Modeling Wireless Sensor Network Architectures using AADL

Amitabha Ghosh[1], Luis Pereira[2], Ting Yan[2], Hui Cao[3]
[1]Dept. of Electrical Engineering, University of Southern California, Los Angeles, CA, amitabhg@usc.edu
[2]Innovation Center - Eaton Corporation, Milwaukee, WI, {LuisRPereira, TingYan}@eaton.com
[2]Dept. of Computer Science & Engineering, Ohio State University, Columbus, OH, caohu@cse.ohio-state.edu

***Abstract*— Recent technological advances have enhanced the possibilities of large-scale development and commercial deployment of diverse applications using wireless sensor networks. As this development effort expands, it becomes increasingly important to build tools and formal methodologies that ease large-scale deployments of such networks. In this paper, we advocate the use of an architecture description language called AADL to describe wireless sensor network architectures. We highlight the use of component-connector paradigm of AADL in designing robust, performance-critical, real-time sensor network applications incorporating relevant dependability metrics. By careful analysis and extraction of the common requirements, we describe a case study, that of a simple data collection application using sensor networks, as a proof of concept of the AADL modeling approach. Lastly, we propose several wireless sensor network specific extensions to AADL that will further enhance the richness of such models.**

***Index Terms*— Wireless sensor networks, AADL, architecture modeling, dependability.**

I. Introduction

Wireless sensor networks (WSN) have found diverse applications in several domains, such as, industrial plants and environmental conditions monitoring [18], diagnosis of health of civil infrastructures [28], military and security surveillance [29], healthcare [20], search and rescue operations, etc. These networks typically consist of hundreds to thousands of MEMS (Micro-electro mechanical system) enabled tiny wireless embedded devices that are severely resource constrained in terms of memory, energy, computation, and communication capabilities. In applications where reliability and dependability are of concern, these networks should be fault tolerant and robust in the face of occasional node failures and lossy wireless channels while guaranteeing application specific performance goals.

Traditionally, mainstream WSN research has focused toward specific problems, such as, designing energy efficient routing and medium access control protocols, finding optimal deployment strategies to guarantee coverage and connectivity, accurate node and network localization, micro-second level time synchronization protocols, etc. On the contrary, practitioners of WSN have stressed the importance of architectural analysis of systems [14] prior to implementation and analyzing trade-offs in design parameters in order to achieve desired performance and dependability goals. This *tuning* of system architecture is an increasingly complex process, which has been managed by architectural description languages (ADL), such as MetaH [16], AADL [6], [12], ACME [10], Rapide [19], Wright [1], etc. ADLs provide a formal way of representing the architecture in terms of software and hardware components, their interconnections, constraints, and configurations. They seek to increase the understandability and reusability of architectural designs. A necessary feature of an ADL is its support for multiple analysis approaches so that trade-offs can be realized across different domains. It should also support incremental analysis during the development process and multiple levels of fidelity for system evaluation. This incremental aspect allows the architecture specification to be used throughout the life-cycle.

Our contributions in this paper are the following:

1) We advocate the use of an architecture description language called *AADL* [7] (Architecture Analysis and Design Language) to model wireless sensor network architectures. As noted earlier, due to the applicability of sensor networks in diverse domains, we believe that such an architectural modeling approach will provide insights in developing reliable systems and help designers play with tunable parameters prior to deployment to best suit application specific needs.
2) We address the lack of tools and formal methodologies in designing networked sensor systems and illustrate how AADL can help in describing their architectures that are flexible to analysis prior to implementation. We also discuss appropriate error modeling using the AADL Error Model annex.
3) As a proof of concept of our proposed approach, we describe a case study, that of a sensor network data collection application, and analyze the following three dependability metrics: (a) end-to-end average packet success rate (PSR), (b) end-to-end average latency, and (c) system life time. As the implementation environment, we used OSATE [8] (Open Source AADL Toolset Environment) on the Eclipse platform to build the AADL model and performed Monte-Carlo simulations.
4) Lastly, we propose several extensions to AADL that are needed to model WSN systems more accurately.

The rest of the paper is organized as follows. In Section II, we discuss the advantages offered by architecture description languages in contrast with the limited flexibilities of existing

tools and simulators that have been traditionally used in modeling. In Section III, we give an overview of AADL by describing its basic building blocks, their interactions, and language features that are relevant in WSN design. In Section IV, we discuss the factors influencing the architecture of a WSN and define a mapping between the AADL and sensor network components. Section V describes a specific data collection application, and Section VI proposes several extensions to the basic AADL that would provide wireless sensor network specific features. Finally, we conclude in Section VII.

II. Benefits of Architecture Description Languages in Current WSN Design Practices

Industrial wireless sensors systems have gained recent popularity due to advances in low cost deployments, efficient energy and spectrum management, and local processing capabilities. There are commercial solutions currently available [2], [5] and standardization efforts [15], [30] to commoditize this technology by end users. Still, there exist application spaces in which these mainstream efforts yield suboptimal results in terms of resource efficiency. In applications like energy management [17], localized properties of WSN such as topology, information volumetric, energy sourcing, and work-flow processes are exploited to provide a tailored solution.

Unfortunately, most of the current design processes and tool sets, besides being affected by architectural informalities, have a strong bias toward addressing some of the WSN architectural concerns like scalability, reliability, and energy consumption while neglecting or oversimplifying others like commissioning, cost, upgradeability, deployment, and architectural refinement. Common tools that are traditionally used for modeling and simulation, such as MATLAB/Simulink [21], ns2 [22], GlomoSim [11], xUML implementations (Rhapsody [25] and Rational Rose [24]), and OPNET [23] provide partial answers to these concerns since they focus mostly on the functional, informational, and concurrent viewpoints of the architecture (service primitives, data/packets structures, reactive behavior, and interactions), while the deployment and operational aspects are simply out of scope. Table I shows our assessment in their expressiveness power to a particular architectural viewpoint following the Rozanski-Woods approach [26], and Table II shows our assessment on the their capabilities to support architectural perspectives relevant to WSN. Obviously, a determined and skilled person can improve these tools to cover more aspects, but our advocacy is to focus the efforts in domain specific problems rather that transform the tool to cover objectives better suited by an architecture description language. An ADL enables a domain-neutral approach that facilitates automated and interrelated analysis wherever possible, while architect-friendly viewpoints (e.g., functional, informational, concurrency, and deployment) can be generated from the description for understandability and stakeholders' discussions.

