Braid: Distributed Patient Information for Health Care Providers

Mike Rogers, Sheikh Ghafoor, Jeff Graves
Department of Computer Science, Tennessee Technological University, Cookeville, TN, USA

Abstract— *Sharing patient records among heath care providers improves the quality of patient care. Despite having electronic health record systems and advances in information technology, integrating patient record systems of health care providers can be difficult. This paper presents a configurable framework called Braid that enables health care organizations to exchange patient information without any modification of the individual organization's internal patient record system. The architecture of Braid is very configurable. Braid can accommodate changes in information exchange protocol, and addition and modification of the information services in an organization's internal patient record system. The paper also discusses a prototype implementation which has been tested with actual patient data.*

1. Introduction

Sharing medical records among health care providers improves the quality of patient care. However, tracking patient records of individuals in health care industries requires a high level of collaboration among many health care providers. For example, when obtaining the medication history for John Doe, more than one health care provider may need to be queried if John has switched practitioners, visited emergency wards while on vacation, is a patient of multiple specialist, etc. Not only does a practitioner need to know which health care providers that the patient has visited, the practitioner needs to contact all health care providers to retrieve the patient's medical records.

Traditionally, sharing medical record has been accomplished by physical means, such as having patients carry their records from one provider to another or via communication mechanisms such as telephone or fax. Unfortunately, such mechanisms are slow and error prone. Recently, health care providers have begun efforts to share electronic medical records via computer networks. Unfortunately, electronic medical records sharing is still rare because some significant barriers exist. Policy barriers, such as patient privacy, are beyond the scope of this paper. However, the following are some of the major technical issues that have kept electronic medical records sharing from becoming widespread.

This work is supported by the State of Tennessee, Tennessee Technological University, Vanderbilt University, and the Agency for Healthcare Research and Quality (AHRQ), Contract no. 290-05-0006. This publication has not been approved by AHRQ.

Native Systems: Many care providers in United States keep patient information electronically, which is an important initial step toward electronic medical records sharing. However, many of these native systems have been developed in-house, and others are commercially off-the-shelf products coming from various vendors. Most of these systems are not compatible with each other and are not designed to operate in a distributed manner.

Integration: Web services are quickly becoming an industry standard for B2B (Business-to-Business) communication because web services provide messaging and communication abstractions that allow businesses to share information. Web service messaging constructs and communication paradigms have been standardized, such as SOAP which is the most popular web service protocol standard. Businesses that use standards can export internal services as web services.

Likewise, a very limited recent trend in the health care industry is to share electronic medical records via web services. As the desire for collaboration grows among hospitals, so has the need to *integrate* web services among hospitals. Integrating web services together into higher level services allows the practitioner to make one simple request instead of tens or even hundreds. However, medical records sharing is unlike B2B interaction in that health care providers are concerned with tracking the medical records of individuals. It can be difficult to integrate such fine-grained services into a single seamless and coherent system.

Furthermore, many health care providers have not yet exposed their internal services as web services. In such cases, integrating new web services from existing internal services may require substantial effort because health care providers must somehow convert their non-standard internal formats into web service message formats. Furthermore, when a new provider does become web service enabled, a systemic infrastructure change is needed to integrate that provider's web services. For providers that are already web service enabled, each

provider's web services have their own request and reply payload format, even if all the providers use a standard messaging protocol such as SOAP. In other words, higher-level communication patterns and payloads are not standardized. For example, to request a patient's medication history, you must know the patient's medical record number (MRN) which is a unique identifier for the patient at a specific health care provider. Each health care provider has their own formats for their MRN's. Likewise, even though the response is a SOAP message, the medication history records inside the message are formatted to that health care provider's specifications, which typically differs substantially from other providers.

Moving protocol targets: Although some organizations have contributed several competing "standards" for sharing health care records, there is no clear winner [1], [2], [3], [5], [8]. When a winner does emerge, existing services will have to change to support it. However, for now, it is difficult for existing web services to support these moving protocol targets.

This paper introduces a configurable framework called Braid that enables health care organizations to exchange patient information from their native patient record systems, integrates existing native services across heath care providers, and supports web service infrastructure changes and moving protocol targets. In fact, adding new services, modifying existing services, and changing protocols requires only simple configuration changes, not sophisticated restructuring, recoding, and regeneration of web services.

