Demonstrating Semantic Interoperability of Diagnostic Reasoners via AI-ESTATE

John W. Sheppard
Department of Computer Science
Montana State University
Bozeman, MT 59717
406-994-4835
john.sheppard@cs.montana.edu

Stephyn G. W. Butcher, Patrick J. Donnelly
Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218
410-516-6115
steve.butcher@jhu.edu, donnell@cs.jhu.edu

Abstract—The Institute for Electrical and Electronics Engineers (IEEE), through its Standards Coordinating Committee 20 (SCC20), is developing interface standards focusing on Automatic Test System-related elements in cooperation with a Department of Defense (DoD) initiative to define, demonstrate, and recommend such standards.[1][2] One of these standards-IEEE Std 1232-2002 Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE)-has been chosen for demonstration. Previously, we presented the results of the first phase of the AI-ESTATE demonstration, focusing on semantic interoperability of diagnostic models. The results of that demonstration successfully showed the effectiveness of semantic modeling in information exchange. In addition, the engineering burden imposed by stronger semantic requirements was demonstrated to be manageable. In the second phase, the focus was on supporting reasoner interoperability by implementing semantically defined software services in a service-oriented architecture. Here, we present an overview of the semantic interoperability problem in the context of diagnostic reasoning and discuss the results of the second phase of the demonstration.

TABLE OF CONTENTS

1. INTRODUCTION

The Department of Defense (DoD) has established a partnership between government, industry and academia to address architectural design and standardization issues for automatic test systems (ATS). This DoD ATS Framework Working Group is focusing on defining an information framework and identifying standards for next-generation ATS. The principal requirement to be satisfied by the framework and associated standards is providing an open architecture for ATS to reduce overall cost of development and ownership for resulting families of standards. Based on work in the 1990s when the ATS Research and Development Integrated Product Team defined a set of "critical interfaces" for ATS, the current working group has been selecting, supporting the development of, and demonstrating commercial standards to be used in ATS with the intent of, ultimately, recommending these standards in future ATS procurement programs.

Of specific interest in the work reported here are the DIAS (diagnostic services) interface. Specifically, DIAS is defined as follows:

> Diagnostic Services are those standardized interfaces that facilitate transmission, conversion, and retrieval of diagnostic data for utilization in the maintenance process. These services link results obtained from the execution of a test with a diagnostic process that utilizes these results and suggests conclusions or additional actions that are required.

In 1976, the IEEE established the Standards Coordinating Committee 20 (SCC20) for purposes of standardizing on the Abbreviated Test Language for All Systems (ATLAS). Since then, SCC20 has expanded its scope to develop standards for larger system-level test and diagnostic related systems. In 1989, the IEEE approved a project authorization request (PAR) for SCC20 to develop a new standard focusing on diagnostic systems that use techniques from the maturing field of artificial intelligence—the Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE) standard under project P1232. In 1995, SCC20 approved and published the AI-ESTATE standard, IEEE Std 1232-1995 [1], and in 2002, the standard was updated [2].

The AI-ESTATE standard is undergoing a major revision to update several diagnostic models and to define data and software interfaces consistent with modern software architectures [3]. Prior to recommending any standards, several demonstrations are being performed to show that the standards were capable of meeting relevant DoD requirements. In September 2009, the second demonstration of the AI-ESTATE was completed, and this paper presents

[1] 978-1-4244-3888-4/10/$25.00 ©2010 IEEE
[2] IEEEAC paper#1071, Version 4, Updated 2009:12:31

the results of this demonstration. The results of the first demonstration can be found in [4]

2. THE AI-ESTATE STANDARD

The AI-ESTATE standard has been under development since the late 1980s and has undergone several significant modifications and advances over the intervening 20 years. AI-ESTATE is based upon the definition of formal semantic information models developed in the EXPRESS modeling language [5]. Initially, the standard was published in three parts:

- Architecture: The first part focused on a conceptual view of a diagnostic reasoner in the context of an abstract test environment [1]. This first standard imposed requirements on the following two standards rather than on any particular AI-ESTATE implementation.

- Knowledge Exchange: The second part introduced the first information models for diagnosis [6] and incorporated a little-used member of the Standard for the Exchange of Product model data (STEP) family for file exchange—EXPRESS-I [7].

- Software Services: The third part used the syntax of the EXPRESS information modeling language to specify software services, focusing mostly on model access services [8]. No specific implementation strategy was either specified or assumed; however, a binding strategy illustrated with the C programming language was provided as guidance.

Subsequent to their publication, all three standards were upgraded to full-use, and the revision process began. During the revision of the standard, the DMC determined that maintaining the standard would be simplified if the three "components" were combined into a single document. In addition, it was pointed out that the exchange format was not particularly useful since it was never intended for file exchange. Furthermore, the STEP community had already specified a standard specifically for file exchange, known as the "STEP physical file format" [9]. In 2002, a new version of the standard was approved making these changes [2].

The IEEE requires that all of their standards either be reaffirmed, revised, or withdrawn every five years. Shortly after publication of the 2002 version of AI-ESTATE, it was discovered that a significant error was introduced into the standard. Specifically, while the information models were being updated, the specification for failure rate had been deleted from the standard. At the time the error was discovered, the DMC believed a simple "corrigendum" could be prepared repairing the error.