With the employment of an ADL to represent the complete architecture and the usage of mapping rules to transform it for each domain specific tool, it is possible to perform a more comprehensive architectural evaluation while leveraging the existing knowledge base embedded in such tools for detailed evaluation. Under this framework, designers can build multiple models of the same functional element (e.g., physical, MAC, network) of a system representing different levels of fidelities (e.g., closer to a specific hardware platform) depending on the kind of analysis to be performed while preserving the underlying architecture.

AADL is gaining support from the industry because of its standardization efforts and UML interoperability, while providing support for a subset of desired ADL features needed for WSN design. Since AADL has a textual form, the inclusion or removal of elements satisfying particular requirements can be managed and potentially traced to requirements managing tools, enabling analysis under different requirements and assumption sets. For functional and informational viewpoints, every component in AADL has a type specification representing its external interface. Multiple implementations, possibly differing in internal structure of their subcomponents and connections, can be defined for the same component type, thus, supporting multiple model variants of the same architecture. For deployment and concurrency viewpoints, AADL enables the ability to specify the runtime architecture of the embedded application as a set of communicating tasks that execute on one or more processors. Users can specify concurrent units of execution (threads), their rates, deadlines, worst case execution times, as well as the interactions between those tasks. Thus, AADL focuses on platform specific models and defines semantics that can be executed to assess target specific characteristics. Also, an AADL specification can be annotated with fault and error conditions on the component models when studying dependability analysis. For instance, to model the occasional failure of sensor nodes or the lossy wireless channel one can define an error state to represent the failure, an error event that caused the failure (such as, sensor hardware failure or presence of noise), and an error propagation that represents the effect of failure among interdependent components. These kinds of software and hardware reliability analysis are difficult to model with simulators as they are architecture dependent.

III. AADL: Architecture Analysis and Design Language

AADL is an architecture description language that is derived from the experiences based on DARPA funded ADL projects, MetaH [3] in particular, developed by Honeywell. AADL is a textual and graphical language to provide a standard and sufficiently precise (machine-processable) way of describing and analyzing the software and hardware architectures of real-time, performance-critical, embedded systems. It can describe the structure of such systems as a set of interconnected software components that are mapped onto a set of execution platform components. AADL describes common functional interfaces, such as events and data flow, as well as performance-critical features related to timing, resource allocation, fault-tolerance, etc. Using the notion of operational modes and mode transitions, AADL can also describe the dynamic behavior of a system. In the following, we give a brief overview of the essential features of AADL that are relevant in designing a

TABLE I

POWER OF TOOL LANGUAGE EXPRESSIVENESS TO SUPPORT ARCHITECTURAL VIEWPOINTS.

Tools	**Functional** (Describes a system's runtime functional elements and their responsibilities, interfaces, and primary interactions)	**Informational** (Describes the way in which the architecture stores, manipulates, manages, and distributes information)	**Viewpoints Deployment** (Describes the environment into which the system will be deployed, including the dependencies the system has on its runtime environment)	**Operational** (Describes how the system will be operated, administered, and supported when it is running in its production environment)	**Concurrency** (Describes the concurrency structure of the system, mapping functional elements to concurrency units to clearly identify the parts of the system that can execute concurrently, and shows how this is coordinated and controlled)
ns2	Limited	Limited	No	Very Limited	Yes
Executable UML	Yes	Yes	Very Limited	Limited	Yes
OPNET	Yes	Yes	Very Limited	Very Limited	Yes
Matlab	Very Limited	Limited	No	No	Very Limited
Simulink	Yes	Limited	Limited	Very Limited	Yes

TABLE II

CAPABILITIES OF TOOL SUPPORT ANALYSIS ALIGNED TO ARCHITECTURAL PERSPECTIVES.

Tools	**Evolution** (The ability of a system to be flexible in the face of the inevitable changes that all systems experience after deployment, balanced against the costs of providing such flexibility)	**Development** (The ability of a system to be designed, built, deployed, and operated within known constraints related to people, budget, time, and materials)	**Perspective, Availability, and Resilience** (The ability of a system to be fully or partly operational as and when required and to effectively handle failures that could affect system availability)	**Performance and Scalability** (The ability of a system to predictably execute within its mandated performance profile and to handle increased processing volumes)	**Security** (The ability of a system to reliably control, monitor, and audit who can perform what actions on these resources and the ability to detect and recover from failures in security mechanisms)
ns2	Very Limited	No	Yes	Yes	Very Limited
Executable UML	Very Limited	No	Yes	Limited	Very Limited
OPNET	Very Limited	No	Yes	No	Very Limited
Matlab	No	No	Limited	Limited	No
Simulink	Very Limited	No	Yes	Yes	Very Limited

WSN architecture. A detailed description of AADL and its usage can be found in [27].

The basic building blocks of AADL are called *components*, which are connected together to build a complete system. AADL provides several mechanisms to exchange control information between components, including message passing, event passing, synchronized access to shared components, and remote procedure calls. A component abstracts a specific element of the system by defining its interfaces and contents. The language syntax is composed of keywords that represent basic components, connections, and behaviors. Keywords are often annotated with additional keywords that describe features, properties, refinements, and extensions to the basic components. Features are used to exchange control and data via connections with other components, and properties are expressions that represent attributes with possibly associated values and behaviors of the component. AADL components are named and have types that represent their externally visible interfaces. They support the object oriented paradigm of inheritance [4], in the sense that they can extend other component types and refine the inherited features. The AADL standard defines three categories of components: (a) *Software*, (b) *Platform*, and (c) *Composite*. We briefly describe these components and their inter-relationships in the following (see Fig. 1 for their graphical representations).

