Assessing Approaches for Evaluating Remote Heterogeneous Applications

L. Gebase[1] **and R. Snelick**[1]
[1]National Institute of Standards and Technology (NIST), Gaithersburg, MD, State, USA

Abstract - *We examine several methods for testing disparate, heterogeneous systems that intercommunicate through message exchanges governed by a set of well defined message exchange rules. Rather than basing our assessment on a set of abstract applications, we will look at a specific set of healthcare applications that exhibit these characteristics. We will then assert that the specific findings can be generalized and applied to a broader class of applications.*

The Integrating the Healthcare Enterprise (IHE) organization defines a technical framework for implementing a broad range of existing healthcare standards. These standards include HL7, DICOM, CDA and others. Implementations of a single standard share a common protocol for the exchange of messages, but direct message exchanges among different standards implementations are generally not possible. IHE, nevertheless, is interested in an environment made up of this diverse set of applications, and we will investigate various approaches to evaluate the behavior of implementations that make up this environment.

Keywords: Automated Testing; Conformance; Interoperability; Test Agents; Testing Framework.

1 Introduction

Our focus is primarily on conformance testing. One aspect of testing, separate but related to conformance testing, is message validity, and we will begin by addressing this issue. Following message validation, we will briefly address interoperability. In our examination of testing tools, we look at two tools developed at the National Institute of Standards and Technology (NIST) for evaluating certain aspects of conformance testing. Integrating the Healthcare Enterprise (IHE) [1] is an organization interested in examining multiple healthcare standards that address a variety of healthcare informatics needs ranging from the exchange of textual patient administration data to binary diagnostic imagining data. IHE holds an annual event in which numerous organizations from around the world participate. The challenge at this event is to establish interoperability among the numerous participating vendors whose applications implement one or more of the standards IHE includes in its Technical Framework Documents [2]. Beyond interoperability, IHE is also interested in assessing how effectively participating vendors adhere to the standards referenced in the IHE technical frameworks. This presents a challenging problem in conformance testing and to address this problem a testing architecture has been formulated that allows the broad range of implementations encompassed by IHE technical framework to be examined. We investigate some approaches to testing based on the use of this architecture. A common element present in each of the conformance testing methods we examine is the use of testing applications that support the functionality of one of the standards that may be tested. Finally, we introduce a novel approach to testing that makes use of this functionality present in each of the testing applications.

2 Message Validation

All standards referenced in the IHE technical framework define a protocol for message exchanges between applications. All messages are constrained by a grammar; in addition, Healthcare Level 7 (HL7) [3] messages may be further constrained by a message profile [10, 11]. One component of conformance then is adherence to the message grammar, or, in the case of HL7, a message profile. At NIST, we have developed a tool for evaluating HL7 messages. The tool has been implemented as a web application and any developer can upload messages for the tool to evaluate [4]. The message validation utility is also available as a web service. Message validation provides a good first step in assessing an implementation's adherence to the requirements of the standard.

3 Interoperability

The ultimate goal of conformance testing is achieving interoperability between applications. Constructing and sending valid messages increases the likelihood of two applications interoperating, but does not assure it. Additional conformance testing can further increase the likelihood of interoperating, but still does not assure it. Interoperability testing largely reduces to simply trying to interconnect two applications and exchange messages between them. The messages exchanged may be captured or logged and evaluated to uncover problems when they occur, but assessing conformance is primarily addressed as a separate issue and will be investigated further below.

4 Testing Tools

HL7 is one of the standards IHE is interested in testing. At NIST, we have developed two tools to directly assist in the testing of HL7 applications. One tool is test driven, i.e., it is designed to execute specific test cases. The second is user driven, i.e., the user is given control over the execution of each test step. Each of these tools is rested on a testing framework that we have developed at NIST for evaluating HL7 applications. Both tools, along with our test framework are openly available, including source code, from our tools website [5].

IHE refers to an implementation supporting a specific set of functional requirements as an *actor*. The NIST testing framework is made up of a set of HL7 actors, along with the infrastructure needed to support the actors. The test system actors are IHE actors that are primarily intended to be used for testing other IHE actors. For this reason and to distinguish the test system actors from other actors, we refer to the test system actors as *test agents* rather than actors. Presently the test system includes three test agents. IHE defines many different types of actors, and the system is designed so that new test agents modeling other actors can be readily incorporated into the test system.