The rest of the paper is organized as follows. Section 2 presents a brief discussion on related work. The architecture of Braid is presented in Section 3. The prototype implementation is discussed in Section 4. Section 5 presents our conclusions and future work.

2. Background and Related Work

Work in web services and web service frameworks, workflow patterns and languages, web service integration efforts, and enterprise service buses have influenced the design and implementation of Braid. The following sections describes each of these areas and overviews pertinent work.

2.1 Web Services and Web Service Frameworks

A *web service* is a software infrastructure that supports message oriented computer-to-computer data exchange and service invocation. A typical web service protocol stack consist of five protocol layers and the application layer. The network layer is typically TCP/IP. At the transport layer, standards-compliant, application-level Internet transport protocols such as HTTP or SMTP exchanges messages among hosts. The messaging layer provides interoperability between computers using text based messaging that provides abstractions for data types and formats. Example messaging protocols are Representational State Transfer, or REST, and XML based protocols such as XMLRPC and Simple Object Access Protocol, or SOAP. At the service description layer, web services export service descriptions using a language such as the Web Service Description Language, or WSDL, so that clients can generate local interfaces to the remote services. Finally, the service discovery layer provides protocols, such as UDDI, for clients to search for and find web services that match a particular set of criteria.

Many frameworks exist for implementing web services. The Java language and its Web Service Framework include WSDL generators and compilers, libraries, and the web services enabled servers Tomcat and Glassfish. Similarly, the Microsoft .Net framework and other application development frameworks support building web services.

Much of the technology developed for web services has been directly applied to Braid. In other words, Braid is itself a web service framework with its own set of tools and libraries, and can interoperate with any framework that supports the XML messaging protocols. Therefore, Braid itself can be used to implement web services, although it is primarily an integration framework.

2.2 Workflow Patterns and and Languages

As collaboration among businesses and institutions has grown, so has the need to integrate web services. Once area of development is *workflow patterns*. A workflow is a description of a business processing model. The workflow describes the route that the data follows though the network, the transforms and business rules applied to the data, and services invoked on the data. Workflow patterns have emerged that classify various models of workflow control[14]. Although Braid is a web service integration framework and lacks full support for business processing, it can be used to implement many workflow patterns.

The need for methods that describe and implement workflow patterns has lead to the development of several implementation languages. One such language, BPEL [9] is a complex language in which higher level services are programmed as state transitions. BPEL is a complex language in which even a simple "Hello World" program has a complex structure and can take many lines of code

to implement[12]. Braid's simplicity is a product of its service description model. Braid does not model services as state transitions, but instead models web services as dependent activities. Braid, not the programmer, determines the flow of activities and which activities happen sequentially and which happen in parallel.

Other flow languages have been developed as alternatives to BPEL's complexity. Medjaed, Bouguettaya, and Elmagarmid [13] propose an Ontology based framework for the automatic composition of web services using composability rules. Unlike Braid, an ontology based framework requires an ontology that describes each web service. XL [10] is a language designed for writing web services that also supports web service integration. Web component technologies, such as the one proposed by Yang and Papazoglou [15], provide a component based system for composing web services. Unlike Braid, these technologies do not automatically build flow graphs from dependencies; the programmer must specify the service invocations step-by-step.

2.3 Web Service Integration Efforts

Medical records sharing has become an active topic because of its impact on patient health care. Various organizations and research efforts have released specifications as well as working prototypes. The Markle foundation [4] has formed the Connecting for Health initiative (CFH) [1] for the purpose of improving networked information sharing among health care providers. As part of its initiative, CFH has release a specification, called the Connecting for Health Common Framework based on web services. The Framework specifies HL7-based messaging formats for medical data, standards for medical records location services, and other technical specifications as well as policy guidelines. Prototype services have been built based on the CFH Common Framework. Other medical records sharing efforts include MA-SHARE [3] and the Indiana Network for Patient Care [2]. Braid was created to support service integration in the domain of medical records sharing as part of the Mid-south E-Health Alliance [5].