A. Software Category

The software components of AADL are data, subprogram, thread, thread group, and process. They model source text, virtual address spaces, and units of concurrent execution.

1) Data: The data components model static data, which may be shared by multiple threads and processes by declaring appropriate access control tags. Concurrent access to shared data components is managed by a concurrency control protocol specified as a property.

2) Subprogram: The subprogram components model functions that are executed sequentially. They can be called from within threads and subprograms.

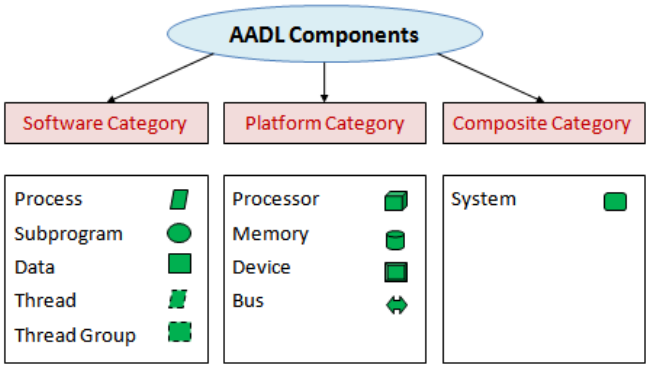


Fig. 1. AADL components and their graphical representations.

3) Thread and Thread Groups: The AADL thread components model units of concurrent execution that are managed by a scheduler. A thread can in turn contain subprogram and data components. The dynamic semantics of a thread are defined using hybrid automata. It includes different *states*, such as suspended, ready, and running; *transition events* between states, such as dispatch requests, faults, and runtime service calls; and *constraints* on time that a thread will spend in a given state under a given circumstance. A structural grouping of threads within a process comprises a thread group. A thread group may require and provide access to data components.

4) Process: A process models partitions in terms of protected address spaces. Access protection of these address spaces, to represent fault tolerance, is enforced at runtime using properties. The binary image produced by compiling and linking the source text must execute properly when loaded into this virtual address space. As processes do not represent units of concurrent execution, they must contain at least one thread. However, they can also contain thread groups, data components and can access or share data components among different threads.

B. Platform Category

The execution platform components of AADL are processor, memory, bus and device. They represent hardware or software that is capable of scheduling threads, enforcing virtual address space protection, storing source code and data, interfacing with an external environment and performing communication for system connections.

1) Processor: A processor component is an abstraction that can schedule and execute threads. Processors can contain memory and require access to buses and can support different scheduling protocols.

2) Memory: Memory components in AADL model randomly accessible storage devices such as RAM or ROM. They can contain nested memory components and require access to buses and can have properties such as the number and size of addressable storage locations.

3) Bus: AADL bus components model communication channels that can exchange control information and data between processors, memories, and devices. A bus is typically a hardware that supports specific communication protocols, possibly implemented through software. Logical connections between threads that are bound to different processors transmit their information across buses which provide the physical connection between processors. Buses can be directly connected to one another and might require access to other buses.

4) Device: AADL device components model physical devices such as sensors and actuators that interface with external environments. Devices are logically connected to application software components and physically connected to processors and may require access to buses.

C. Composite Category

1) System: AADL system components organize hierarchical compositions of software and execution platform components. It is generally used early in the modeling process as a generic component and can be made more specific when decisions are made about which components will be included in the system.

One of the ways AADL supports logical connections among these components is through the directional flow of data and control using ports. There are three types of ports: (a) *data ports* that represent connection points for transfer of state data (e.g., sensor measurements), (b) *event ports* that represent connection points for transfer of events, and (c) *event-data ports* that represent connection points that represent transfer of events along with data. AADL components can have different modes that represent alternative configurations of the component implementation, with only one mode being active at a time. The dynamic behavior of the system can be captured through mode transitions that changes the set of active components, their configurations and connections. An AADL specification can be used in a variety of ways during system development, e.g., for documentation during preliminary specification, for reliability and dependability analysis, and for building of efficient, robust, and consistent system architecture.

IV. Sensor Network Architecture Modeling

The idea of modeling a sensor network using AADL is to build a system architecture that will help in analyzing the cross-cutting aspects of design and the effects of embodiments on system performance, especially in resource constrained devices. AADL supports modeling a system at various levels of abstractions allowing quantification of system behavior by focusing on various performance characteristics (e.g., end-to-end latency, network lifetime). Since the requirements of a sensor network are application specific, trade-offs are often made among the design parameters in order to satisfy specific performance and dependability goals. For example, in a surveillance network where real-time data delivery is necessary, a typical trade-off is between network latency and probability of packet reception. In order to guarantee freshness and low latency, data should be transmitted at a higher rate, however, this increases the probability of packet drops. Therefore, a suitable tuning of transmission rate and acceptable network latency is required. In this paper, our main goal is to describe an AADL model of a sensor network that

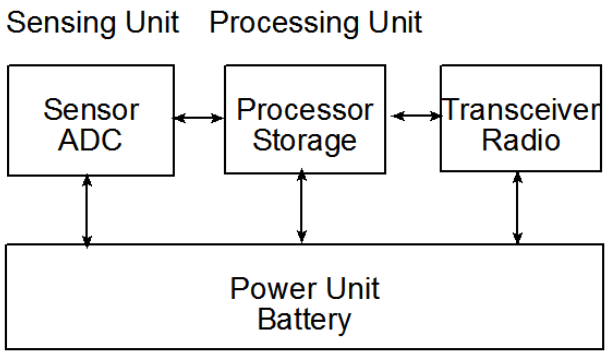


Fig. 2. Basic components of a sensor node.

would support dependability analysis. To this end, we focus on the following three dependability metrics.

