HADL: HUMS ARCHITECTURAL DESCRPTION LANGUAGE

R. Mukkamala V. Adavi N. Agarwal S. Gullapalli P. Kumar P. Sundaram

Department of Computer Science, Old Dominion University, Norfolk, Virginia

Abstract

Specification of architectures is an important prerequisite for evaluation of architectures. With the increase in the growth of health usage and monitoring systems (HUMS) in commercial and military domains, the need for the design and evaluation of HUMS architectures has also been on the increase. In this paper, we describe HADL, HUMS Architectural Description Language, that we have designed for this purpose. In particular, we describe the features of the language, illustrate them with examples, and show how we use it in designing domain-specific HUMS architectures. A companion paper [15] contains details on our design methodology of HUMS architectures.

Introduction

Architecture of a system specifies its components as well as the interactions among the components. In particular, it deals with structural issues such as the organization, component/function distribution, and communications. Often, architectural specifications include protocol specifications describing the system behavior and the details of component interactions.

Prior to implementing an architecture (software or hardware), it is important that we study its characteristics either through analysis and/or simulation. In particular, with the quick turn-around time expected in most of today's commercial applications, the architectural simulation is even more important [7].

The first step in the analysis or simulation of an architecture is architectural specification. In fact, when developing domain-specific architectures, it is desirable to store the available architectures (i.e., their specifications) in a design library. Thus, whenever an architecture needs to be developed for a new application, the available architectures could be analyzed for its suitability, and the new architecture arrived at [7]. Several architectural description languages (ADL) have been described in literature to meet the specification needs of applications in different environments.

In recent studies, monitoring the health of certain aerospace structures has been shown to be a key step in reducing the lifecycle costs for structural maintenance and inspection [13,14]. Since the health of the structures ultimately determines the health of a vehicle, health monitoring is also an important prerequisite for improved aviation safety. The need for developing architectures for integrated structural health monitoring has been discussed in [13].

For the last few years, we have been investigating some of the key characteristics of architectures for health and usage monitoring (HUMS) of aerospace structures. As part of this research, we have been looking into domain-specific architectures. For example, the HUMS architectures for battleships are different from those of HUMS for aircrafts. Similarly, HUMS for aircrafts are different from HUMS for helicopters. In order to maximize the benefits of architectural reuse, we have arrived at a design-expert system that uses a database of existing architectures to arrive at new architectures [15]. One of the major challenges in this effort is the efficient storing, retrieval, and evaluation of architectures in the database.

Keeping this in mind, we have defined a new architectural language called HADL or HUMS Architectural Definition Language. It is a customized version of xArch/xADL [4]. It is based on XML and, hence, is easily portable from domain to domain, application to application, and machine to machine. In addition, specifications written in HADL can be easily read and parsed using the currently available XML parsers. Thus, there is no need to develop a plethora of software to support HADL.

In this paper, we describe HADL and illustrate its use in specifying architectures for the HUMS domain. The paper is organized as follows. In section 2, we summarize existing work in the area of architectural description languages. In section 3, we provide a brief summary of HUMS architectures including reference architecture. Section 4 summarizes the features of HADL. Section 5 briefly summarizes the use of HADL based specifications in our design effort. Finally, section 5 describes future work.

Related Work

As discussed in the introduction, specification of an architecture is an important prerequisite for architectural design and evaluation. For this reason, several architectural description languages (ADLs) exist in current literature. Wright [2], Rapide [8], Aesop [12], xArch/xADL [4], and VHDL [5,11] are a few example ADLs. All these languages have their own syntax and their own compilers. Although all of these languages are concerned with architectural design, each provides certain distinctive capabilities. For example, Aesop supports the use of architectural styles [12]; ADAGE (Avionics Domain Application Generation Environment) supports the description of architectural frameworks for avionics navigation and guidance [1]; MetaH provides specific guidance for designers of real-time avionics control software [17]; C2 supports the description of user interface systems using a message-based style [10]; Rapide allows architectural designs to be simulated, and has tools for analyzing the results of those simulations [8]. Some of these ADLs are described below.