2.4 Enterprise Service Bus

Many Enterprise Service Buses have been developed such as Oracle SOA suite [6], IBM Websphere [11], Apache Synapse [7], and others. These complex systems target the business industry, and thus incorporate many sophisticated technologies for complex business process management, such as BPEL. Braid's focus on service integration makes Braid a simpler yet very powerful framework.

3. Proposed Architecture

Figure 1 shows the high level architecture of the distributed patient information system built using the Braid framework. Each outer-most rounded box represents a regional health information organization, or RHIO, of collaborating, Internet connected health care providers. Each RHIO in the proposed architecture consists four parts: each health care provider's Proprietary Patient Record System (PPRS), an XML Translator for each PPRS, a Braid Server and a web service front end. A brief high level description of the proposed architecture follows.

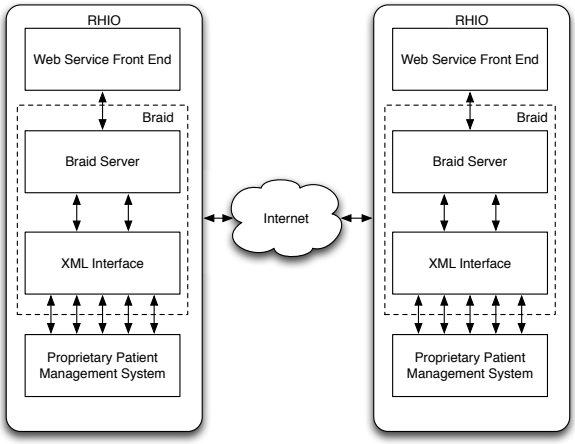

Fig. 1
HIGH LEVEL ARCHITECTURE.

Each health care provider usually has some type of patient record system that in our architecture is called the Proprietary Patient Record System (PPRS). The PPRS usually provides several information services such as patient demographics (name, date-of-birth, address, etc.), lab history, medication history and other clinical data. The interface, request format, and format of the patient information provided by the PPRS is native to a particular provider. Any client that requires patient information from one or more providers will make a request in a standards compliant, e.g. compliant with the CFH Common Framework, web service request through the exposed web service interface. When the Braid Server receives the request from the web service front end, it determines what PPRS services and in what order those services need to be invoked to fulfill the client request. The Braid Server composes a workflow of PPRS calls in an XML message format and forwards the XML messages to the corresponding XML Interface. An XML Interface translates a PPRS message in XML format to a PPRS native format, e.g. a SQL query using a database

specific driver, and sends the query directly to the PPRS. It also translates the native PPRS response into XML and sends it to the Braid Server. The Braid Server then converts the PPRS response into a standards compliant format, such as the CFH common framework format, and sends it to the client.

Figure 2 shows the detailed architecture of the Braid Server and XML Interface. The Braid Server has the following components: the configuration file, a Workflow Manager (WM), a Dependency Processor, and a Message Handler. The configuration file contains descriptions of predefined dependencies. The dependencies define the order in which each PPRS service must be invoked to respond to the corresponding web service. Each dependency also contains a transform in XSLT that formats outputs from source PPRS services into inputs to target PPRS services.