As the process of preparing the corrigendum began, the DMC also decided to explore incorporating an XML-based file exchange into the standard. To accomplish this, the corrigendum changed into an amendment, and the DMC decided to develop a set of XML schemata for the models in the standard [10], [11]. As the amendment proceeded, the DMC determined that there were several places where the standard could be improved, and the XML schemata should follow the STEP specification for deriving XML from information models [12]. AI-ESTATE now started a full revision process.

In the spring of 2009, the newly revised AI-ESTATE standard was balloted [3]. That standard is now in the process of being revised based on over 700 comments received. The demonstration described in this paper has been developed in parallel and has provided several additional "rogue comments" for incorporation into the draft to be recirculated by the end of the year.

3. SEMANTIC INTEROPERABILITY

As mentioned in the previous section, the fundamental principle that underlies AI-ESTATE and the approach developed to write the standard is that of providing a sound semantic specification of the information to be exchanged with a diagnostic reasoner. Within software, generally two approaches exist to exchanging information: 1) through the exchange of static files, and 2) through a set of software services defined as an Application Program Interface (API). AI-ESTATE provides specifications both for file exchange and for software APIs. Although not stated explicitly, access to model elements can also be done through the STEP Standard Data Access Interface (SDAI) [13].

Several previous papers have been published focusing on the role of semantic models in addressing data exchange within AI-ESTATE [4], supporting information integration [14], and facilitating data mining [15]. In this paper, we focus on the role of semantic interoperability in defining software services and on the demonstration of the AI-ESTATE services. Within AI-ESTATE, five information models were developed to support model exchange. In addition, a sixth information model has been developed—the *dynamic context model* (DCM)—that defines the semantics of much of the information required when performing diagnosis. In addition, a new information model is under development as a result of the P1232 ballot that defines a set of types for the information passed by way of the services. We will discuss each of these two models here.

Dynamic Context Model

The AI-ESTATE standard describes the DCM as a model used that "captures a record of the reasoning session that may be used in any diagnostic context by an adequate abstraction of widely used diagnostic principles. … The DCM data and knowledge are developed during a

diagnostic session, unlike those of the [other AI-ESTATE models] (which consist of static diagnostic data and knowledge." As described, the DCM represents only a history of the diagnostic session; however, the semantics of the model do far more than that. First, the model defines the concept of a diagnostic *session*. A session is a container for everything that happens while a system is being tested or monitored and while reasoning about test/monitor information is going on. Thus, the session encapsulates the entire diagnostic process.

While the session is used as a "container" for the information captured during diagnosis, the key entity defined within the DCM corresponds to a *step* in the diagnostic process. The balloted version of AI-ESTATE defines a step as follows:

```
ENTITY Step;
  Name: OPTIONAL NameType;
  actionsPerformed : LIST OF ActiveAction;
  activeModels : SET [1:?] OF
       DiagnosticModel;
  optimizedByCost : SET OF CostCategory;
  optimizedByDistribution :
       ReliabilityDirected;
  optimizedByUser : HypothesisDirected;
  outcomesInferred : SET OF ActualOutcome;
  outcomesObserved : SET OF ActualOutcome;
  serviceLog : OPTIONAL LIST [1:?] OF
       ServiceState;
  stepContext : OPTIONAL ContextState;
  timeOccurred : OPTIONAL TimeStamp;
  userHypothesis : OPTIONAL SET [1:?] OF
       ActualOutcome;
  lifeCycleStatus : LIST [1:?] OF
       ActualUsage;
INVERSE
  owningSession : Session FOR trace;
WHERE
  modelsInSet :
       (SELF.activeModels <=
       SELF.owningSession.modelSet);
  hypothesisWithUserDirected :
       (NOT(SELF.optimizedByUser) OR
       EXISTS(SELF.userHypothesis));
END_ENTITY;
```

The step entity defines the main pieces of information to be collected during diagnosis and includes concepts such as the actions that are performed (including any setup actions, adjustments or repairs, and especially tests), the results of these actions (especially test results/outcomes), and any inferences drawn as a result.

The step also provides the ability to specify how the diagnostic reasoner should choose tests to perform, assuming the reasoner has that capability. Specifically, the standard assumes the reasoner as a default test choice capability. The specifics of that capability are immaterial; it could follow a static fault tree, perform a brute-force end-to-end test, or optimize test selection based on information gain. Up to three additional criteria can also be selected for optimization:

(1) Cost-based optimization: Incorporate cost factors as specified in an instance of the AI-ESTATE Common Element Model (CEM) such as the time required to perform a test or the skill level of the technician performing the test.

(2) Probability-based optimization: Incorporate information on the prior probability of a diagnosis incorporated into an instance of the CEM. Usually, these priors are determined based on failure rate information; however, the specific way the reasoner determines priors is not specified.

(3) Hypothesis-directed optimization: Incorporate knowledge of an expert, a technician, or any other "agent" that might be able to furnish a hypothesis to the reasoner. The reasoner then redirects its test choice process to focus on verifying or denying that hypothesis.