VHDL (Very High Speed Integrated Circuit (VHSIC) Hardware Description Language) is a high-level VLSI design language with which we could draw designs of digital hardware that will enable us to specify designs and simulate these designs to produce mostly complete systems [3,5]. These systems incorporate sufficient details and help testing hardware systems for correctness. One of the main uses of VHDL is to describe the structure of a system. It has been highly successful in its domain due to its reusability and robustness. Its main disadvantage is that it is too restrictive and cannot be extended to other domains for similar purpose.

Acme is another architectural description language developed at the Carnegie-Mellon University [6]. Its main goal was to provide a common language that could be used to support the interchange of architectural descriptions between varieties of architectural design tools [6]. One of the main advantages of Acme is that it is simple to use. It also provides the user with a representation for describing, storing and manipulating the designs produced. It also describes the architectures very efficiently and elegantly by having its own set of language constructs for describing architectural structure, types and styles. Its main disadvantage is that it is not applicable to all applications.

Darwin is a language for describing hierarchical architectures for systems [9]. Its strengths include the ability to model hierarchically constructed systems and systems that are distributed across many machines. Darwin also has limited support for dynamism through the definition of structures that can be dynamically instantiated. It has both a component based description and a graphical representation.

XArch/xADL is a standard, extensible XML-based representation for software architectures [4]. It is probably the first ADL that introduced the notion of using XML to specify architectures. According to [4], xArch specifically provides the following features: (i) xArch is a standard, extensible XML-based representation for software architectures; (ii) It provides a common core XML notation for architectures that can serve: as a simple stand-alone representation for architectures. Specifically, xArch provides a common set of bare-bone features that can be used to model an architecture. In addition, it can be used as a starting point for other, more advanced XML-based architectural notations. HADL draws its inspiration from xArch.

HUMS Architectures

Typical HUMS architectures consist of sensors at the lowest level. The sensors generate data (analog or digital) representing the health of the components that they are monitoring. The sensors are connected to transducers that convert the sensor data to digital form. The digitized data is now stored/processed by several high-level processors. A reference architecture for HUMS is shown in Figure 1 [13,14]. It is divided into three levels. The lower layer deals with the sensors; the middle layer handles the data storage and retrieval; and the upper layer deals with data processing (applications) and interaction with the users.

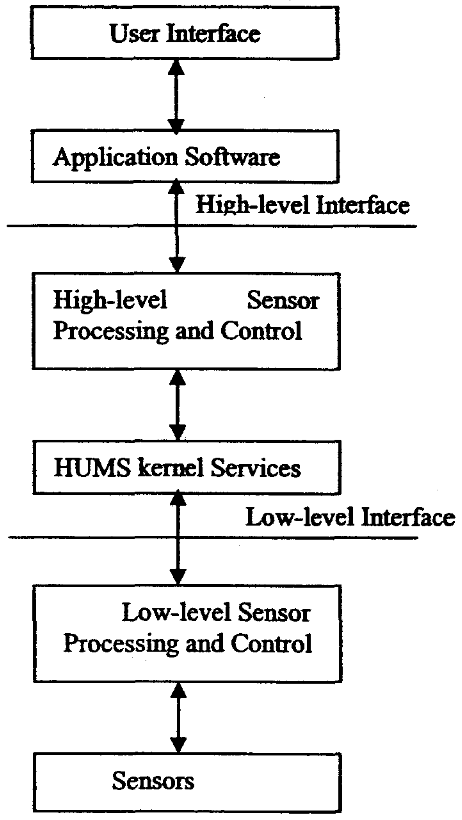

Figure 1. HUMS Reference Architecture

While figure 1 describes the required services in a hierarchical fashion, it does not provide sufficient details for us to build a HUMS system. For this purpose, we need an architectural description with details of the system components and the links that interconnect them. In other words, we need a more detailed description with implementation details for each layer along with inter- and intra-level connections of the components. Typically, the links are either point-to-point connections or broadcast media such as buses or rings. In this paper, we describe the use of HADL for the purpose of specifying such low-level architectural details.