1) Packet Success Rate (PSR): It measures the average end-to-end packet reception rate for all the nodes in the network over a period of time. The output of this metric could be a statistical distribution, or its mean and variance.
2) Latency: It measures the average end-to-end packet delivery delay for all the nodes in the network over a period of time. Like PSR, the output could be a statistical distribution, or its mean and variance.
3) System Life Time: It is defined as the maximal expected time that a certain fraction of nodes (e.g., 80% of all nodes) can still communicate with the base station. In practice, it can be measured by the percentage of disconnected nodes at a certain time.

The three dependability metrics as defined above have a correspondence with the general computer system dependability attributes: PSR basically maps to reliability, while latency maps to availability, and system life time maps to both availability and maintainability. We do not consider the performance metric, throughput, in this work. Note that, these three metrics might weigh differently for different applications.

To model a wireless sensor network, our first task is to identify the functionalities and interdependencies among the different sensor network components, and then describe them using appropriate AADL abstractions that map best with respect to their functionalities. At a very high level, a wireless sensor network is composed of three elements: (a) sensing nodes, (b) wireless medium, and (c) set of protocols and algorithms running on each node. At the highest level we model a sensor network as the AADL composite component, *system*.

A. Sensing Node

A sensing node has four basic components, as shown in Fig. 2: (1) a sensing unit consisting of a sensor and an ADC, (2) a processing unit consisting of a micro-processor and memory, (3) a radio transceiver unit, and (4) a power unit. AADL provides the flexibility to model these components at different levels of abstractions; however, a system designer would choose the appropriate level based on the analysis that needs to be performed for the specific application. Thus, a sensing node could be modeled at a high level using the composite component called *system* along with its associated properties, or it could also be modeled at a finer level using its subcomponent-specific models, such as representing the sensor hardware and the analog to digital converter using *devices*. In general, any kind of hardware component whose internals are hidden from the end user and which interfaces with the external environment could be modeled as a device along with its associated properties. In a similar way, the sensor measurements could be modeled using *data* and the interface between a sensor and the environment as a *bus* using *ports*. The processing unit, which consists of a CPU and a small memory, could be modeled using the platform component called *processor* with relevant CPU and memory properties. Lastly, the transceiver unit that enables node-to-node communication and the power unit, usually a battery, could also be modeled using *device* with relevant properties, such as its voltage and energy content. However, in our case we adhere to the high level modeling of the sensing node and the overall sensor network as a *system* and define properties corresponding to their subcomponents. This level of modeling is appropriate and sufficient for the dependability case study that we present later. The details of the AADL model of the sensing node is shown in Fig. 3. The variable names are self-explanatory and hence we do not explain them in details.

The model for the overall sensor network is a *system* that specifies several parameters, such as the data sets location, the sink node, and the number of simulations. In addition, to compare the performance for different configurations, we add "analysis" scenarios for different cases (or profiles), battery capacity, power consumption, channels, transmission power levels, and latency. Due to lack of space we do not show the AADL code here.

B. Modeling Errors

Mapping the WSN components to appropriate AADL components with respect to their functionalities and the level of analysis required is only part of the complete AADL specification. We also need to define failures and error conditions that the components might be subjected to and define a complete error model of the system that would lend itself to a variety of analysis methods. The *AADL Error Model Annex* [9] provides this flexibility. In this section, we describe the basic concept of how to define error models on components and overall error behavior of the system.

The AADL error model annex is basically a means to annotate the architecture specification by supporting failures and error behaviors of the components. The error behavior of a complete system emerges from the interactions between the individual component and connection error models. Therefore, the overall system error model is a composition of error models of its components, where the composition depends on the structure and properties of the architecture specification. Each component error model is a stochastic automaton and the rules for building the overall system error model by composing the individual error models depend on potential error propagations and error management behaviors.

```
property set NodeProp is
    node_id: aadlinteger applies to (system);

–Location Properties
    node_x: aadlreal applies to (system);
    node_y: aadlreal applies to (system);

–Hardware Components Properties
    node_memory: aadlreal applies to (system);
    node_cpu_speed: aadlreal applies to (system);
    node_type: aadlinteger applies to (system);

–Environments Properties
    op_temperature: aadlreal applies to (system);
    dormant_temperature: aadlreal applies to (system);
    humidity: aadlreal applies to (system);
    vibration_level: aadlreal applies to (system);

–Operating Profiles Properties
    case_ID: aadlinteger applies to (system);
    profile_ID: aadlinteger applies to (system);
    node_sensing_freq: aadlreal applies to (system);
    node_sensing_range: aadlreal applies to (system);

–Battery Properties
    node_battery_energy: aadlreal applies to (system);
    power_consumption: aadlreal applies to (system);
    duty_cycle: aadlreal applies to (system);
    isBattery: aadlinteger applies to (system);

–Node State
    node_state: aadlinteger applies to (system);
    node_time: aadlreal applies to (system);
    route_time: aadlreal applies to (system);
    node_update_interval: aadlreal applies to (system);
    route_update_interval: aadlreal applies to (system);

–Node Reliability Properties
    node_Live_Prob: aadlreal applies to (system);
    link_quality: aadlreal applies to (system);
    node_neighbors: aadlreal applies to (system);
    battery_ok: aadlinteger applies to (system);

–Radio Properties and others
    node_comm_range: aadlreal applies to (system);
    channel_no: aadlinteger applies to (system);
    tx_Power: aadlinteger applies to (system);
    processing_time: aadlreal applies to (system);
    transmission_time: aadlreal applies to (system);
    ack_time: aadlreal applies to (system);
    timeout_time: aadlreal applies to (system);
end NodeProp;
```

Fig. 3. The AADL model of a sensing node.

