Hybrid Annex: An AADL Extension for Continuous Behavior and Cyber-Physical Interaction Modeling

Ehsan Ahmad
School of Computer Science
Northwestern Polytechnical University,
State Key Lab. of Comput. Sci. Inst. of Software
Chinese Academy of Sciences
ehah@ios.ac.cn

Brian R. Larson
Computing and Information Systems
Kansas State University
Manhattan, KS 66506
brl@ksu.edu

Stephen C. Barrett
Computing and Information Systems
Kansas State University
Manhattan, KS 66506
scbarrett@ksu.edu

Naijun Zhan
State Key Lab. of Comput. Sci. Inst. of Software
Chinese Academy of Sciences
Beijing, 100190
znj@ios.ac.cn

Yunwei Dong
School of Computer Science
Northwestern Polytechnical University
Xi'an, 710129
yunweidong@nwpu.edu.cn

ABSTRACT

Correct design, and system-level dependability prediction of highly-integrated systems demand the collocation of requirements and architectural artifacts within an integrated development environment. Hybrid systems, having dependencies and extensive interactions between their control portion and their environment, further intensify this need.

AADL is a model-based engineering language for the architectural design and analysis of embedded control systems. Core AADL has been extended with a mechanism for discrete behavioral modeling and analysis of control systems, but not for the continuous behavior of the physical environment. In this paper, we introduce a lightweight language extension to AADL called the Hybrid Annex for continuous-time modeling, fulfilling the need for integrated modeling of the computing system along with its physical environment in their respective domains. The Isolette system described in the FAA Requirement Engineering Management Handbook is used to illustrate continuous behavior modeling with the proposed Hybrid Annex.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*continuous behavioral modeling*

General Terms

Design, Languages, Reliability

Keywords

AADL annex; continuous behavior modeling; cyber-physical; hybrid annex; Hybrid CSP; hybrid systems

1. INTRODUCTION

Integrated dynamical systems, where computing units of discrete dynamics interact with a physical world possessing continuous dynamics are known as *hybrid systems.* Such systems interact with their external environment so as to monitor and control those physical quantities necessary for ensuring correct system functionality. These physical quantities are often termed *controlled variables* in the parlance of embedded-systems engineers.

The behavior of a computing unit is described by its responses to discrete events within a countable set of *states* $\{s_1, s_2, ...s_n\}$ where $n \in \mathbb{Z}$. Ordered events along with certain real-time properties are used to model the behavior of a hybrid system's computing units. Behavior of the physical portion, on the other hand, concerns a Euclidean space $\mathbb{R}^n$ where $n \geq 1$, and is specified using continuous domain differential equations. Obtaining these equations can make the definition of critical issues related to physical quantities a challenging task.

Hybrid systems complicate the matter further with external changes in the environment's continuous domain introducing behavior variation into the computing unit's discrete domain, and vise-versa: evolution of physical quantities on the continuous side of the system can trigger events on the discrete side, while events can, in turn, interrupt the evolution of continuous physical quantities by replacing one set of specifying differential equations with another. This circular dependency greatly increases the difficulty of correct hybrid system design and development, especially with regard to timing, safety, and reliability related properties.

Model-Based Engineering (MBE) is considered to be an effective way of developing correct, complex safety-critical systems, and has been successfully employed to that end in the embedded-systems industry [11, 20]. We contend that in order to formally specify, model, and take full advantage of advancements in MBE for dependability prediction and hybrid system certification, the requirements related to both discrete and continuous behaviors need to be collocated in a single, integrated development environment. The Architecture Analysis & Design Language (AADL)—a description language for embedded systems based on the MBE paradigm—is a strong candidate for the modeling of highly-integrated systems.