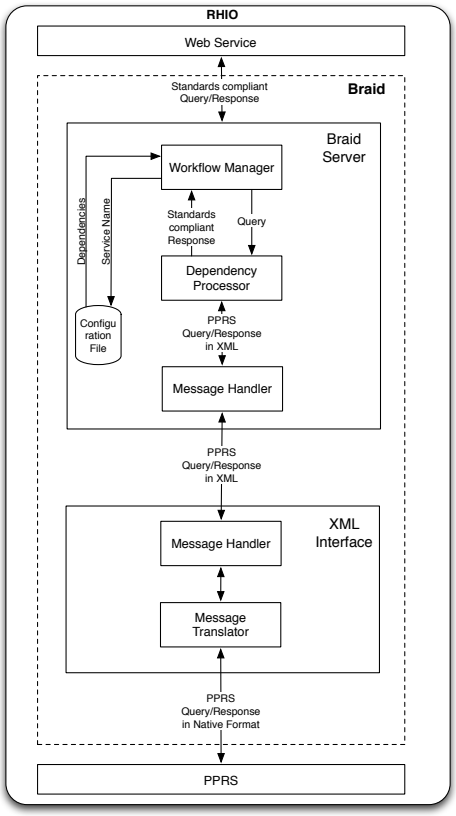

Fig. 2
DETAILED ARCHITECTURE

The Workflow Manager is the kernel of the Braid Server and coordinates the overall functioning of the server. The Workflow Manager maintains a map of web service calls to their corresponding dependencies. When the Workflow Manager receives a web service request it finds the corresponding dependencies from the map and loads them from the configuration file. It then passes the dependencies and web service request to the Dependency Processor. The Workflow Manager also keeps track of each client request by maintaining a unique session for the request.

The Dependency Processor invokes each PPRS service according to the dependency order starting with a special *entry* dependency. The Dependency Processor invokes a PPRS service by creating a PPRS request in XML format and sending the request to the XML Interface via a Message Handler. As the Dependency Processor receives replies from the XML Interfaces, it updates the state of its dependencies. Once all a target's sources reply, that dependency is considered to be fulfilled. Therefore the Dependency Processor applies the transform to the source outputs and invokes the target PPRS. If two targets do not depend on each other then they can be executed in parallel. The top-level target PPRS services depend upon the special entry target which passes them the original web service request. The input for subsequent PPRS services is extracted from the web service request and/or responses from previous PPRS services.

The Message Handler manages the communication between the Braid Server and the XML Interface. The Message Handler is designed to separate the Braid Server and XML Interface communication complexities from the Dependency Processor. The Message Handler implements standard communication protocols, e.g. HTTP, SMTP, AMQP, etc.

The XML Interface usually resides on a PPRS and consists of two components: a Message Handler and the Message Translator. The Message Handler sends and receives messages to and from the Message Handler of the Braid Server. Once it receives a message (XML PPRS service request) from the Braid Server, it passes the message to the Message Translator. The Message Translator converts the XML request into a native PPRS request and invokes the corresponding PPRS service. Once a response is received from the PPRS, the Message Translator converts the response into an XML format and passes it to the Message Handler. The Message Handler sends the resultant message to the Braid Server.

4. A Prototype Service

Braid is a part of the Mid-south E-Health Alliance [5] effort. Although Braid web services are not yet a part of daily use in a live environment, we have developed Braid service prototypes and tested them using actual patient data. The purpose of our prototype is to illustrate:

- How Braid can construct web services from PPRS data that is in a proprietary format.

- How Braid can integrate a provider's existing web services using Braids service composition capabilities.
- How Braid's higher level web services can be reconfigured.

An example high level Braid web service from the prototype, as depicted in Figure 3, is a patient medication history report. The input to this web service is a set of Medical Record Numbers, or MRNs. The web service returns a set of patient records describing the patient's medication history. In our prototype, PPRS A represents a member institution of the Mid-South E-Health Alliance, and we have used real but anonymized patient data from that institution, where the true identities of the patients have been replaced with a false, generated identities.

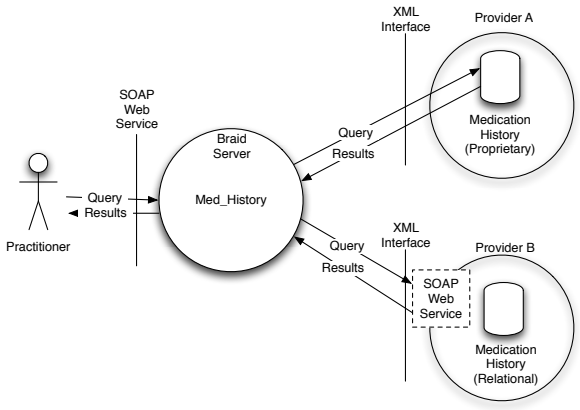

Fig. 3
EXPERIMENTAL SETUP.