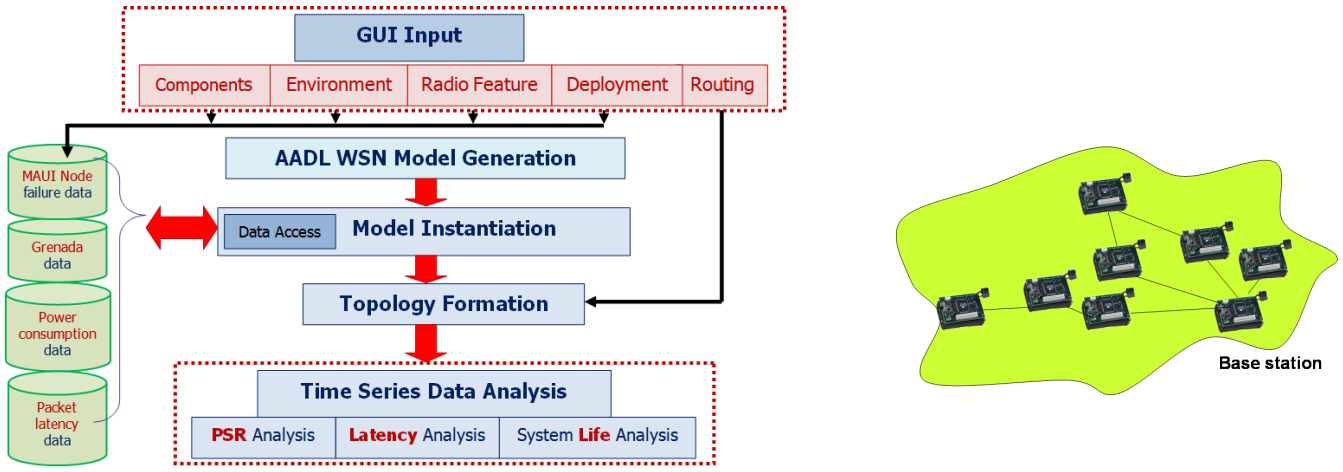
An AADL error model consists of the following declarations: (a) *Error States* that identify the current error state the component or connection is currently in; (b) *Error Events* that specify the transitions as to how the error states change; (c) *Error Propagation* that defines how state transitions happen; and (d) *Error State Transitions* that are conditions triggering transitions, and propagations as a result of transitions. Once an error model is defined, it can be associated with one or more appropriate AADL hardware, software, and composite components or connections. In Fig. 5 we illustrate a sample error model and associate it with the CPU (modeled as a processor) of a sensing node.

The first code segment defines an error model *type* called *dependent* with two error states: (a) the initial state *Error_Free*, and (b) the error state *Failed*. It assumes that the failure event *Fail* and the repair event *Repair* can occur according to a Poisson distributions with parameters λ and μ, respectively. The error model assumes that a failure event can influence the behavior of other components that depend on it. This is made visible through the error propagation clause *FailVisible* with a fixed probability p. In propagations are the consequences of out propagations from other components; hence they do not need Occurrence properties. We defined two primary transitions for this error model *implementation* applied to a CPU in the second code segment: (a) when the *Fail* event occurs the state changes from *Error_Free* to *Failed*, and (b) when the *Repair* event occurs the state changes from *Failed* to *Error_Free*. The two supplementary transitions defined on the error model implementation states that: (a) if a component is in *Error_Free* state and receives a *FailedVisible* in propagation then it transitions to the *Failed* state, and (b) the component remains in the *Failed* state when propagating *FailedVisible* out propagation. Once the error model is defined, we associate it with the CPU of a node within the constructs " annex Error_Model $\{$ ** " and " **$\}$ ", as shown in the last code snippet. The error state of the CPU will follow the transitions as defined in the error model in case a failure or a repair event has occurred.

V. Case Study: A Data Collection Application

Reliable and efficient delivery of sensor data is a fundamental requirement of almost all WSN applications. In this section, we describe a data collection application that serves as a proof of concept of the advantages and flexibility that AADL provides in modeling a WSN architecture. The software architecture of our system is shown in Fig. 4(a). There are essentially five components: (a) a GUI that takes inputs corresponding to each of the WSN subcomponents, such as the environment, radio characteristics, deployment, and routing; (b) an AADL WSN model generation component that generates the AADL model from the GUI input; (c) a model instantiation component that instantiates an AADL object from the model and the data set; (d) a topology formation component; and (e) a time series data analysis component that analyzes the WSN system with respect to the three dependability metrics described earlier.

We used four kinds of data in our analysis: (a) MAUI hardware reliability and node failure data (Eaton proprietary



(a) (b)

Fig. 4. (a) The software architecture for AADL modeling. (b) The WSN under analysis: 8 nodes send data to the base station using AODV routing.

```
error model dependent
features
    Error_Free : initial error state;
    Failed : error state;
    Fail : error event { Occurrence ⇒ poisson λ };
    Repair : error event { Occurrence ⇒ poisson μ };
    FailVisible : in out error propagation { Occurrence ⇒
fixed p };
end dependent;

error model implementation dependent.CPU
transitions
    Error_Free - [ Fail ] → Failed;
    Failed - [ Repair ] → Error_Free;
    Error_Free - [ in FailedVisible ] → Failed;
    Failed - [ out FailedVisible ] → Failed;
end dependent.general;

system implementation node.sink
    subcomponents
        CPU : processor ATmega16;
        . . .
    annex Error_Model { **
        Model ⇒ CPU_Error_Model ::
                dependent.CPU applies to CPU; **}
end node.sink;
```

Fig. 5. Sample CPU error model definition

radios), which is a function of temperature and the environment; (b) Grenada link quality evaluation data [13], which is a function of several parameters, such as time, channel number, packet length, transmission power level, and deployment of nodes; (c) power consumption data inferred from the microprocessor data sheet; and (d) packet (MAC) latency data, which is assumed to be 20 ms per hop based on 802.15.4 standard.