AADL provides abstractions for *components* and their *connections*. Additionally, it supports precise behavior modeling with extensive analysis at various architectural levels. Static architectures are specified as hierarchical compositions of interconnected components, the internal structures of which are themselves formed from interconnected (sub)components. A dynamic architecture, on the other hand, is modeled by presenting the *modal* behavior of the system. Modes contain the component and connection configurations for different operational and error states. Due to its extensive support for modeling (abstraction, reusability, composition, etc.) and its substantial analysis capabilities, AADL is being successfully used by embedded system designers in aircraft manufacturing.

Unfortunately, core AADL only provides mechanisms for modeling the discrete behavior of a computing unit (i.e., the control software and the platform on which it runs), and nothing at all related to the behavior of the physical process to be controlled. Hence, to equip AADL for hybrid system modeling and analysis, the core language needs to be extended. A predefined language extension mechanism makes the specification of such an *annex* possible.

In this paper, we propose a *Hybrid Annex* for AADL, a lightweight language extension for specifying the continuous behavior of model components. Based on the idea of Hybrid CSP (Communicating Sequential Processes) [9, 26, 24], the annex allows for the modeling of the continuous behavior of physical processes external to the system being designed with which the system's sensors and actuators interact.

The next section presents AADL and motivates the need for a dedicated continuous behavior modeling annex. Section 3 introduces the example system used to illustrate the Hybrid Annex specification for continuous time modeling. The proposed sublanguage and its grammar are discussed in Section 4, with the example system used to detail its constructs. Section 5 demonstrates behavior constraint specification using BLESS Assertions. Section 6 presents related work, while Section 7 concludes the study.

2. BACKGROUND AND MOTIVATION

Architecture describes how a system is decomposed into constituent parts and the ways in which those parts interact. It is a "prudent partitioning of a whole into parts, with specific relations among the parts" [5]. Traditionally, it has fallen to informal box-and-arrow drawings to communicate a system's decomposition. Despite their failings, such elementary notations served their purpose, but flourishing research in the area of software documentation has pointed to better ways. One promising line of inquiry has resulted in domain-specific architecture description languages, of which AADL is an exemplar.

2.1 Overview of AADL

AADL is an SAE International standard language for the architectural description of embedded systems [16]. It is an architecture-centric, model-based engineering approach that was introduced to cope with embedded system design challenges. AADL strives to minimize model inconsistency, decrease mismatched assumptions between stakeholders, and support dependability predictions through analyzable architecture development [7]. Several safety-critical industrial case studies in domains like medical and aerospace engineering have used AADL for system architecture design and analysis.

The important collaborative System Architecture Virtual Integration (SAVI) project for designing complex distributed aerospace systems has selected AADL as its lingua franca [8]. SAVI emphasizes an *"Integrate, Then Build"* approach—the key concept being to verify virtual integration of architectural components *before* implementing their internal designs. AADL supports virtual integration with an effective mechanism for component contract specification based on interfaces and interactions, and through well defined semantics for extensive formal analysis at different architecture levels.

2.1.1 System Architecture Modeling

Architectural modeling in AADL is realized through the component specification of both the *application software* and the *execution platform* it is to run on. Component *Type* and *Implementation* declarations, or classifiers, corresponding to system entities are instantiated and then connected together to form the system architecture model.

Component interface elements, called *ports*, are specified in the *features* section of a type classifier. AADL provides *data*, *event* and *event data* ports to transmit and receive data, control, and combined control and data signals, respectively. Port communication is typed and directional. The externally observable attributes of a component are specified in the *properties* section of its type.

An implementation classifier defines a particular internal structure of the component by specifying its subcomponents and the connections between them. Application software may contain *process*, *data*, *subprogram*, *thread*, and *thread group* components. The process component represents a protected memory space shared among thread subcomponents. A data component represents a type, local data subcomponent, or parameter of a subprogram, i.e., callable code. A thread abstracts sequential control flow.

The execution platform is made up of computation and communication resources, consisting of *processor*, *memory*, *bus*, and *device* components. The processor represents the hardware and software responsible for thread scheduling and execution. The memory abstraction is used for describing code and data storage entities. Devices can represent either physical entities in the external environment, or interactive system components like actuators and sensors. Physical connections between execution platform components are accomplished via a bus component.