The ability to optimize by any or all of these approaches has been available to several diagnostic tools for at least 20 years, one of which was co-developed in 1988 by the lead author [16].

The AI-ESTATE standard does not specify how to use this model except to require that a reasoner be able to export a fully populated DCM using one of the two interchange formats. These exported files can then be used by applications to record the history of the session or support *post mortem* analysis of the session to support trending or maturation of the models. In addition, the DCM includes the ability to reference other DCMs to enable reasoners with the ability to incorporate historical information into the reasoning process to access that historical information. Because of the formal information model, these tools are able to use the exported information and understand exactly what the information means.

It is possible that a reasoner can also use the DCM to specify its internal representation of the state of diagnosis. While such a use is neither required nor necessarily even recommended, this use points out the richness of the model in that it is more than just historical data. In fact, it is this potential use that led to the word "dynamic" being incorporated into the name. As diagnosis proceeds, information integrity is maintained by mapping the information used to the DCM.

Service Interface Model

In addition to the DCM, the DMC is in the process of developing a new information model to specify semantics of the data being passed to and from the services. To explain this model, we also need to explain the service specification itself. In the balloted version of P1232, two classes of services were specified—model management services and reasoner manipulation services. The model management services were intended to permit a client to create, get, put,

and delete elements in any of the AI-ESTATE models. The reasoner manipulation services were intended to provide the primary means of a client communicating with a diagnostic reasoner during a diagnostic session. As a result of the P1232 ballot, the model manipulation services are being deleted, leaving only services for communicating with the reasoner.

We will describe a complete use case in the next section; however, here we describe three of the services specified. Diagnosis is a process of obtaining observations (e.g., test results) and drawing conclusions about the target of diagnosis based on those observations. Thus, three key services include getting a recommendation for a test to perform (`recommendActions`), reporting the results of that test to the reasoner (`applyActions`), and asking that the reasoner update its belief in the state of the target system (`updateState`). These services are specified using the syntax of EXPRESS as follows:

```
FUNCTION recommendActions(
  maxNumber : OPTIONAL INTEGER;
  levelsOfInterest : OPTIONAL SET [1:?] OF
       NameType;) :
  LIST [1:?] OF ActionRecomendation;
END_FUNCTION;

PROCEDURE applyActions
  (actions : LIST [1:?] OF ActualAction);
END_PROCEDURE;

PROCEDURE update_state;
END_PROCEDURE;
```

The `recommendActions` service has two optional attributes that specify the maximum number of actions to return as the levels of indenture of the target system for the analysis. It returns a set of `ActionRecommendation`, which is a new entity to be included in the new information model.

```
ENTITY ActionRecomendation;
  actionNames : LIST [1:?] OF NameType;
  actionDescriptions : LIST OF [1:?]
       DescriptionType;
  sequenceDescription : OPTIONAL
       DescriptionType;
  costCategories : LIST [0:?] OF NameType;
  catDescriptions : LIST [0:?] OF
       DescriptionType;
  estimates: LIST [0:?] CostValue;
  uppers : LIST [0:?] CostValue;
  lowers : LIST [0:?] CostValue;
  units : LIST [0:?] STRING;
END_ENTITY;
```

The `applyActions` service passes a list of `ActualAction`, which is an entity also defined in the new model.

```
ENTITY ActualAction;
  actionName : NameType;
  statusValue : OPTIONAL AssignedValue;
  statusConfidence : OPTIONAL
       ConfidenceValue;
  costLabels : OPTIONAL LIST [0:?] OF
       NameType;
  costValues : OPTIONAL LIST [0:?] OF
       CostValue;
END_ENTITY;
```

Thus, by including semantic definitions of the parameters and return values, AI-ESTATE provides an additional layer of semantic interoperability in the service specification as well.

4. DEMONSTRATING AI-ESTATE

The AI-ESTATE interoperability demonstration consists of three phases, where the first phase (now complete) focused on seamlessly exchanging diagnostic models between applications. The emerging revision of AI-ESTATE includes six schemata defining the semantics of the domain of system-level diagnosis using the ISO 10303-11 EXPRESS language [4]. Phase two uses standard services to interact with diagnostic reasoners, and phase three will integrate an AI-ESTATE conformant reasoner into an automatic test system. This paper focuses on the results of the first phase.

As we will describe below, the results of this demonstration successfully showed the effectiveness of using semantic modeling to define the standard services. All applications seamlessly interacted with the client application. In addition, the engineering burden was demonstrated to be manageable: all three applications were constructed in less than six months by two graduate students working part time.

The principal requirement of the second phase of this demonstration was to show that reasoner applications accessed remotely using a standardized application programming interface (API) defined relative to the information models. The result is that, as long as the reasoner publishes these standard services, then the underlying implementation should be irrelevant to the client application accessing that reasoner. To accomplish this, Web Service Definition Language (WSDL) specifications were developed and specified [17]. The usual way to use these specifications is by sending messages via the Simple Object Access Protocol (SOAP) [18]. Diagnostic reasoners and a separate client application were developed, including a fault-tree reasoner and a Bayesian reasoner. The demonstration process considered the impact on using previously defined AI-ESTATE conformant models and to provide test recommendations and hypotheses based on the results of performing the recommended tests. The development process was also monitored to determine the engineering burden to develop an application based on AI-ESTATE conformant models.