Description of HADL

The core elements of HADL are component instances, connector instances, and interface instances. In addition, we define links and network types, which are of connector type, to simplify the specification of HUMS architectures. They are also the basic elements of xArch [4]. All these elements are necessary and sufficient to design HUMS architectures.

As in xArch, XML and XSD are used to describe an architecture in HADL. While the actual representation of the architecture is done in XML, XSD expresses the restrictions placed on the architecture. According to W3C consortium, "XSD (XML Schema Definition) specifies how to formally describe the elements in an Extensible Markup Language (XML) document. This description can be used to verify that each item of content in a document adheres to the description of the element in which the content is to be placed. [18]" In other words, it acts as a compiler for the XML representation. In the XSD schema-file, all the possible data types that are necessary are considered. Through this schema, the manner in which these data types are to be defined is stated.

Components

These are the building blocks for any architecture. Every part of the architecture can be represented using this data type. This element type represents the sensors, transducers, and other computational units. This can contain one or more elements of type componentInstance. ComponentInstance has a provision for every component to have an ID attribute by which the component can be identified. It also has a name element through which it is possible to assign a specific name to the component. Every component has an interface ID that facilitates connections to other components in the architecture. Components can be simple or complex. A simple component (type: *simple*) in HUMS can be one of the fundamental units like sensors, transducers, nodes, etc. A complex component (type: *arch*) can be a sub-architecture that is a combination of simple components with connectors, interfaces, and associated links. Consider the following example component.

```
<Componenents>
    <componentInstance id="1">
        <description>
            name="sensor1"
            Properties{requestRate:float=17.0;
                       sourcecode=Lib/Sensor.c }
         </description>
    </componentInstance>
</Components>
```

Here, a sensor component is being defined by its id (*1*), name (*sensor*1), and its properties. The properties include the maximum rate of requests that the sensor can respond to and the location of code that describes its functionality. The source code is often helpful in detailed simulation or analysis of an architecture.

Connectors

Connection instance provides users the capability to establish an interconnection between different components. This element type represents *coordination of activities* or *interactions* between components. This enables the user to define the type of network (protocol and topology) to be used like the Ring, Star, Bus or RPC (Remote Procedure Call). Through the network element, it is possible to define all the components that are being connected and the order in which they are being connected. For example, consider the example in figure 2. This describes a connector instance with id=1. It is configured as a "Star" configuration (A few basic configuration types are defined a priori). In addition, it specifies the components involved. In this example, three components were involved. Each component is also specified. For each connector, and interface is also specified. In summary, a connector instance consists of network type, components, and an interface definition.

Interfaces

In any architecture, a component is connected to one or more components. Interfaces specify the behavior of a component (or a connector) with respect to the external entities. It indicates the correlation between the component and the external environment. In Figure 2, the interface is simply specified as "star." In this case, it is assumed that the interface description of a star is known a priori and hence it is not explicitly specified.

Links

Links are used for interconnection between various components. Right now in the case of HUMS, three types of links are considered: RPC link, bus link, and cluster link. RPC link is a one-to-one link having one component connected to another component. The bus link makes it possible to have one-to-many connections. Cluster link represents a cluster of components such as nodes. Consider the example in Figure 2. The connector has three components connected using three links. The link specifies the type of components being connected. In this case, all three components are of type "arch" reflecting the fact that they are sub-architectures (and not simple components). The id of each component that a link connects is also specified (e.g., s1, s2, s3).