2.1.2 System Behavior Modeling

AADL core language is extendable: additional sublanguages for modeling and analysis can be added through its *annex* mechanism. For example, standardized Data Modeling and Error Modeling annexes have been introduced to associate architectural components with data and error models, respectively, and an ARINC653 annex was added for defining ARINC653-compliant system architectures.

The component and connection constructs of AADL are sufficient for modeling the structure of a system architecture. However, the extensive formal analysis needed for dependability prediction requires detailed behavior modeling, which AADL lacks. The Behavior Annex (BA) and BLESS were introduced to address this shortcoming [13, 18]. They both use state transition mechanisms with guards and actions to model the discrete behavior of control systems. To prove correctness, BLESS adds a tool for the automatic generation of proof obligations and interactive theorem proving, based on temporal logic formulas specified with *Assertions*[1].

2.2 Motivation

Most systems exist to control *something*. Any system controller, whether human or automated, must know the current state of the process being controlled, and be able to judge the effect on that state of any control actions it might take. For this kind of awareness, a controller must either be, or contain, a model of the entirety under consideration: that is, the process being controlled and the controller's role in doing so.

This so-called *process model* is what supplies a snapshot of the system's condition. It can vary from having one or two variables, to defining control laws, to being a very complex model with a large number of state variables and transitions. A valid process model is essential to the proper and safe operation of a controller. According to Levenson, "...many accidents have been caused by the controller incorrectly assuming the controlled system was in a particular state and imposing a control action (or not providing one) that led to a loss" [22]. Causality and hazard analyses like STAMP and STPA also rely extensively on knowing about the process.

As most digital controllers interact with, or try to control some aspect of the physical world, they are, by definition, hybrid systems. Realizing the promise of MBE (system analysis, code generation, implementation transformation, etc.) necessitates an ability to describe the behavior of the process model. In a hybrid system, this entails enumeration of discrete events for the controlling part, and continual evaluation of differential equations for the real process.

So much for a system's realization, what about its design? According to Heimdhal et al. [10], approaching the "Twin Peaks" design activities of requirements and architecture through modeling can uncover, and help better understand the requirements (e.g., rate of change, settling time, cumulative error propagation) needed to adequately constrain desired system behavior. Again, we see the need for behavior modeling.

Additionally, in the case of a hybrid system, extensive interaction between the embedded computing unit and its environment, and their mutual dependence on each other intensifies the need for *integrated* requirements specification and design modeling. The contributions of such an integration at the requirements specification and early design (i.e., architecture) stages are twofold. Firstly, it supports requirement identification for both discrete and continuous variables. And secondly, correct operation of the physical portion of the system can be assessed through several dependability related analyses, allowing for the systems level correctness certification of a hybrid system.

Capturing system requirements and providing controller process models establishes the need for being able to model the behavior of real world entities (i.e., continuous domain), while describing system architectures requires an ability for modeling computing unit behavior (i.e., discrete domain). To fully understand how the behaviors in one domain influence those in the other demands an integrated approach to the modeling of the computing units and the physical environment of the respective domains. It is for this purpose that we propose the Hybrid Annex to AADL.

[1]The capital 'A' proper noun signifies temporal logic formulas used by BLESS

3. EXAMPLE SYSTEM

In the Requirement Engineering Management Handbook (REMH) [21], its guide to managing requirements for embedded systems, the Federal Aviation Administration (FAA) describes an infant incubator known as an *isolette*. Because the specification is simple enough to grasp, yet rich enough to highlight the need for our proposed annex, we use it to demonstrate application of the Hybrid Annex notation to modeling the continuous dynamics of interactions between a control system and its environment.

The isolette example has previously been used to introduce important research efforts, and to advocate for AADL-based development and new annexes. It has been used by Blouin to illustrate the Requirements Annex [3], by Larson to explain detailed behavior modeling with the BLESS Annex [18], and to demonstrate hazard analysis techniques using the Error Model Annex(v.2) [19].