The medication history web service is called Med_History and is a SOAP web service that conforms to the CFH Common Framework standard [1]. A practitioner makes the web service call via a web service client application. The web service interface passes the request to the Braid Server. The Braid Server retrieves dependencies from the configuration file for the Med_History service and passes the dependencies and the web service request to the Workflow Manager. The Workflow Manager then, according to the dependencies, applies transforms to the SOAP payload to generate the appropriate inputs, and makes the corresponding requests to the XML Interfaces of PPRS A and PPRS B. Once the XML Interfaces have responded, the Workflow Manager applies a final transform that formats the results as a CFH Common Framework medication history report, and returns the results to the practitioner.

To better understand the Braid framework and PPRS integration, consider supporting only PPRS A initially and adding PPRS B later. The proprietary format for the patient data at PPRS A is shown in Figure 4.

```
:ADM:01H
:ID:81506706301
:NA:Lastname1056, Firstname2045
:DOB:1977/05/07
:SEX:F
:DOC:Lastname1201, Firstname0894 MD
:TYP:HP
:STYP:Claim Rx
:DAT:2006/03/17
:TIM:00:00
:ESPV:rxview.cgi
:ACC:2751510
:HP:<RX>
   <ACTIVE_FLAG>A</ACTIVE_FLAG>
   <MEDICATION>
     <MED_DESCRIPTION>
      Promethazine HCl Tab 25 MG
     </MED_DESCRIPTION>
     <MED_STRENGTH>
      25.00000 MG
     </MED_STRENGTH>
     <MED_NAME>
       PROMETHAZINE HCL
     </MED_NAME>
     <DOSAGE_FORM_DESC>
       Tablet
     </DOSAGE_FORM_DESC>
     <PKG_DESC>BOTTLE</PKG_DESC>
     <PKG_SIZE>
       100.000 EA
     </PKG_SIZE>
     <PKG_QUANTITY>
      1
     </PKG_QUANTITY>
   </MEDICATION>
   <!-- truncated -->
 </RX>
```

Fig. 4
EXCERPT OF MEDICATION HISTORY DATA AT PROVIDER A.

The Message Translator converts these records into an XML response to the Workflow Manager's request. An Message Translator is configured at PPRS A with a simple Perl script that reads the patient data and converts it to XML using Perl's XML library. The script translates the data in a straightforward manner, as depicted in Figure 5.

In order for the Workflow Manager to implement the Med_History service, it requires a configuration file that describes a simple dependency: Med_History depends on PPRS A's medication history service. The Med_History configuration file is shown in Figure 6.

The configuration file must define two PPRS sources called ENTRY and EXIT. These are the entry and exit points of the Med_History service. A PPRS source for the remote medication history request at PPRS A is also

```
<medhist>
  <ADM>01H</ADM>
  <ID>81506706301</ID>
  <NA>Last1056, First2045</NA>
  <DOB>1977/05/07</DOB>
  <SEX>F</SEX>
  <DOC>
   Lastname1201, Firstname0894 MD
  </DOC>
  <TYP>HP</HP>
  <STYP>Claim Rx</STYP>
  <DAT>2006/03/17</DAT>
  <TIM>00:00</TIM>
  <ESPV>rxview.cgi</ESPV>
  <ACC>2751510</ACC>
  <!-- RX element copied here -->
</medhist>
```

Fig. 5
EXCERPT OF TRANSLATED MEDICATION HISTORY DATA.

defined and given an identifier of "pprs_a" so that the source can be referenced later. After the source definitions, a set of dependencies is defined. Each dependency names a target and its sources, which can also be other targets. In this case, the Med_Hist web service must be invoked by the practitioner before invoking the service at PPRS A. This constraint is enforced by the dependency that names pprs_a as a target and ENTRY as its source. Finishing the service requires that the service at PPRS A be completed, as constrained by EXIT's dependency on pprs_a. Each dependency has a transform associated with it that selects and re-formats the output XML from the sources on which it depends. This transformed XML is the input to that target. EXIT's transform formats the data so that it matches the CFH Common Framework format for a medication history report that is expected by the caller.

Consider adding a new provider which is depicted as PPRS B in Figure 3. The medication history information stored at PPRS B is in a relational database, but PPRS B has already implemented a SOAP web service. In the prototype, the web service at PPRS B does not support the CFH Common Framework's medication history report format. Supporting the added PPRS requires adding a XML translator for the SOAP service. In this case, the translator is just a pass-through because the PPRS data is already formatted as XML.