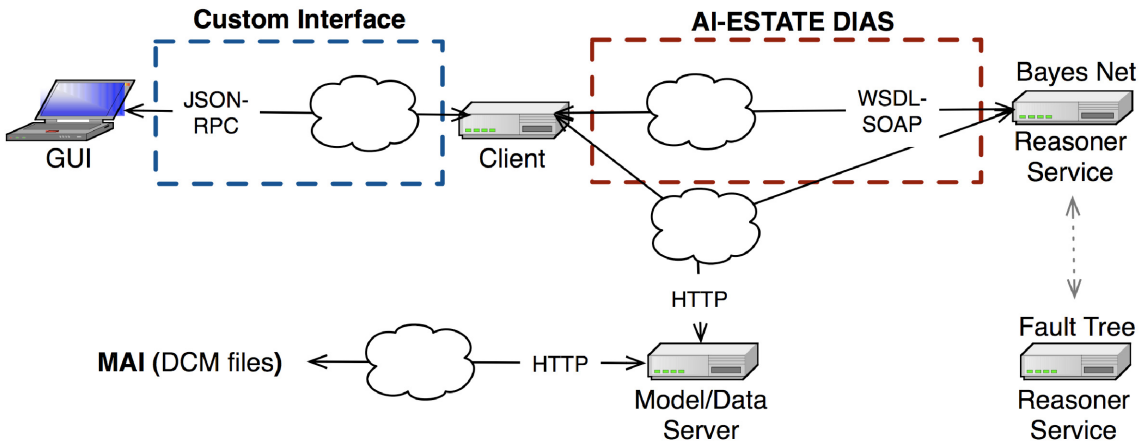


Figure 1. AI-ESTATE Demonstration Architecture

The architecture of the demonstration system made use of a software component architecture consisting of five major components—the user interface, a client gateway, two diagnostic reasoners, and a model server (Figure 1). From the user's perspective, a user interface is presented that is common for any AI-ESTATE reasoner. This user interface communicates via JSON/RPC to a "client" gateway that serves as intermediary with the actual reasoner. It is in this gateway that requests from the user interface are translated into SOAP service calls to the reasoner. For the demonstration, two different reasoners were implemented where WSDL-based services were published as specified by the standard. One reasoner was a fault tree reasoner to provide basic, legacy-type diagnostics. The second was a more advanced reasoner using a Bayesian network for purposes of diagnosis. The actual Bayesian reasoner used was the SMILE reasoner provided by the Decision Systems Laboratory at the University of Pittsburgh. Finally, a separate model server was developed to handle providing specific diagnostic models both to the reasoner and to the gateway. In addition, the results of diagnostic sessions were formatted according to the AI-ESTATE standard and exported for use by systems implementing the IEEE P1636.2 Maintenance Action Information (MAI) standard [19].

5. Demonstration Plan

The primary objective of the second phase of the demonstration can be summarized as follows:

> *To demonstrate the ability of IEEE Std 1232 (AI-ESTATE) to ensure the interchangeability of diagnostic reasoners through the definition of encapsulated services.*

To this end, the focus of this demonstration was on the implementation of the services specified in Clause 7 (Services) of the current draft of IEEE P1232. That clause specifies two classes of services—model management services and reasoner manipulation services.

Key to the demonstration is the abstraction created by specifying encapsulated reasoner manipulation services. This was demonstrated by using two different types of reasoners—a fault tree reasoner (to exemplify legacy diagnostic systems) and a Bayesian reasoner (to exemplify state-of-the-art diagnostic systems). Using these two types of reasoners also provides continuity between the two phases since these two reasoner types were also used in Phase I.

The following provides an outline of the demonstration process implemented in this phase.

(1) Defined use cases that would reasonably be expected to exemplify typical use of a diagnostic reasoner in a networked environment that also exercise all specified services with Clause 7.3 of IEEE P1232 (Reasoner Manipulation Services).

(2) Selected an existing fault tree reasoner and Bayesian reasoner from Phase I to serve as the baseline reasoners for Phase II.

(3) Developed a reusable WSDL/SOAP wrapper to serve as the interface layer for both diagnostic reasoners.

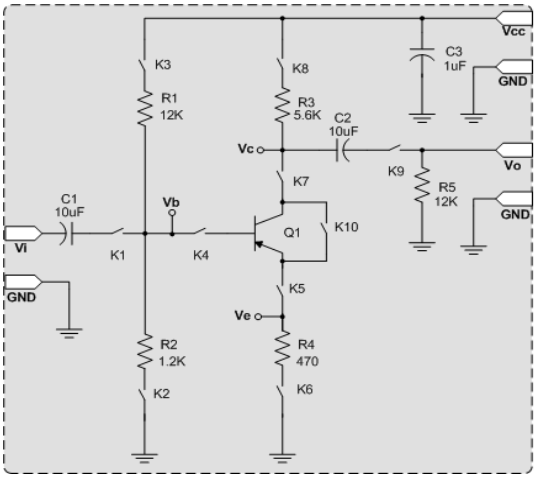


Figure 2. ATML UUT for Demonstration

(4) Modified the fault tree and Bayesian reasoners to use the WSDL/SOAP wrapper for communicating with an external diagnostic client.