3.1 Isolette Operation

The context diagram of Figure 1 depicts a classical control loop with controller, actuator, controlled process, and sensor units. The system exists to maintain the temperature of the *Air* in the Isolette—a physical process—for the benefit of the infant, within a desired range as set by the Nurse through the *Operator Interface* and controlled by the *Thermostat*. We focus on modeling the continuous behavior of the Air, and its interactions with the *Heat Source* and *Temperature Sensor* units. The internals of the controller (i.e., the Thermostat and Operator Interface) are not considered.

The Thermostat monitors the Air temperature through the Temperature Sensor, and attempts to manipulate it with the Heat Source actuator. The *control strategy* followed by the Thermostat, is derived from a process model that is implicit in the interpretations it gives to the current Air information coming from the sensor, and the commands it has previously issued to the actuator. Making any part of this loop digital makes the overall system a hybrid.

The amount of heat required of the Isolette Heat Source depends on the rate at which the Air component cools. Observations about the environment in which the system is to operate, and upon which its correct operation depends are termed *environmental assumptions*. Other relevant properties about the environment for this relationship might include construction material and dimensions of the incubator, and body size and skin temperature of the infant.

The continuous behavior of the Air's changing temperature, which the Heat Source must work to balance can most easily be assumed to follow the differential equation known as Newton's law of cooling (or heating).[2]

$$\dot{T_o} = -k \cdot (T_o - T_e)$$

[2]The effect of heat transfer between the Isolette and the

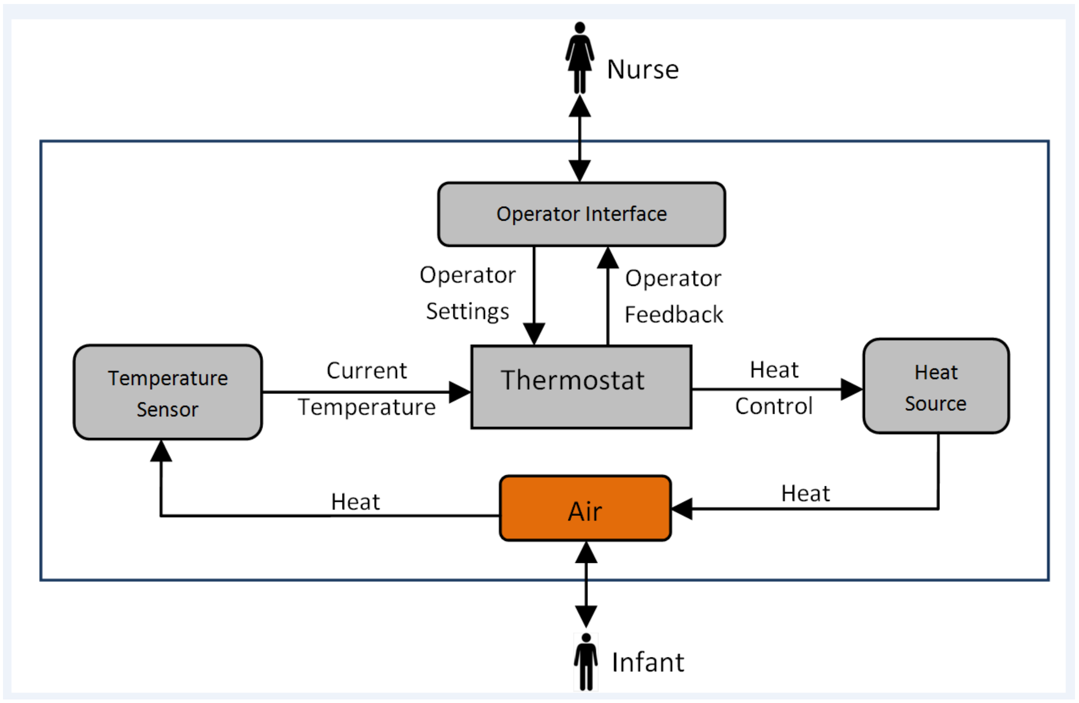


Figure 1: Isolette Context Diagram with Controller (Thermostat) and Physical Environment (Air)