Additionally, modifications to the Med_History configuration file are required. The modifications are straightforward: The Med_History configuration file defines a new source and dependency for the SOAP service, and adds a source to the dependency with the EXIT target. Additionally, the XSLT transform, CFH_med_hist_results.xsl, must be modified so that it transforms the results from PPRS A and PPRS B into the CFH Common Framework format.

```
<braid_service>
   <name>Med_History</name>

   <source>
       <id>Entry</id>
   </source>

   <source>
       <id>pprs_a</id>
       <url>
        braid://pprsa.com:5555
       </url>
       <method>med_by_mrns</method>
   </source>

   <source>
       <id>EXIT<id>
   </source>

   <target>
       <id>ENTRY</id>
       <!-- depends on nothing -->
   </target>
   <target>
       <id>pprs_a</id>
       <depends>
           <id>ENTRY</id>
       </depends>
       <transform>
          A_med_hist.xsl
       </transform>
   </target>
   <target>
       <id>EXIT</id>
       <depends>
           <id>pprs_a</id>
       </depends>
       <transform>
           CFH_med_hist_results.xsl
       </transform>
   </target>
</braid_service>
```

Fig. 6
THE CONFIGURATION FILE FOR THE MED_HISTORY SERVICE.

5. Conclusions and Future Work

In this paper, we have presented a framework called Braid for integrating patient medical records systems. We have implemented web services using the framework and have found the implementations to be straightforward and the framework to be flexible. Typical modifications such as changing a Braid web service or adding patient providers to an existing web service require modest modifications. In particular, integrating a set of services requires writing scripts for the XML translator,

constructing a configuration file of dependencies, and implementing XSLT transforms for each dependency. Adding new services requires writing XML translator scripts for each additional service, adding the necessary dependencies to the dependency configuration file, and including the necessary XSLT transforms.

We have installed the prototype and added additional services. We have tested it under heavy loads with hundreds of thousands of service calls. For future work, we plan on presenting our experimental results that focus on performance, including response times, throughput, and overheads incurred during heavy loads. Additionally, we will test Braid's scalability by emulating a hierarchical, multi-RHIO system of networked Braid servers and provider services. We will also continue to examine and experiment with Braid's flexibility by implementing various web services such as those supported by MA-SHARE [3] and the Indiana Network for Patient Care [2].

References

[1] Connecting for health. http://www.connectingforhealth.org, Mar 2009.

[2] Indiana network for patient care. http://www.regenstrief.org/medinformatics/inpc, Feb 2009.

[3] Ma-share. http://www.mahealthdata.org/ma-share, Feb 2009.

[4] Markle. http://www.markle.org, Fed 2009.

[5] Midsouth e-health alliance. http://www.midsoutheha.org, Feb 2009.

[6] Oracle soa suite. http://www.oracle.com/technologies/soa/soa-suite.html, Feb 2009.

[7] Apache. http://synapse.apache.org, March 2009.

[8] Marco Eichelberg, Thomas Aden, Jörg Riesmeier, Asuman Dogac, and Gokce B. Laleci. A survey and analysis of electronic healthcare record standards. *ACM Comput. Surv*, 37:2005, 2005.

[9] Onyeka Ezenwoye and S. Masoud Sadjadi. Composing aggregate web services in BPEL. In *In Proceedings of The 44th ACM Southeast Conference*, 2006.

[10] Daniela Florescu and Donald Kossmann. Xl: An xml programming language for web service specification and composition. pages 65–76, 2002.

[11] IBM. http://www-01.ibm.com/software/websphere, March 2009.

[12] P. Louridas. Orchestrating web services with BPEL. *Software, IEEE*, 25(2):85–87, March-April 2008.

[13] Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. Composing web services on the semantic web. *The VLDB Journal*, 12:2003, 2003.

[14] B. Kiepuszewski W.M.P van der Aalst, A.H.M. ter Hofstede and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, July 2003.

[15] Jian Yang and Mike. P. Papazoglou. Service components for managing the life-cycle of service compositions. *Information Systems*, 29:2004, 2003.