The sensor network used for our analysis is shown in Fig. 4(b). There are eight nodes in the network, one of which is the base station. All the nodes are configured as full function devices (FFD). The network is configured in the *Mesh* mode (one of the three modes in ZigBee specification) and it uses AODV (Ad hoc On-Demand Distance Vector Routing) routing protocol to forward data to the base station. We perform Monte-Carlo simulation using the MAUI reliability data to simulate node aliveness. The process, as illustrated in the Fig. 6, comprises four basic steps: (a) get node aliveness data at a certain time from MAUI hardware data; (b) throw a dice based on the aliveness probability to decide whether a node is alive or dead, assuming node aliveness follows Bernoulli distribution; (c) once a node is found to be alive, its link quality is queried using several node properties, such as the transmission power level, time, and channel number; and finally (d) calculate end-to-end reliability. In a similar way we also calculate end-to-end latency and system life time.

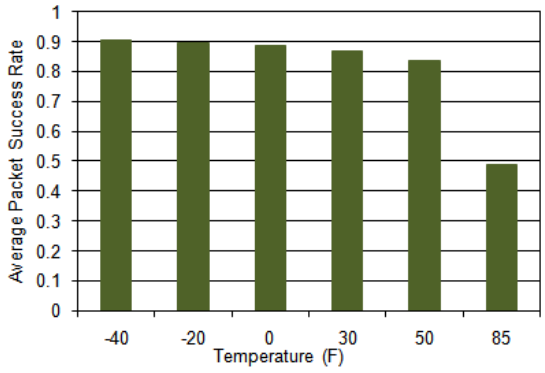


Fig. 7. Packet success rate with temperature.

A. Packet Success Rate

We evaluate the dependency of packet success rate (PSR), measured as a percentage of the average number of packets delivered by all the nodes to the sink, with different parameters, such as temperature, channel id, and transmission power level. Fig. 7 shows that temperature has a major impact on

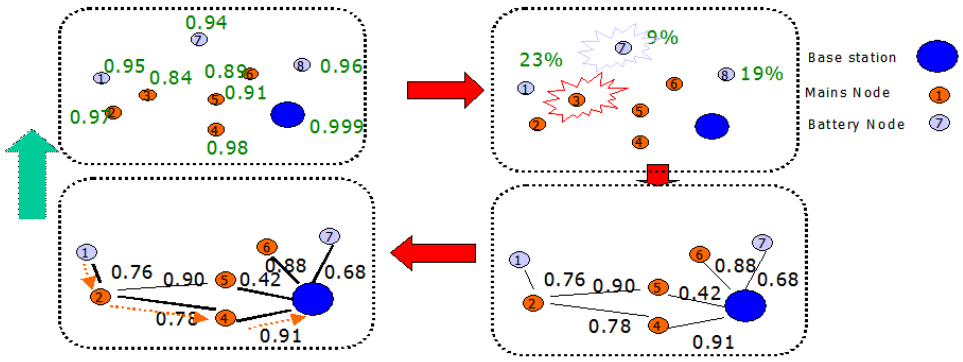


Fig. 6. Sensor network configuration and dependability analysis methodology. Top-left: node aliveness probability from MUAI data; top-right: node 3 is dead and hence not taken into account; bottom-right: link qualities determined; bottom-left: AODV routing path calculated.

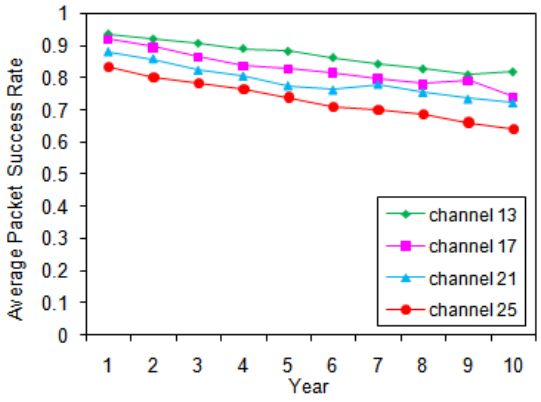


Fig. 8. Packet success rate for different channels with time

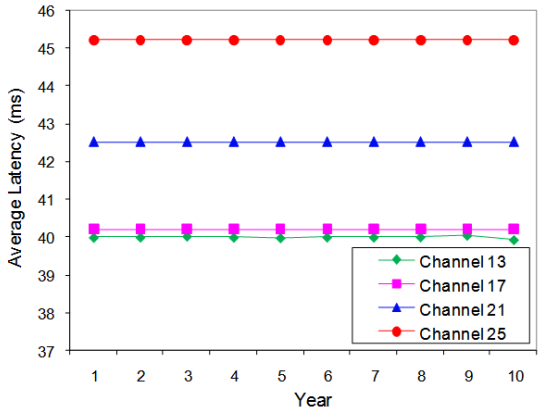


Fig. 10. Average end-to-end latency with time for different channels

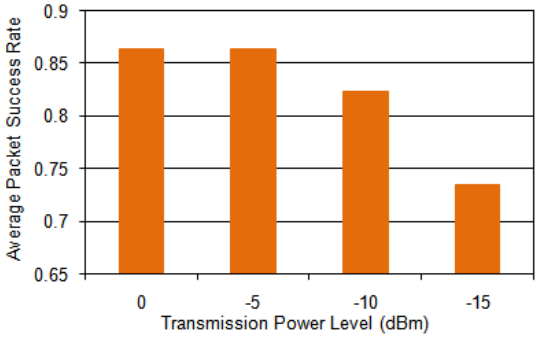


Fig. 9. Packet success rate with transmission power

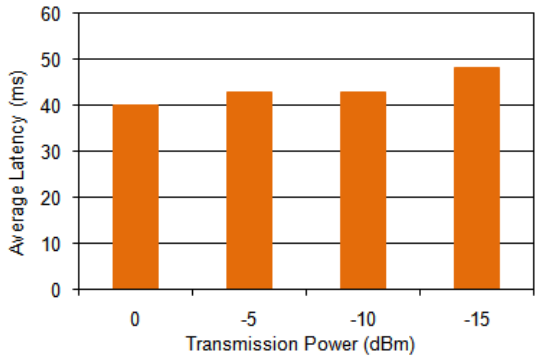


Fig. 11. Average end-to-end latency with transmission power

PSR - at high temperatures the end-to-end reliability decreases substantially, and is observed to be as low as 50% at 85 F. Fig. 8 shows the variation of PSR over time when different channels are used to communicate. At higher channel numbers (higher frequencies) the 802.15.4 signals interfere more with other signals, such as 802.11, and hence have reduced PSR. Lastly, Fig. 9 shows that the PSR decreases as the transmission power is reduced from 0 dBm to −15 dBm. This is due to the fact that at lower power levels nodes can communicate over fewer hops, and thus the number of hops required to reach the sink increases, therefore, reducing the end-to-end PSR.