The law states that the rate of change in temperature of an object $\dot{T_o}$ at time t is directly proportional to the difference in temperature between the object T_o and its environment T_e. Proportionality constant k is the thermal conductivity of the object, which depends on the physical properties of the object i.e., more environmental assumptions. The sign of the constant indicates whether the object is cooling down (-), or warming up (+).

The specification in Section A.5.1.3 of the REMH [21] provides us with two environmental assumptions concerning the continuous behavior of Air temperature change in the Isolette:

EA-IS1: *When the Heat Source is turned on and the Isolette is properly shut, the Current Temperature will increase at a rate of no more than* 1°F *per minute.*

EA-IS2: *When the Heat Source is turned off and the Isolette is properly shut, the Current Temperature will decrease at a rate of no more than* 1°F *per minute.*

The continuous change in current Air temperature $\dot{c}$ (i.e., heating or cooling of the Isolette) depends on the current status of the Heat Source, and, if **on**, the heat q being produced by it. Assumptions **EA-IS1** and **EA-IS2** lead us to set the change in q equal to a rate of 1°F per unit time, as summarized here:

$$\begin{cases} \dot{c} &= -0.026 \cdot (c - q) \\ \dot{q} &= 1 \quad \text{if heater is on} \\ \dot{q} &= -1 \quad \text{if heater is off} \end{cases}$$

infant's body can be modeled by using Fourier's law of conduction: $\dot{Q} = -kA\frac{dT}{dx}$.

where c is the current temperature of the Air, 0.026 its thermal conductivity k, and q the temperature of the heater.

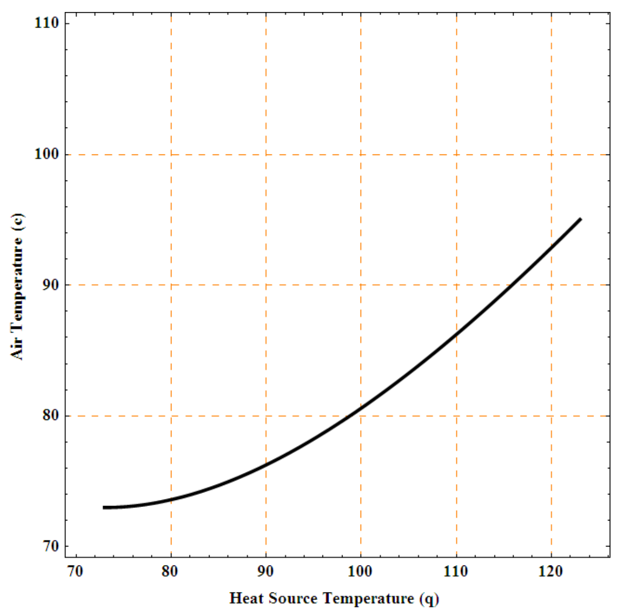
An initial plot of Air temperature behavior can be seen in the graph of Figure 2(a) with the vertical axis denoting the temperature of the Air (c), and the horizontal axis the temperature being output by the Heat Source (q). With the system quiescent, both c and q are equal to the room temperature, assumed to be a constant 73°F. However, when the system controller turns on the Heat Source, the resulting rise in q forces a consummate rise in c.

The observed effects of the Isolette operating as intended are given by Figure 2(b). Starting from the extreme *Upper Desired Temperature* of 100°F as mentioned in Section A.3.4 of the REMH, curve 'm' exhibits first the cooling of the Air, since the Heat Source is **off**, and then, once the heat goes **on**, the subsequent rise in its temperature. Conversely, curve 'n' tracks the heating of the Air from the minimum *Lower Desired Temperature* of 97°F, only to cool when the initially **on** Heat Source is turned **off**.