Network type

In HUMS architectures, it is common that the various monitoring systems will be connected in star, ring, or bus topologies. These are universal standards for network connections. Hence, we have introduced the network type to specify the topology of a connection. This provision helps us reduce the duplication due to repeated specification of these topologies repeatedly in an architectural specification. Instead, just the type of network needs to be defined. (In case a new topology is introduced, it could be defined once using other HADL constructs and repeatedly used later.)

Apart from the above, there are other types such as type of data medium, mobility (mobile or stationary), sensor type, etc. by which the components in an HUMS architecture may be specified.

Example Architecture

In this section, we first illustrate how different attributes of HUMS components may be specified using HADL. We then use an example system to further illustrate HADL features.

Consider the sensor attributes in Figure 3. Using a set of predefined tags (e.g., medium, protocol, mobility, weight, peak data rate, etc.), the attributes

```
</ConnectorInstance>
 <ConnectorInstance id="1">
   <Network>
      <Type>STAR</Type>
       <Components_Involved>
         <anchorInterface xlink:type="arch" xlink:href="#s1" />
         <anchorInterface xlink:type="arch" xlink:href="#s2" />
         <anchorInterface xlink:type="arch" xlink:href="#s3" />
      </Components_Involved>
    </Network>
     <InterfaceInstance id="2">
       <description>name="star"</description>
    </InterfaceInstance>
 </ConnectorInstance>
```

Figure 2. Illustration of a Connector, link and interface constructs

```
<HUMS:Sensor>
   <HUMS:Identity> </HUMS:Identity>
   <HUMS:Function> </HUMS:Function>
    <HUMS:Type></HUMS:Type>
   <HUMS:Medium> </HUMS:Medium>
   <HUMS:Protocol> </HUMS:Protocol>
   <HUMS:Mobility> </HUMS:Mobility>
    <HUMS:MeasurementRangeLower>-</HUMS:MeasurementRangeLower>
   <HUMS:MeasurementRangeUpper></HUMS:MeasurementRangeUpper>
    <HUMS:InstallationType> </HUMS:InstallationType>
   <HUMS:SignalType> </HUMS:SignalType>
   <HUMS:Accuracy></HUMS:Accuracy>
    <HUMS:Power></HUMS:Power>
    <HUMS:TransmittingPower></HUMS:TransmittingPower>
    <HUMS:Weight></HUMS:Weight>
    <HUMS:AvgDataRate></HUMS:AvgDataRate>
    <HUMS:PeakDataRate></HUMS:PeakDataRate>
    <HUMS:SelfMonitor> </HUMS:SelfMonitor>
 </HUMS:Sensor>
```

Figure 3. Sensor attributes in HUMS

```
<HUMS:Transducer>
    <HUMS:Identity> </HUMS:Identity>
   <HUMS:Function> </HUMS:Function>
   <HUMS:Type></HUMS:Type>
   <HUMS:Accuracy></HUMS:Accuracy>
   <HUMS:Power></HUMS:Power>
   <HUMS:Weight></HUMS:Weight>
   <HUMS:AvgDataRate></HUMS:AvgDataRate>
   <HUMS:PeakDataRate></HUMS:PeakDataRate>
    <HUMS:BufferSpace></HUMS:BufferSpace>
    <HUMS:TransmittingPower></HUMS:TransmittingPower>
   <HUMS:ReceivingPower></HUMS:ReceivingPower>
 </HUMS:Transducer>
```

Figure 4. Transducer attributes in HUMS

of available sensors can be completely specified and stored in the design library. Similarly, the attributes of transducers can be specified (as in Figure 4) and stored in the component database or design library. In addition, a user may specify design constraints as input to designing an architecture. For example, Figure 5 specifies the constraints on networks that may be used in an architecture. Such constraints may also be specified for other components or links. In this manner, the user input information, component specifications, as well as other subsystem and system architectures may be specified and stored in the design library.

