEMENT session
  (login, menu*, inside_chart*, logout)
>
<!ELEMENT login
  (sleep, USER_LOGIN, USER_GET_PROFILE,
   BOOKMARK_GET, NIA_GET_INCR_UPDATE,
   PATIENT_LIST_GET)
>
<!ELEMENT logout
  (sleep, USER_LOGOUT)
>
<!ELEMENT menu
  (sleep, (USER_BLANK_SCREEN |
   USER_SET_PROFILE | BOOKMARK_GET |
   BOOKMARK_SET | BOOKMARK_DELETE) )
>
<!ELEMENT inside_chart
  (menu | get_demographics | get_labs |
   get_image_report | get_messages |
   send_messages )*
>
<!ELEMENT get_demographics
  (sleep, RR_PATIENT_DEMOGRAPHICS)
>
<!ELEMENT get_labs
  (sleep,
   (IDR_GET_PROCEDURE_LIST,sleep)?,
   IDR_GET_PROCOBV_LIST)
>
<!ELEMENT get_image_report
  (sleep, (RR_GET_IMAGE_REPORT,sleep)?,
   (RR_FULL_REPORT |
   IMAGESET_IMAGES,IMAGESET_IMAGES+)) )
>
<!ELEMENT get_messages
  (sleep,
   (MESSAGE_LIST_BY_USER_PATIENT_GET,
   sleep)?,
   MESSAGE_LIST_DETAIL_BY_USER_GET)
>
<!ELEMENT send_messages
  (sleep, DOCS_FOR_PATIENT, sleep,
   MESSAGE_SEND)
>
<!ATTLIST sleep
  duration CDATA #REQUIRED
>
<!ELEMENT sleep
  (#PCDATA)
>
...

```

Figure 9. Portion of the DTD for representing MIRACLE's navigation rules.

5.2. XML Usage Pattern File

Figure 10 shows an XML usage session file. The file references the "Requests.dtd" DTD file that is shown in Figure 9, and validates correctly against it. The user's session consists of logging in, navigating inside the chart of a patient and logging out. The login and its associated 5 requests occur at time 0. The first request inside the chart of a patient is for demographics information and occurs 15 seconds later. After another 20 seconds the user requests the list of lab results and after 3 more seconds the user selects an item from the list. The user selects another

lab result after 6 more seconds. Note that the list request is not required the second time because the list is already cached. The user requests to see the list of images and reports after 12 seconds, and selects a report from the list 5 seconds later. The user logs out after another 7 seconds.

```

<?XML version="1.0"?>
<!DOCTYPE session SYSTEM "Requests.dtd">
<session>
  <login>
    <sleep duration="0"/>
    <USER_LOGIN/>
    <USER_GET_PROFILE/>
    <BOOKMARK_GET/>
    <NIA_GET_INCR_UPDATE/>
    <PATIENT_LIST_GET/>
  </login>
  <inside_chart>
    <get_demographics>
      <sleep duration="15"/>
      <RR_PATIENT_DEMOGRAPHICS/>
    </get_demographics>
    <get_labs>
      <sleep duration="20"/>
      <IDR_GET_PROCEDURE_LIST/>
      <sleep duration="3"/>
      <IDR_GET_PROCOBV_LIST/>
    </get_labs>
    <get_labs>
      <sleep duration="6"/>
      <IDR_GET_PROCOBV_LIST/>
    </get_labs>
    <get_image_report>
      <sleep duration="12"/>
      <RR_GET_IMAGE_REPORT/>
      <sleep duration="5"/>
      <RR_FULL_REPORT/>
    </get_image_report>
  </inside_chart>
  <logout>
    <sleep duration="7"/>
    <USER_LOGOUT/>
  </logout>
</session>

```

Figure 10. A valid XML usage session

All client requests to the middleware have an attribute or set of attributes. For example, the requests for the user's profile and bookmarks require the user-id name. The requests for a patient's demographics information or list of labs/images/reports require the patient-id. The request for a specific report requires the report-id. The request for an image series requires a data structure parameter that contains information such as an imageset-id as well as width, height, number of images and bits-per-pixel. For simplicity we have not included these parameter in Figures 9 and 10. If these request parameters are not specified our load simulation program will retrieve the first item it finds in a list.

We have developed a "crawler" program that finds all patients and their information parameters for all

authorized clinical users. We use a subset of the crawler's findings to simulate users and send specific requests to MIRACLE's middleware servers. The request patterns of these virtual users are similar to recorded usage patterns of physicians in a clinical trial of MIRACLE [3]. We represent the sessions of the virtual users with XML files similar to those in Figure 10, but with the request parameters added. All request parameters, including data structures, are converted to strings. These XML files are first validated to make sure that the simulated session requests conform to the navigation rules of MIRACLE. The requests of validated files are then used to load test the system.

6. Discussion

We have presented results of using XML for three different aspects of a Java/CORBA multi-tier medical record application. We presented personalized display of XML documents inside the application, and demonstrated the use of XML for the application's on-line help. These two examples demonstrated the display of well-formed XML content, but did not touch on XML validity. In the third example, however, we described the use of an XML DTD in representing allowed usage patterns, and showed how XML file representations of usage patterns can be validated against the DTD.

There are efforts to develop medical industry-specific vocabularies and tags [8]. When these XML vocabularies are agreed upon XML can also be used for intelligent searching and structured reporting. The extraordinary growth of the Internet and XML as the next language for the web means that in the near future many transcription and document management systems will be XML-based [6]. XML is gaining popularity in some areas as a messaging format [9]. XML can also be used to pass the parameters to remote procedure method calls [5], thereby fixing the method signature contract but allowing flexibility in changing the method parameters.

7. Acknowledgements

The development of MIRACLE would not have been possible without the contribution of several of our colleagues at Philips. We also had close cooperation with CareFlow|Net, Inc. for interfacing to legacy systems.

8. References

[1] "Extensible Stylesheet Language (XSL), version 1.0, World Wide Web consortium working draft 16-December-1998", <http://w3c.org/TR/WD-xsl>

- [2] "Associating stylesheets with XML documents Version 1.0, W3C Proposed Recom., 14 Jan. 1999", <http://www.w3.org/TR/PR-xml-stylesheet>
- [3] M. Moshfeghi, B. Kane, "Clinical installation and usability study of a Java and CORBA-based electronic medical record system", Submitted to AMIA Annual Fall Symp., 1999
- [4] "XML|IT: A tool kit to leverage distributed object infrastructures", June 1998, <http://www.careflow.com>
- [5] J.P. Morgenthal, "XML and Java", <Tag>'99 West, Feb 8, 1999, <http://tag99.com>
- [6] V.J. Jagannathan, S. Friedman, T. Davis, K. Wreder, E. Butler, Y. Alsafadi, M. Moshfeghi, "XML and transcription process automation", HIMSS 99 Panel presentation, Feb 1999
- [7] "JavaHelp home page", <http://www.javasoft.com/products/javahelp/>
- [8] "HL7 SGML/XML Special Interest Group", <http://www.hl7.org/> "SGML/XML SIG White Papers and Drafts",
- [9] <http://www.mcis.duke.edu/standards/HL7/committees/sgml/index.html#white>