(5) Developed a WSDL/SOAP wrapper for a generic diagnostic client to communicate with an arbitrary diagnostic server.

(6) Developed a diagnostic client user interface and integrate with the WSDL/SOAP wrapper.

(7) Mapped the use cases to test scenarios for the demonstration process.

(8) Conducted tests based on the test scenarios and evaluate the effectiveness of the implemented systems.

(9) Exported AI-ESTATE Dynamic Context Models in ISO 10303 Part 28 format for use with an associated IEEE Std 1636.2 (MAI) demonstration.

(10) Published the services and server address on the internet to permit evaluation from external parties (e.g., Impact Technologies, Lockheed Martin, or Boeing).

To illustrate the application of the standard reasoner, imagine the diagnostic client resides on a test station at a maintenance depot but two diagnostic reasoners with possibly different types of models are located on possibly different servers at remote locations. The test station has access to the internet or a closed intranet and is able to communicate with these reasoners over the network. A UUT is attached to the test station, and the maintenance technician requests that a model be loaded for that UUT. Once the model is loaded, testing begins.

At each step of testing, the tester would request a test recommendation from the diagnostic server. Upon receiving the recommendation, the tester would run the corresponding test (if possible) and provide the results back to the server. Based on the results of the test (which might include an inability to perform that test), the diagnostic hypothesis would be updated and returned. The process would continue until either no more testing was required or the technician determined the hypothesis was sufficient to take action. At the end of the session, the technician would request that the session be exported and associated with a maintenance action form (MAF) for the UUT.

6. A USE CASE—THE ATML UUT

In 2009 and 2010, additional demonstration programs were funding by the US Navy focusing on exchange of information in automatic test systems using IEEE standards for XML-based exchange. The Automatic Test Markup Language (ATML) family of standards (IEEE Std 1671 [20]). In the first phase of the demonstration, a simple low-frequency analog Unit Under Test (UUT) was built to support both parametric testing and diagnostics. A schematic of the designed UUT is shown in Figure 2.

As shown, the UUT consists of five resistors, three capacitors, and a transistor. For purposes of demonstration, a separate fault-insertion device was created to insert 13 different faults. Seven of the faults corresponded to a single failure mode for the resistors and capacitors as follows. Capacitors C1 and C2 could be failed open, and C3 could be shorted. Open faults were induced for resistors R1 through R4. For the transistor, six faults were able to be injected—an open on each of the base, emitter, and collector, a collector short, a base-emitter short, and a base-collector open.

Eleven tests were defined and implemented on the tester used for the demonstration designed to detect each of the faults above. Note that the tests were not sufficient to uniquely isolate each fault; however, the fault isolation capabilities of the test set were not the main point of the demonstration. The tests defined correspond to the following:

- An AC voltage test at V_O.
- A DC voltage test at V_O.
- An AC voltage test at V_C.
- A DC voltage test at V_C.
- A test for high DC voltage at V_B.
- A test for low DC voltage at V_B.
- A test for high DC voltage at V_E.
- A test for low DC voltage at V_E.
- A test for high DC voltage at V_C while bridging the base and emitter.
- A test for low DC voltage at V_C while bridging the base and emitter.
- A V_{CC} resistance (i.e., continuity) test.

The *D*-matrix corresponding to the associations between tests and faults is shown in Figure 3.