Now, let us look at a simple HUMS architecture shown in Figure 6. It has three temperature sensors (S1-S3) connected through a ring and three fiber-optic sensors (S4-S6) connected through a bus. The ring and the bus are connected to another bus. Thus, the data from all the six sensors is put out on a bus. (For simplicity, we have not included any transducers). A processor, also connected to the bus, processes the data. A user's terminal accesses the processed data. It is also connected to the bus through which it accesses the processor data. The HADL specification for this example architecture is shown in Figure 7. The specification describes the components, the processor, the ring, and the two busses. It also specifies the order in which the components are connected to the networks. The specification for the components also includes references to the code that describes their behavior. This is useful in analysis or simulation of the architectures. Similar references could be included for any link or connector types.

Using HADL specification in Domain-specific Architectures

Our primary objective in developing HADL was to use it as a tool in designing domain-specific architectures. In particular, in our project, we are working on methods to develop domain-specific architectures for HUMS. Our approach is as follows.

- For the particular domain under consideration, several architectures may have been developed a priori. In fact, for large systems (such as ships and aircrafts) we assume that a variety of architectures have been developed for several sub-systems (e.g., engine subassembly, fan and cooling subsystems, etc.). These architectures are specified using HADL and stored in a design library.
- The user specifies the system requirements such as location of sensors, expected data rates, expected performance metrics (e.g., response time, availability, etc.), and constraints (e.g., weight limitations) using HADL.
- The user also may specify (if necessary) constraints on components that may be used in the architecture. For example, due to electro-magnetic interference, the user may only suggest the use of fiber-optic networks with certain data rates. Such limitations are also expressed in HADL and provided to the designer.
- The design explorer reads the user inputs (in HADL), parses it, and then using its logic searches the design library for subsystem and system specifications available. It then uses the evaluator module to evaluate the suitability of the retrieved architectures in meeting system requirements.
- Since most architectures may only be available in skeleton forms, the design explorer needs to derive detailed architectures from the chosen skeletal architectures. Details of this step are explained in [15].

Future work

The HADL development is currently in its early stages. So far we have used it to express only simple HUMS systems. As we experiment with more complex architectures, we may identify more attributes and features needed in HADL to make the architectural specification simple and complete. However, from our current experience, as well as the experience of other ADL developers, we are confident that the XML-based approach to specification is far more flexible than other approaches to specification. We are also developing graphic-interfaces through which user can provide input which is then converted to HADL form.

In summary, the work reported in this paper is work-in-progress. Each module of our

```
<constraints>
   <Type>BUS</Type>
   <Category> device net </category>
   <data rate> .05 </data rate>
   <bit time>2   </bit time >
   <max length>100</max length>
   <max no of nodes>64</max no of nodes>
   <max data size> 47/8</max data size>
   <min msg size>47 </min msg size>
 </constraints>
<constraints>
   <Type>BUS</Type>
   <Category> Ethernet</category>
   <data rate> 10   </data rate>
   <bit time>.01   </bit time >
   <max length>2500</max length>
   <max no of nodes>1000</max no of nodes>
   <max data size> 1500</max data size>
   <min msg size>72 </min msg size>
 </constraints>
<constraints>
   <Type>BUS</Type>
   <Category> control net</category>
   <data rate> 5   </data rate>
   <bit time>.02 </bit time >
   < max length>1000</max length>
   <max no of nodes>99</max no of nodes>
   <max data size> 504</max data size>
   <min msg size>7</min msg size>
 </constraints>
```