B. Average Latency

We evaluate the average end-to-end latency for all packets delivered to the sink node and plotted its variation with time for different channels in Fig. 10, and with different transmission power levels in Fig. 11. As before, we again observe that the latency increases at higher channel numbers due to increased interference and also at lower power levels due to increased number of hops to the sink.

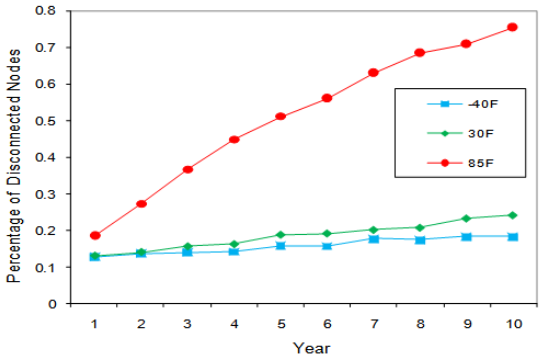


Fig. 12. Average percentage of disconnected nodes in the first ten years for different temperatures

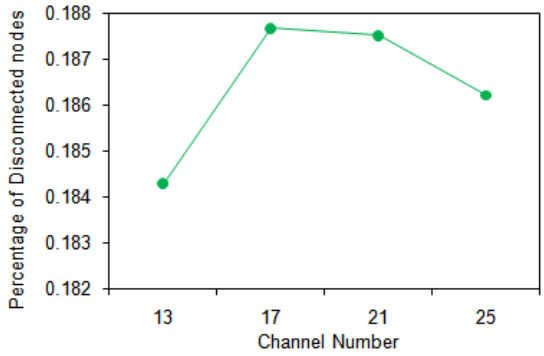


Fig. 13. Average percentage of disconnected nodes for different channels

C. System Life

Lastly, we evaluate system life time as the average percentage of disconnected nodes with time and for different channels. The corresponding graphs are shown in Fig. 12 and Fig. 13, respectively. We observe that at higher temperatures the number of disconnected nodes increases sharply.

VI. Wireless Sensor Network Specific AADL Extensions

Although, in general, AADL provides a very good set of syntactic and semantic elements to support the modeling, design, and analysis of real-time embedded systems, to model wireless sensor networks in particular, the language as described in the first AADL standard needs to be extended to include additional WSN specific features. In this section we describe several aspects of the needed extensions.

A. Multiplicity Support

In most realistic scenarios a typical WSN deployment has hundreds to thousands of nodes. A homogeneous WSN consists of nodes that are very similar in hardware and software configurations, whereas a heterogeneous WSN may contain multiple significantly different types of nodes in both software and hardware. Moreover, for heterogeneous WSN the nodes can be categorized into small number of groups with nodes in each group of very similar nature. For such systems it is cumbersome to specify the model for each node individually since many of them are of identical configurations. This *multiplicity* issue is a challenge in the original AADL standard (version 1). The proposed AADL version 2 standard to be rectified in 2008 defines the syntax and semantics for array, which we believe will be very useful for WSN modeling.

B. Location Support

Besides names or IDs, for nodes with similar hardware and software configurations, the most important difference among them is their physical locations. Locations have significant impact on wireless communications, and therefore, play a very important role in the run-time functionality of both the network and an individual node. Based on the concrete application scenarios, we need to provide flexible means for the user to specify the locations of the nodes with various granularities. In some scenarios, the users may know the exact location of each node, while in others, the users may only know the density of the node deployment. We propose the following means: (a) specifying each individual location in the AADL model; (b) importing the locations from a file, whose path and name are specified in the AADL model; and (c) applying a specific deployment topology to an area specified in the AADL model. The deployment topology can be a rectangular grid or based on a statistical distribution, and tools (e.g., OSATE plug-ins) need to be implemented to take the topology and related parameters to allocate the detailed locations of the nodes in the network.

C. Wireless Channels Support

Realistic modeling of wireless channels is one of the most challenging tasks in a sensor network design. It is well known that radio signal strength is affected by various environmental factors, such as noise, interference, multi-path, and shadowing effects, as a result of which links could be asymmetric and stochastic in nature. Nodes that are very close to each other usually have strong links with high packet reception rates (PRR), while nodes that are farther apart have weak links with low PRR. However, nodes that are at medium distance away from each other exhibit random link characteristics. This region, where the uncertainty in the quality of wireless links is highest, is known as the transitional region [31], and depending on the output power levels and multi-path effects it can extend a wide range. Accurate means of specification of such an unreliable communication medium is a very important factor in WSN design that does not exist in the AADL standard. Since the quality of wireless links vary so much, it is cumbersome to specify all of them explicitly in an AADL model. One approach to take the per-link performance into account for the WSN model is based on empirical measurements. We build an AADL-based tool to get link level performance from empirical measurements into further simulation and analysis through a file interface. However, in reality the empirical measurement results are not always available, especially in the design phase prior to network deployment. In this case, it is easier to specify a well-known wireless model with related parameters in the AADL model. Tools need to be built to read the wireless model, related parameters, and other information such as node locations, infer the per-link performance and apply it to the model.