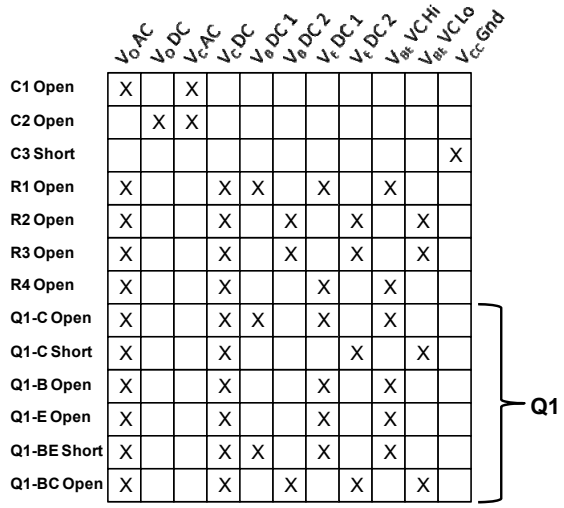


Figure 4. *D*-Matrix for ATML Test Circuit

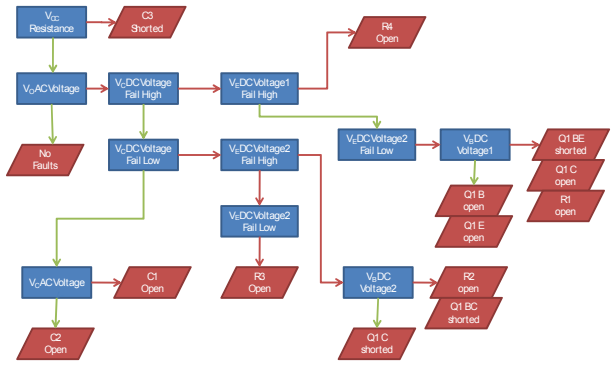


Figure 3. Fault Tree for ATML Test Circuit

As mentioned, both a fault-tree based reasoner and a Bayesian reasoner were created. The fault tree was generated by the ATML demonstration group and provided for purposes of encoding according to the AI-ESTATE standard and used by the AI-ESTATE-conformant fault tree reasoner. The fault tree is shown in Figure 4. As can be seen in the fault tree, only nine of the 11 tests were used. Specifically, the voltage tests where the base and emitter were bridge were omitted. The fault tree also reveals that three ambiguity groups of size greater than one are included.

Developing the Bayesian network was a bit more of a challenge. An initial network was provided by Lockheed-Martin, which was developed for the 2009 ATML demonstration. Upon close examination of the network, two issues were revealed. First, the transistor was not modeled according to the different failure modes. Only a single failure mode was assumed. Second, once the failure modes were inserted into the network, several probability tables needed to be defined, and these tables were likely to be very large because of the number of dependencies. To address this latter issue, several latent variables were added to the network to make the distributions more manageable. This had the added benefit of introducing several "hierarchical" relationships in the model corresponding to the ambiguity groups in the circuit. (Strictly speaking, these latent variables were not necessary; however, they greatly simplified the modeling problem.) The resulting network is shown in Figure 5.

7. RESULTS

The DoD is placing more and more emphasis on net-centricity in future ATS and maintenance system procurements [21]. Within the context of AI-ESTATE and diagnostics, net-centricity can be achieved through different approaches which can be broadly grouped into those that are data-centered and procedure-centered. Data-centered approaches generally fall under the category of RESTful interfaces although their adherence to the principles of representational state transfer (REST) can vary. REST centers on resources and Uniform Resource Locators (URL) to resources. In REST, the standard's `startDiagnosticProcess` would be mapped to `/myreasoner/session/create` where the parameters for the UUT and serial number of the `actualRepairItem` are supplied as XML or query parameters. The call would return a session identifier, which would be used in subsequent API calls. For example to call `recommendActions` using REST, one would access the recommended actions as a resource on the reasoner using a URL such as `/myreasoner/123456/actions/list`. The session id has become part of the URL which identifies the resource for this particular session. While the semantics of the standard could be achieved using REST, the standard has an explicit implementation bias towards procedure-centered approaches because the services must have the same names as those in the standard.

Procedure-centered approaches have a long history. The Common Object Request Broker Architecture (CORBA) is one of the oldest object-oriented remoting approaches. Remote Method Invocation (RMI) fulfills a similar role in Java-based solutions. For net-centric solutions, the oldest remote procedure call (RPC) type of approach is the standard HTTP GET/POST. This approach uses a URL and query parameters. Our `startDiagnosisProcess` could be implemented in HTTP GET/POST using a URL such as

```
/myreasoner/startDiagnosticProcess?uut=X&actu
                    alRepairItem=Y.
```