Both testing tools are made up of a user friendly Web Client Graphical User Interface (GUI) supported by the testing framework. The test case tool provides the user with a complete description of test cases and is designed to execute test cases based on the user's application. The user selects the actor or SUT (Systems Under-test) that is to be tested, and the test tool automatically selects the test agents that are needed to test the SUT. The tool automatically engages in the correct actions once the SUT is selected and configuration is completed. The applications supported presently are limited, but it is a very effective evaluation tool within its scope of applicability. One drawback of the tool, though, is its lack of flexibility. If an error occurs, generally the testing must stop. Testing cannot be resumed from the point where it was left off at, but instead must start at the beginning each time a test case is run.

The second tool offers more flexibility, but more intervention on the part of the user is required to conduct a test. With this tool the user selects each actor and test agent that will participate in the testing. Each step of the test case is controlled by the user, each message to be sent must be constructed by the user, and the user specifies each message destination. The user must also configure the tool and provide all addressing and ancillary information. The tool does, however, save configuration data and also provides utilities to assist with message construction, thereby, reducing the effort required to conduct a test.

Both of these tools, while facilitating only a very limited part of the total testing conducted within the IHE environment, could be used as models for further development. If similar tools where developed for other protocol standards—and if the scope of the current tools was expanded—a substantial part of the IHE testing effort could be automated. Nevertheless, the approach is ad hoc and lacks overall coordination; in an environment as large as IHE, such an approach risks become unwieldy.

5 The IHE Connectathon

IHE hosts an annual event [6] for the purpose of testing interoperability among groups of IHE applications, i.e., among applications that implement one or more functional components or IHE actors. Vendors bring their implementations to this event and attempt to interact with other vendor implementations. IHE then attempts to evaluate how successful vendors are in achieving this goal.

For a number of years the evaluation approach has been to employ a number of monitors who manually evaluate the execution of each step making up the test cases employed for evaluating applications. Each step in the test case is evaluated and the monitor coordinates the execution of each test step with users who know how to direct their applications to take specific actions.

Actors can broadly be divided into two categories, responders or servers and initiators or clients. Thus to evaluate a test case a monitor must first make sure that responding actors are started and ready to receive messages. Once this step is completed, the monitor directs the first initiating actor to send a message to the appropriate responder. If the actors were successful in establishing a connection, there will be a message received by the responder that must be examined and typically a response will then be returned, which also has to be evaluated. The entire test case is typically made up of a sequence of such steps. For a large number of actors subjected to a large number of test cases, each made up of multiple transactions; this is a very labor intensive process.

The monitoring approach is significantly improved by employing tools such as those described in the previous section, but a more comprehensive approach to testing IHE applications is needed to improve testing.

6 The Gazelle Testing Architecture

There have been essentially two approaches to automating IHE testing. One has been the development of ad hoc tools to facilitate isolated parts of testing. The second has been work on the development of a single, integrated testing architecture to automate the entire testing process. The latter is a significant challenge in that such a test system will require testing a wide array of applications running at disparate locations without a standardized method for intercommunications among all applications.

The architecture that is being explored and developed to address this problem is called The Gazelle Testing Architecture [7].

The Gazelle project is an effort to formulate a comprehensive solution to the evaluation of all IHE applications through the development of a single, fully integrated testing framework. The project represents a significant challenge since it must find a way to incorporate multiple, independently operating applications with no commonly defined means of communicating into a single, cohesive environment.