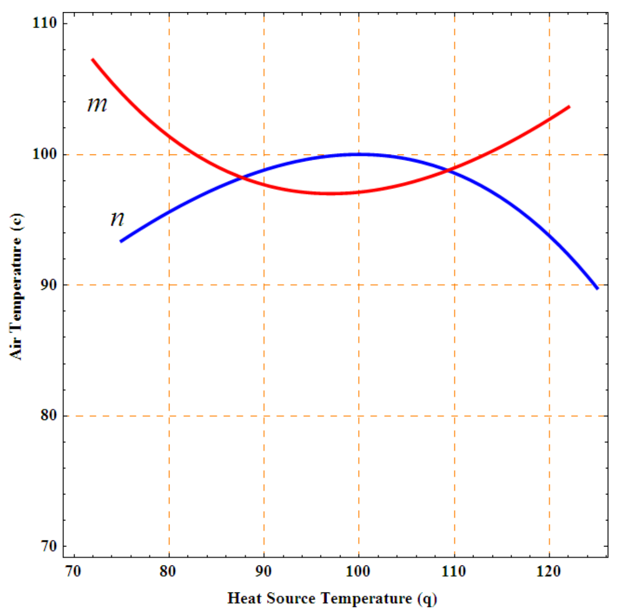
The control strategy of the Thermostat is to turn the Heat Source **on** or **off**, depending on the desired direction of temperature change, at the points where the 'm' and 'n' curves intersect. This gives the system the time needed to overcome the Heat Source *latency* (Section A.5.1.3 of the REMH) and thereby keep the current temperature c within the desired temperature range (in this case [97°..100°F]) as set by the clinician through the Operator Interface.

Modeling the just described physical behavior of the Isolette example with the proposed Hybrid Annex is detailed in Section 4 along with an AADL *implementation*. As an introduction, the Air AADL *type* is first presented here.

The type classifier of Listing 1 declares the interface of the Air component as an AADL *abstract* component. The `hss` *in data port* is used to get the current status of the Heat Source



(a) Initial behavior



(b) Cooling and heating behavior

Figure 2: Physical Behavior of the Air Component

while the `hin` port is used to get a measure of the amount of heat being produced. The Heat Source is linked to `hss` and `hin` through appropriate port connections. The `tout` *out data port* communicates the heat energy in the Air to the Temperature Sensor for measurement. Type `Heat` references an AADL Data Model Annex component defined in package `Iso_Variables` that specifies the range of possible values a variable of this type can take on, and its unit of measure. The corresponding implementation classifier for the Air type is given in Listing 2 and will be considered in the next section in the context of the Hybrid Annex.

Listing 1: AADL Air Component Type

```
abstract Air
  features
  hss: in data port Iso_Variables::on_off;
  hin: in data port Iso_Variables::Heat;
  tout: out data port Iso_Variables::Heat;
end Air;
```

4. HYBRID ANNEX

In order to equip AADL for hybrid system modeling and analysis, we propose a lightweight extension to the language named the Hybrid Annex (HA). The annex brings with it the ability to model those physical, real-world elements, or processes, that the system must interact with to achieve its goals of monitoring and controlling one or more of those processes. The annex subclause grammar and semantics are inspired by an extension of CSP known as Hybrid Communicating Sequential Processes (HCSP) [9, 26] with the added intention of impartially supporting other continuous behavior modeling tools and methodologies. Formal syntax and operational semantics of HCSP are detailed in [24].

In use, HA subclauses annotate either AADL *device* component implementations in order to model the continuous behavior of sensors and actuators, or *abstract* component implementations so as to model the continuous behavior of physical processes.

An HA specification may consist of up to six sections: `assert`, `invariant`, `variables`, `constants`, `channels`, and `behavior` to specify: assertions, or predicates; a single assertion that must hold throughout operation of the continuous behavior model; local variables; constants; communication channels; and continuous behavior, respectively.