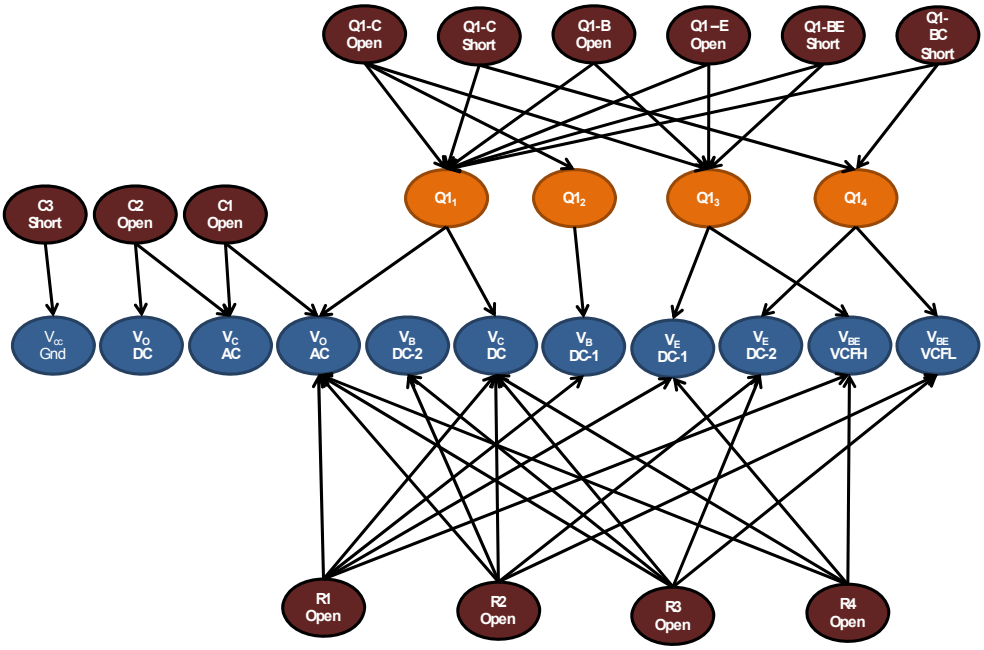


Figure 5. Bayesian Network for ATML Test Circuit

Unlike REST, HTTP GET/POST services can also be formally described in a WSDL. Another alternative is XML-RPC. In this case, an XML document is sent describing the method to be invoked and the parameters to that method on the remote server by POSTing an XML document to a service URL. XML-RPC eventually evolved into SOAP; however, XML-RPC is still used today for those who find defining a SOAP service through a WSDL to be "overkill." One final alternative is to use SOAP itself and one of its various encoding combinations including "rpc" or "document" with "literal" or "encoded." There is also a format recommendation called "wrapped document literal." All of these formats have their good and bad points.

Although the standard services definition precludes some implementations (REST), it lays the foundation for a simplified process of coordination and integration when parties must negotiate or use implementations. This process might be even further enhanced by the presence of reference implementations; however, SCC20 has historically taken a position against either providing reference implementations or conformance test suites.

Other than the various implementation issues and alternatives, the process by which the services have been implemented for demonstration served a very useful purpose. Several items were found to be missing in the balloted standard that would be of value (and even required), given common uses of diagnostic applications. Two areas where such deficiencies were found were with the management of resource availability and the specification of optimization criteria.

The balloted standard includes a service that permits a client to specify which resources (e.g., test instruments, power sources) are available: `setAvailableResources`. While the AI-ESTATE CEM includes information on resources, the DCM does not capture what resources have been identified as available. From the perspective of the service, the DCM does not need to store this information; however, given the DCM is supposed to capture a history of all information relevant to the diagnostic process, omitting this information was a serious oversight. As a result, the DCM has been modified as part of the demonstration to capture this information.

While the problem identified above illustrates a situation where the DCM does not support a service, the second problem illustrates the opposite situation—where no service exists to support information in the DCM. Specifically, the balloted version of the DCM defines four attributes of a step to support the optimization process:

- `Step.optimizedByCost`
- `Step.optimizedByUser`
- `Step.userHypothesis`
- `Step.optimizedByDistribution`

Unfortunately, no corresponding reasoner manipulation services were defined that enabled the client to set these attributes. Technically, the model management services could be used to set these values, except an approach was taken where model management services worked with entity ids and reasoner manipulation services worked with entity names. This mismatch made it impossible to use the model management services with the reasoner manipulation

services to set the optimization information. As a result, three services have been added as a part of this demonstration to address this deficiency:

- `optimizeByCostCriteria`
- `optimizeByDistribution`
- `optimizeByUserHypothesis`

The purpose of the demonstration process was to provide a way of "testing" a standard being considered by the DoD for future mandate. The intent is to provide a proof-of-concept that the standard works and will satisfy the DoD's requirements. But there is a more important benefit to the demonstration process, especially when no *de facto* standard exists. The demonstration process serves as a valuable tool for identifying deficiencies/errors and testing alternatives, thus providing a mechanism for producing an even more effective standard.

8. CONCLUSIONS

Two of the primary motivations behind the DoD adopting consensus standards for ATS development are to reduce cost by improving interoperability and to minimize repeated design of similar systems. The IEEE SCC20 standards focus on promoting information interoperability between components of a test or health monitoring system. The emphasis by the DOD on acquisition reform based on commercial standards for ATS, combined with declining budgets mandates the need for more affordable health management system development and operation.

A key objective of the demonstration reported here was to show that diagnostic reasoners developed according to the standard services specified in AI-ESTATE are capable of interacting seamlessly with a corresponding diagnostic client with minimal engineering requirements beyond those specified in the standard. An associated concern is whether the implementation of standards purporting to provide these characteristics can be done in a cost-effective manner. It is clear from this demonstration that AI-ESTATE satisfies all of these requirements when interacting with a client via the standard services. A pair of complete fault tree and Bayesian diagnostic systems was developed, including implementation of all of the reasoner manipulation services (e.g., test recommendation, test outcome application, and hypothesis recommendation). Furthermore, these systems were developed from scratch by two graduate students working part time for six months.

The question that remains involves the likelihood of tool builders to adopt the standard. We believe that this demonstration shows there are no major technical obstacles to adoption. We also believe that the two phases of the demonstration have demonstrated the value of incorporating semantic definition into the interfaces. Fortunately, early indicates from industry show that the idea is catching on. Members of SCC20 have indicated that organizations such as Lockheed Martin, Northrop Grumman, Honeywell, and Boeing are already developing tools conforming to the standard and are even adapting the models to other applications.

REFERENCES

[1] IEEE Std 1232-1995, *IEEE Trial-Use Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE): Overview and Architecture*, Piscataway, NJ: IEEE Standards Press.

[2] IEEE Std 1232-2002, *IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE)*, Piscataway, NJ: IEEE Standards Association Press.

[3] IEEE P1232, *IEEE Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE)*, Draft 4, Piscataway, NJ: IEEE Standards Association Press, 2009.