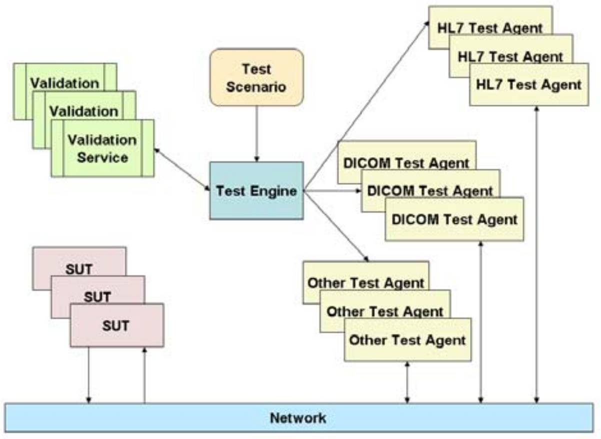

Figure 1: The Gazelle Architecture

The first step taken in the project's development was to recognize the utility of developing multiple, independent modules that perform the same functionality as the applications they will be used to test. While this kind of modular development would be essential for the project to succeed, there would also have to be some means of providing overall coordination among the modules if a single, unified system were to be developed. The solution presented to this problem was to incorporate a test engine at the core of the testing framework that would be responsible for the overall orchestration of all events carried out in conducting testing. The proposed Gazelle model is shown Figure 1.

The primary components of the architecture are the test engine, one or more systems or SUTs that are to be evaluated, and a number of test agents that interact with the SUTs and support testing. The test agents serve primarily as facilitators and as substitutes for actor implementations that interact with the SUT. An SUT may require a specific actor to carry out a message transaction, but the required actor may not be available. By building a test agent to interact with the SUT, the transaction can be conducted and evaluated without the required actor.

The *Validation Service* is an optional part of the architecture that can be used for assessing message validity subsequent to testing. The *Test Scenario* serves as input to the Test Engine and is mapped into a set of actions that the Test Engine carries out by directing the actions taken by the test agents.

Actors and test agents communicate over a network, but only those actor and test agents implementing the same protocol standard are able to communicate in this way. The test engine must have a means of communicating with all test agents regardless of the protocol they support. To satisfy this requirement, a web interface was defined that all test agents must support. The interface defines a set of functions common to all test agents that enables the test engine to direct a transaction with a remote actor or SUT being tested. For example, testing a responding SUT will typically include sending the SUT a message and evaluating its response. For this purpose, the web interface defines a *send* method that allows the test engine to pass a message to the test agent that the test agent, in turn, is to send to an SUT.

There is no web service that has been defined for supporting interactions between the test engine and the SUTs. Particularly for an initiating SUT, some means of communication is necessary. The Business Processing Execution Language (BPEL) [8] has been proposed for this purpose. In effect, this provides a mechanism that allows the test engine to notify the SUT user that an action must be performed; e.g., a message is to be sent to a responding actor or test agent. The web service interface includes a method for starting a test agent either as an initiator or a responder, so, in this case, the test engine uses the method to start a responding test agent.

7 Evaluating Message Transactions

There are several possible approaches to evaluating SUT behavior in the Gazelle environment. IHE requirements do not place any restrictions on how an SUT implements the services it provides, only that it exhibit the correct external behavior as manifested in the message exchanges it participates in. One option then for evaluating behavior is to use the test agents to assess the messages they receive from the SUT they interact with. There is a second, less obvious, method for evaluating behavior using the Gazelle architecture. An intermediary, know as a *proxy*, can be added to the architecture and all message exchanges can be required to go through the proxy. Following this approach, a log of all message transactions is saved, and a post testing analysis of the log is then performed.

Performing a post test analysis of a log of all message transactions presents obvious challenges. The first being how is the proper context in which to evaluate the message transactions reconstructed. The complexity of the problem can be substantially reduced by introducing multiple communication channels through the proxy. Figure 2 shows the architecture with a proxy added.