VII. Conclusions

In this paper we have highlighted the benefits of architecture description languages over traditional modeling and simulation tools, and in particular, advocated the use of AADL in describing wireless sensor network architectures. We demonstrated through a case study that such an architecture driven approach provides great flexibility in tuning system parameters prior to implementation as per application specific requirements, in particular, with respect to three dependability metrics: packet success rate, latency, and system life time. We also proposed sensor network specific extensions to the basic AADL language that would allow modeling WSN architectures in a richer way. Reliability and dependability analysis of larger systems with detailed error modeling based on extensive real-life data and richer modeling of wireless links are part of our future work.

Acknowledgment

The authors would like to thank Prof. Bhaskar Krishnamachari of the Autonomous Networks Research Group at USC and the members of the Innovation Center at Eaton Corporation for their valuable inputs and support. The work described here is supported in part by DoE and NSF grants CNS-0627028 and CCF-0430061. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DoE or NSF.

References

[1] Robert Allen and David Garlan, "A Formal Basis for Architectural Connection", In *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 3, pp. 213–249, July 1997.

[2] Archrock: http://www.archrock.com

[3] Pam Binns, Matt Englehart, Mike Jackson, and Steve Vestal, "Domain Specific Software Architectures for Guidance, Navigation and Control", In *International Journal of Software Engineering and Knowledge Engineering*, vol. 6, no. 2, pp. 201–227, June 1996.

[4] Grady Booch, "Object Oriented Development", *IEEE Transactions on Software Engineering*, vol. 12, no. 2, pp. 211–221, February 1986.

[5] Crossbow: http://www.xbow.com

[6] Peter Feiler, Bruce Lewis, and Steve Vestal, "The SAE Architecture Analysis and Design Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering", In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium, Workshop on Model-Driven Embedded Systems (RTAS'03)*, pp. 1–10, Washington, D.C., May 2003.

[7] Peter Feiler, David Gluch, John Hudak, and Bruce Lewis, "Embedded System Architecture Analysis Using SAE AADL", *CMU Technical Report CMU/SEI-2004-TN-005, http://www.aadl.info*, June 2004.

[8] Peter Feiler, Aaron Greenhouse, and Lutz Wrage, "Open Source AADL Tool Environment (OSATE) Plug-in Development Series", *http://www.aadl.info*, Software Engineering Institute, CMU, Dec 2004.

[9] Peter Feiler and Ana Rugina, Error Modeling with AADL, Technical Report, 2006.

[10] David Garlan, Robert Monroe, and Dave Wile, "ACME: An Architecture Description Interchange Language", In *Proceedings of the 1997 Conference of the Center for Advanced Studies on Collaborative Research (CASCON '97)*, pp. 169–183, Toronto, Ontario, November 1997.

[11] Glomosim: http://pcl.cs.ucla.edu/projects/glomosim/

[12] David Gluch, Peter Feiler, and Bruce Lewis, "Tutorial: Hands-On Experiences with the SAE Standard Architecture Analysis and Design Language in High Dependability Design", *International Conference on Dependable Systems and Networks (DSN'05)*, Yokohama, Japan, June 2005.

[13] Grenada: A Tool for Assessment of IEEE 802.15.4 Link Quality in Harsh Environments, http://www.rawcon.org/rws2007/wmon3.html

[14] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister, "System Architecture Directions for Networked Sensors", *ACM SIGOPS Operating Systems Review*, vol. 34, no. 5, pp. 93–104, 2000.

[15] ISA 100: http://www.isa.org

[16] J. W. Krueger, Steve Vestal, and Bruce Lewis, "Fitting the Pieces Together: System/Software Analysis and Code Integration using MetaH", In *Proceedings of the 17th AIAA/IEEE/SAE Digital Avionics Systems Conference (DASC'98)*, Bellevue, WA, October 1998.

[17] B. Lu, L. Wu, T. G. Habetler, R. G. Harley, and J. A. Gutierrez, "On the Application of Wireless Sensor Networks in Condition Monitoring and Energy Usage Evaluation for Electric Machines", In *Proceedings of the 31st Annual Conference of the IEEE Industrial Electronics Society (IECON'05)*, pp. 1737–1742, Raleigh, NC, November 2005.

[18] B. Lu, T. G. Habetler, R. G. Harley, and J. A. Gutierrez, "Applying Wireless Sensor Networks in Industrial Plant Energy Management Systems", *IEEE Sensors*, October 2005.

[19] David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, and Walter Mann, "Specification and Analysis of System Architecture Using Rapide", In *IEEE Transactions on Software Engineering, Special Issue on Software Architecture*, vol. 21, no.4, pp. 336–355, April 1995.

[20] David Malan, Thaddeus Fulford-Jones, Matt Welsh, and Steve Moulton, "CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care", In *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks (BSN'04)*, Imperial College London, United Kingdom, April 2004.

[21] Matlab, Simulink: http://www.mathworks.com

[22] ns2: http://www.isi.edu/nsnam/ns

[23] Opnet: http://opnet.com

[24] Rational Rose: http://www.ibm.com/software/awdtools/developer/rose

[25] Rhapsody: http://www.ilogix.com/rhapsody/rhapsody.cfm

[26] Software Systems Architecture: http://www.eoinwoods.info/ot2004/RozanskiWoods-ViewpointsAndPerspectives.pdf

[27] Society of Automotive Engineers (SAE), "Architecture Analysis and Design Language (AADL)", *Document Number AS5506, Warrendale, PA: Society of Automotive Engineers*, November 2004.

[28] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin, "A wireless sensor network For structural monitoring", In *Proceedings of the second International Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, pp. 13–24, November 2004.

[29] Ting Yan, Tian He, and John A. Stankovic, "Differentiated Surveillance for Sensor Networks", In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*, pp. 51–62, Los Angeles, CA, November 2003.

[30] ZigBee: http://www.zigbee.org

[31] Marco Zuniga and Bhaskar Krishnamachari, "Analyzing the Transitional Region in Low Power Wireless Links", In *Proceedings of the First IEEE International Conference on Sensor and Ad hoc Communications and Networks (SECON'04)*, pp. 517–526, Santa Clara, CA, October 2004.