[4] John W. Sheppard, Stephyn G. W. Butcher, Patrick J. Donnelly, and Benjamin R. Mitchell, "Demonstrating Semantic Interoperability of Diagnostic Models with AI-ESTATE," *Proceedings of the IEEE Aerospace Conference*, Big Sky, Montana, March 7–13, 2009.

[5] ISO 10303-11:1994, *Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 11: Description Methods: The EXPRESS Language Reference Manual*, Geneva, Switzerland: The International Organization for Standardization.

[6] IEEE Std 1232-1997, *IEEE Trial Use Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE): Data and Knowledge Specification*, Piscataway, NJ: IEEE Standards Press.

[7] ISO/TR 10303-12:1997, *Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 12: Description Methods: The EXPRESS-I Language Reference Manual*, Geneva, Switzerland: The International Organization for Standardization.

[8] IEEE Std 1232-1998, *IEEE Trial Use Standard for Artificial Intelligence Exchange and Service Tie to All Test Environments (AI-ESTATE): Service Specification*, Picataway, NJ: IEEE Standards Press.

[9] ISO 10303-21:1994, *Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 21: Implementation* Method*: Clear Text Encoding of the Exchange Structure*, Geneva,

Switzerland: The International Organization for Standardization.

[10] *eXtensible Markup Language (XML) Schema Part 1: Structures*, Second Edition. W3C Recommendation, http://www.w3.org/TR/xmlschema-1/, 28 October 2004.

[11] *XML Schema Part 2: Datatypes*, Second Edition. W3C Recommendation, http://www.w3.org/TR/xmlschema-2/, 28 October 2004.

[12] ISO 10303-28:2007, *Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 28: Implementation Methods: XML Representation of EXPRESS Schemas and Data Using XML Schemas*, Geneva, Switzerland: The International Organization for Standardization.

[13] ISO 10303-22:1998, *Industrial Automation Systems and Integration—Product Data Representation and Exchange—Part 22: Implementation Methods: Standard Data Access Interface Specification*, Geneva, Switzerland: The International Organization for Standardization.

[14] Timothy J. Wilmering, "Semantic Requirements on Information Integration for Diagnostic Maturation," *IEEE AUTOTESTCON 2001 Conference Record*, Valley Forge, PA, September 2001.

[15] Timothy J. Wilmering and John W. Sheppard, "Ontologies for Data Mining and Knowledge Discovery to Support Diagnostic Maturation," *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, Nashville, TN, May 2007.

[16] William R. Simpson, John W. Sheppard, and C. Richard Unkle, "POINTER—An Intelligent Maintenance Assistant," *IEEE AUTOTESTCON '89 Conference Record*, Philadelphia, PA, September 1989.

[17] *Web Services Description Language (WSDL)*, v1.1, W3C Note, http://www.w3.org/TR/wsdl, March 15, 2001.

[18] *Simple Object Access Protocol (SOAP)*, v1.2, W3C Note, http://www.w3.org/TR/soap, April 27, 2007.

[19] IEEE P1636.2, *Draft IEEE Trial Use Standard Software Interface for Maintenance Information Collection and Analysis (SIMICA): Exchanging Maintenance Action Information via the Extensible Markup Language (XML)*, Piscataway, NJ: IEEE Standards Association Press, 2009.

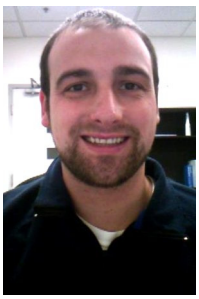
[20] IEEE Std 1671-2006, *Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML*, Piscataway, NJ: IEEE Standards Association Press.

[21] Michael Malesich, "New Direction for the DoD ATS Framework," *IEEE AUTOTESTCON '09 Conference Record*, Anaheim, CA, September 2009, pp. 64–68.

BIOGRAPHIES

John Sheppard *is the RightNow Technologies Distinguished Professor of Computer Science at Montana State University. He is also an Associate Research Professor at Johns Hopkins University. His research interests include algorithms for diagnostic and prognostic reasoning, machine learning and data mining in temporal systems, and reinforcement learning. Dr. Sheppard holds a BS in computer science from Southern Methodist University and an MS and PhD in computer science from Johns Hopkins University. He is a fellow of the IEEE and currently serves as Vice Chair of the IEEE Standards Coordinating Committee 20 (SCC20) on Test and Diagnosis for Electronic Systems.*

Steve Butcher *is currently pursuing his PhD in computer science at the Johns Hopkins University. He has served as a lecturer in economics and grader in computer science. He received his BA in economics from the California State University, Sacramento, his MA in economics from The American University, Washington, DC, and his MS in computer science from Johns Hopkins. His research interests are in machine learning and include Bayesian networks and evolutionary computation.*



Patrick Donnelly *is currently pursuing his Ph.D in computer science at Johns Hopkins University. He previously received a BS in computer science and an AB in music history and Italian literature from Washington University in St. Louis. He also holds an MSE in computer science from the Whiting School at Johns Hopkins, and both the MM in musicology and the MM in computer music from the Peabody Conservatory at Johns Hopkins. His research interests are primarily in machine learning in the musical domain.*