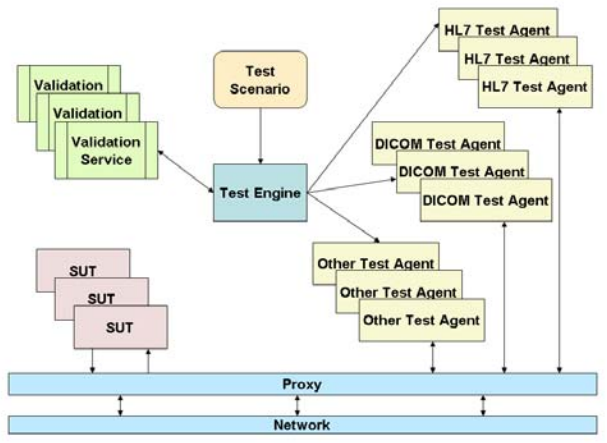

Figure 2: Gazelle Architecture with Proxy

The proxy is added to the architecture without imposing any additional communication requirements on either the SUTs or the test agents. The only changes are in the applications addressing configuration. For example, for *SUT-1* to send a message to *test agent*-1 without a proxy being present, it would send the message to the port the test agent is listening on at the test agent's IP address. With the proxy present, the port and IP address are simply changed to the proxy's IP address at the input port the proxy has assigned for *test agent-1*. Similarly, for the test agent to send to the SUT, it addresses the message to the input port the proxy has assigned for *SUT-1*. Now, rather than having a log file with a list of indistinguishable message transactions, the transactions can be separated based on the entities that were involved in the transaction. Since the test engine is in control of initiating all interactions, it is also possible to largely control the sequencing of the messages. Simultaneous transactions are avoided, allowing messages transmitted to be paired with the message response.

The test agents in this architecture are assumed to behave correctly; (over time, in fact, testing feedback will result in correct behavior). Thus when the test engine directs a test agent to send a message, the correct message is sent, and therefore it is possible to evaluate the response returned by the SUT against predetermined criteria. The approach isn't without difficulties, however, since it will still require the construction of predetermined conditions that must be applied to the correct messages. While constructing preconditions will allow correct evaluation under ideal conditions, unexpected events may result in incorrect evaluation. This can be seen by looking at an example of using this approach.

An approach that is typically applied when a proxy is used is to make use of the validation service to evaluate the preconditions that messages sent by an SUT must satisfy. The validation service can be used to ensure messages satisfy the general criteria; this service can be augmented to allow for the construction of predetermined evaluation criteria. This approach generally works well, but it will not work in all cases. Specifically, the receiving application may be allowed to reject messages under a variety of conditions that cannot be determined in advance. This can result in an SUT returning a valid message rejection, but the rejection failing to satisfy the predefined criteria. When this happens, an SUT may fail a test case while exhibiting valid behavior. This is one example of the more general problem that can arise when an application may respond with multiple valid responses.

If the test agents are used instead to evaluate SUT behavior, some problems encountered with the proxy approach can be overcome. For example, a test agent must be able to correctly handle all possible responses from a remote IHE actor. Thus they can incorporate all the logic necessary to evaluate SUT responses. Another advantage of this approach is that the context for message evaluation does not need to be recreated after testing is completed. But the problem of evaluating the behavior of an initiating SUT must also be addressed. The technique of using predetermined conditions can be employed more effectively in this case. The SUT has only one option; it must send exactly the message dictated by the test case being executed. The message could be evaluated by the external evaluation service, or it could be evaluated by a test agent. The advantage of using the test agent is that it will directly receive the message that is to be evaluated from the SUT, rather than requiring the message to be captured and then sent to the external evaluation service.

Both of these approaches offer significant improvements in automating test evaluation, but still may require substantial effort in constructing evaluation criteria and applying the criteria to the correct messages, and, in the case of the proxy approach, applying the criteria in the right context.

One additional approach which addresses some of these problems will be examined in the following section.

8 Evaluating Behavior

Ideally, conformance testing in general and IHE testing specifically, would be fully automated, not requiring manual intervention or the creation of predefined message criteria. A somewhat novel approach that exploits the capabilities built into the test agent implementations will be examined with this end in mind. Only the essential idea behind the approach will be described; it will be more fully elaborated in a future paper. For the case of HL7, an initial implementation of the approach has been included as part of the NIST testing framework.

The approach is relatively straightforward, but, nevertheless, may not be apparent even after examining the testing architecture. The proxy approach described above captures all message exchanges and thus makes a comprehensive evaluation of the SUT's behavior possible. But, in addition to the drawbacks identified above, the approach does not scale well. A more modularized approach is needed. One way of achieving this modularization is to employ multiple proxies rather than using the single, monolithic proxy described above. It may not be initially apparent, but the test agents that are already part of the architecture lend themselves well to this task.