Figure 5. Examples of specifying network constraints in HUMS architectures

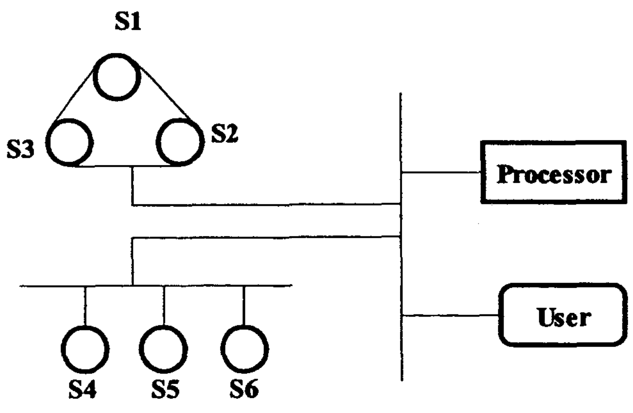

Figure 6. Example HUMS architecture

Figure 7. HADL specification for the Example HUMS architecture

```
<?xml version="1.0" encoding="UTF-8" ?>
 <HumsArch xmlns="http://www.odu.edu/~sgullapa/instance2.xsd"
    xmlns:instance="http://www.odu.edu/~sgullapa/instance2.xsd"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
    <ArchInstance id="0">
      <ArchType>Simple</ArchType>
      <description>SampleDesign</description>
      <Num_Components>8</Num_Components>
    <Components>
      <componentInstance id="1">
          <description>name="sensor1" Properties { requestRate : float =
             17.000; sourceCode : string = "CODE-
             LIB/tempsensor.c"}</description>
        </componentInstance>
      <componentInstance id="2">
          <description>name="sensor2" Properties { requestRate : float =
             17.000; sourceCode : string = "CODE-
             LIB/tempsensor.c"}</description>
        </componentInstance>
      - <componentInstance id="3">
          <description>name="sensor3" Properties { requestRate : float =
             17.000; sourceCode : string = "CODE-
             LIB/tempsensor.c"}</description>
        </componentInstance>
      - <componentInstance id="4">
          <description>name="sensor4" Properties { requestRate : float =
             500.000; sourceCode : string = "CODE-
             LIB/fibersensor.c"}</description>
        </componentInstance>
      - <componentInstance id="5">
          <description>name="sensor5" Properties { requestRate : float =
             500.000; sourceCode : string = "CODE-
             LIB/fibersensor.c"}</description>
        </componentInstance>
      - <componentInstance id="6">
          <description>name="sensor6" Properties { requestRate : float =
             500.000; sourceCode : string = "CODE-
             LIB/fibersensor.c"}</description>
        </componentInstance>
      - <componentInstance id="7">
          <description>name="collectingNode" Properties { collectRate :
             float = 1000.000; sourceCode : string = "CODE-
             LIB/CollectNode.c"}</description>
        </componentInstance>
      - <componentInstance id="8">
          <description>name="UserInput" Properties {InputRate : float =
             34.000; sourceCode : string = "CODE-
             LIB/UserInput.c"}</description>
      </componentInstance>
    </Components>
  - <Connections>
    - <ConnectorInstance id="9">
      - <Network>
          <Type>Ring</Type>
        - <Components_Involved>
            <anchorInterface xlink:type="simple" xlink:href="#1" />
            <anchorInterface xlink:type="simple" xlink:href="#2" />
            <anchorInterface xlink:type="simple" xlink:href="#3" />
          </Components_Involved>
        </Network>
      - <InterfaceInstance id="10">
          <description>name ="Out"</description>
        </InterfaceInstance>
      </ConnectorInstance>
    - <ConnectorInstance id="11">
      - <Network>
          <Type>Bus</Type>
        - <Components_Involved>
            <anchorInterface xlink:type="simple" xlink:href="#4" />
            <anchorInterface xlink:type="simple" xlink:href="#5" />
            <anchorInterface xlink:type="simple" xlink:href="#6" />
          </Components_Involved>
        </Network>
      - <InterfaceInstance id="12">
          <description>name ="Out"</description>
        </InterfaceInstance>
      </ConnectorInstance>
    - <ConnectorInstance id="13">
      - <Network>
          <Type>Bus</Type>
        - <Components_Involved>
            <anchorInterface xlink:type="simple" xlink:href="#7" />
            <anchorInterface xlink:type="simple" xlink:href="#8" />
            <anchorInterface xlink:type="simple" xlink:href="#10" />
            <anchorInterface xlink:type="simple" xlink:href="#12" />
          </Components_Involved>
        </Network>
      </ConnectorInstance>
    </Connections>
  </ArchInstance>
</HumsArch>
```

architectural design system is posing a number of interesting design and implementation issues. We should have the complete system in operation within the next one year.