Test agents already engage in message exchanges with the applications being tested. The test agents, therefore, include nearly all the functionality that is necessary to act as a proxy. As it turns out, there is little more than a switch that is necessary to add the capability. When the switch is turned on, the test agent, rather than processing the messages it receives, saves a copy, and sends the message to its intended destination. This requires deploying test agents at the right location, specifically along side of the actors that they are used to test so that all messages to the actor first pass through the test agent.

The essential architectural component for this model of testing is shown in Figure 3.

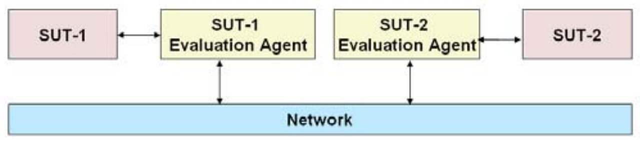

Figure 3: Evaluation Agent Model

The figure highlights the interactions between two SUTs that are being tested. For each SUT, there is a test agent added that is acting as a proxy; all messages exchanged with the SUT go through the test agent.

It may be apparent that calling the entity deployed along side of the SUTs a test agent is no longer appropriate. A test agent is just an actor deployed for a particular purpose, but the entities in this diagram are no longer behaving as actors; they are now simply serving as proxies. Not only are these entities behaving as proxies, as we will see below, they are also behaving as entities for evaluating the behavior of the SUTs. For these reasons rather than call the entities test agents, they are called *evaluation agents.*

It's clear that this approach to including a proxy for capturing message exchanges provides much better modularization, but it can also be exploited to provide better automation to the entire testing process as well. The key to exploiting this capability is recognizing that an inherent characteristic of a test agent is that it implements correct actor behavior. A test agent thus includes all the knowledge needed to assess the behavior of the actor or SUT that is being tested. The task then is to make use of this knowledge. By placing test agents in a position to capture all message exchanges as shown in Figure 3, it is a relatively straightforward task to augment a test agent with this capability and turn it into what is more appropriately called an evaluation agent. When this is done, evaluation can be fully automated, not requiring the construction of any predefined criteria, but simply exploiting the inherit characteristics present in all test agents.

Although our discussion has focused on testing one particular group of healthcare applications, this group of applications comprises a set of disparate, heterogeneous applications and the methodologies employed in testing them should be applicable to any set of applications having similar characteristics.

9 Conclusion

We have presented a number of approaches for evaluating the behavior of a specific set of applications. While our focus has been on IHE applications, the approaches examined should be equally applicable when applied to similar applications. The approaches we have examined have ranged from a very labor intensive, manual approach to evaluation to a fully automated approach requiring no manual intervention. The latter approach has thus far only undergone limited implementation and testing; we plan to more fully investigate the approach in our future work.

10 References

[1] Integrating the Healthcare Enterprise (IHE); http://www.ihe.net.

[2] IHE IT Infrastructure (ITI) Technical Framework Integration Profile, December 12, 2008.

[3] Health Level 7 (HL7) Standard Version 2.5, ANSI/HL7 V2.5-2003, June 26, 2003, http://www.hl7.org.

[4] NIST HL7 Message Validation Web Services. http://xreg2.nist.gov:8080/HL7Web/index.html.

[5] NIST HL7 V2 Testing Tools; http://hl7v2tools.nist.gov

[6] IHE Connectathon; http://www.ihe.net/Connectathon/

[7] Gazelle Testing Framework Project. Gazelle is a work-in-progress system targeted for the IHE connectathon

testing. The project is managed by Steve Moore (Washington University of St. Louis-MIR) and Eric Poiseau (INRIA). MIR, INRIA, NIST and others are developing the system in a joint effort.

[8] BPEL http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[9] DICOM, http://medical.nema.org/

[10] *Towards Interoperable Healthcare Information Systems: The HL7 Conformance Profile Approach*. R. Snelick, P. Rontey, L. Gebase, L. Carnahan. Enterprise Interoperability II: New Challenges and Approaches. Springer-Verlag, *London Limited 2007 pp*. 659-670.

[11] "Conformance Testing and Interoperability: A Case Study in Healthcare Data Exchange" Len Gebase, Robert Snelick, Mark Skall. 2008 Software Engineering Research and Practice (SERP08) conference proceedings, WORLDCOMP'08 July, 2008, Las Vegas, Nevada.