References

[1] *A Rapide-1.0 Definition of the ADAGE Avionics System,* November 1994, 82-page technical report.

[2] Allen, R. J., A formal approach to software architecture, *Ph.D. Thesis, Carnegie Mellon University, CMU Technical Report CMU-CS-97-144, May 1997.*

[3] Christen, E., Bakalar, K., Dewey, A.M., and Moser, E., Analog and Mixed-signal modeling using the VHDL-AMS language, *36th Design Automation Conference*, New Orleans, June 1999.

[4] Dashofy, E.M., Hoek, A.v.d., and Taylor, R.N., A Highly-Extensible, XML-Based Architecture Description Language, *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA 2001),* Amsterdam, The Netherlands, August 28-31, 2001

[5] IEEE Standard VHDL Language Reference Manual, 2002.

[6] Garlan, D., Monroe, R. T., and Wile, D., Acme: An Architecture Description Interchange Language, *Proceedings of CASCON '97*, November 1997.

[7] Kathail, V., Aditya, S., Schreiber, R., Ramakrisha Rau, B., Cronquist, D., and Sivaraman, M., PICO (Program In, Chip Out): Automatically Designing Custom Computers, *IEEE Computer,* vol. 35, no. 9, pp. 39-47, September 2002.

[8] Luckham D. C. and Vera, J., An Event-Based Architecture Definition Language, *IEEE Transactions on Software Engineering*, Vol. 21, No. 9, pp.717-734. Sep. 1995.

[9] Magee, J., Dulay, N., Eisenbach, S., and Krammer, J., Specifying distributed software architectures, *5th ESEC'95*, Sitges, Spain, pp. 137-153, September 1995.

[10] Medvidovic, N. and Taylor, R.N., A Framework for Classifying and Comparing Architecture Description Languages, http://www.ics.uci.edu/ ~neno/dsl/dsl97.html#Med97.

[11] Medvidovic, N., et al., Using Object-Oriented Typing to Support Architectural Design in the C2 Style, *Proceedings of the ACM SIGSOFT '96 Fourth Symposium on the Foundations of Software Engineering*. pp. 24-32, ACM SIGSOFT. San Francisco, CA, October 1996

[12] Monroe, R. T. and Garlan, D., Style Based Reuse for Software Architectures, *Proceedings of the 1996 International Conference on Software Reuse*, April 1996.

[13] Mukkamala, R., Distributed scalable architectures for health monitoring of aerospace structures, *19th Digital Avionics System Conference (DASC'00),* Philadelphia, PA, October 7-13, 2000, pp. 6.C.4.1-6.C.4.8.

[14] Mukkamala, R., Bhoopalam, K., and Dammalapati, S., Design and analysis of a scalable kernel for health management of aerospace structures, *20th Digital Avionics System Conference (DASC'01),* Daytona Beach, FL, 15-19 Oct. 2001, pp. 3.D.2.1-3.D.2.10, 2001.

[15] Mukkamala, R., Adavi, V., Agarwal, N., Gullapalli, S., Kumar, P., and Sundaram, P., Designing Domain-specific HUMS Architectures: An Automated Approach, *22nd Digital Avionics System Conference (DASC'03),* Indianapolis, IN, Oct. 2003 (to appear).

[16] Robbins, J.E., et al., Integrating Architecture Description Languages with a Standard Design Method, *Proceedings of the 1998 International Conference on Software Engineering*. pp. 209-218, ACM Press. Kyoto, Japan, April 1998

[17] Vestal, S., MetaH: Avionics Architecture Description Language, Honeywell, MN, http://www.htc.honeywell.com/metah

[18] W3C XML Schema (XSD), www.w3.org/ XML/Schema.html