HA was first presented to the SAE International AADL Standards Committee AS-2C at Santa Barbara, on April 14-17, 2014 [6]. The first draft of the proposed HA annex standard document was considered, line-by-line, in committee at Orlando on July 7-10, 2014.

4.1 Continuous Behavior Modeling

What follows are details for completing the various sections of an HA behavior specification, examples of which appear in Listing 2 where they are expressed in standard AADL notation [16]. For each section, the Extended Backus-Naur Form (EBNF) of the HA grammar is also given, in which: literals are printed in **bold**; alternatives are separated by a pipe |; groupings are enclosed with parentheses ( ); square braces [ ] delimit optional elements; and the closures { }+ and { }* are used to signify one-or-more, and zero-or-more of the enclosed element, respectively.

4.1.1 Assert Section

HA provides an `assert` section for declaring predicates applicable to the intended continuous behavior of the annotated AADL component. These predicates take the form of BLESS Assertions, and may be used later in defining the invariant (see the next subsection). The grammar and semantics for BLESS Assertions may be found in [17].

4.1.2 Invariant Section

The `invariant` section is used in conjunction with the declarations made in the assert section to define a condition of operation that must hold true throughout the model's execution lifetime. Note that there is only *one* invariant, but it can be logically complex, having as many terms as needed.

The HA assert and invariant sections are touched upon again in Section 5.

4.1.3 Variables Section

Local variables in the scope of an Hybrid Annex subclause are declared in the `variables` section along with their data types. Depending on which AADL component the HA subclause has been applied to, a variable will hold either a discrete or continuous value. Following is the grammar for the HA `variables` section:

```
variable_declaration ::=
variable_identifier
{ , variable_identifier }* :
data_component_classifier_reference
```

A data type is assigned by a classifier reference to the appropriate AADL data component. The referenced external data component must either be part of the package containing the component being annotated, or must be declared within the scope of another package that has been imported using the AADL keyword, `with`.

The `variables` section of the HA specification for the example Isolette system appears in Listing 2. The variables were identified through consideration of environmental assumptions drawn from those system requirements that relate to the continuous behavior of the physical processes the Isolette is meant to interact with, namely, the Air. Variable `c` represents the monitored variable *Current Temperature*, variable `h` is either `0` or `1` to represent the **off** or **on** state of the *Heat Control* controlled variable, and `l` and `u` represent the *Lower Desired Temperature* and *Upper Desired Temperature* controlled variables, respectively, as defined in the REMH [21]. Variable `q` holds the value of the heat, if any, being generated by the Heat Source.

4.1.4 Constants Section

Similar to local variables, constants in the scope of an HA subclause are declared in the `constants` section. Adhering to standard convention: constants can only be initialized at declaration, and cannot be assigned another value afterwards. The grammar for the `constants` section is as follows:

```
constant_declarations ::=
behavior_constant
{ , behavior_constant }*

behavior_constant ::=
behavior_constant_identifier =
( integer_literal | real_literal )
[ unit_identifier ]
```

A constant must be initialized with either an integer, or a real value, and may include a description of its unit of measure.

The `constants` section of Listing 2 contains declarations for the constants `r` and `k` that represent the room temperature and the thermal conductivity of the substance being modeled, which of course is the air in the room. The room is assumed to be held at a constant temperature of 73°F. Keeping in mind that the value of the monitored *Current Temperature* variable can vary from 68°F to 105°F lets us assign a value of 0.026 as the average air thermal conductivity to constant `k` of the Air implementation.

HA also supports the specification of measuring units that have been defined using the AADL Unit Relation Annex [4]. As a result, common constants like the mathematical ratio π and the physical gravitational attractive force g can be easily declared in HA with `pi = 3.14159` (no units) and `g = 9.81 mpss` (*meters per second*^{2}), respectively. In our model, the measuring unit for temperature is °F, indicated in the model with an `f`, and the measuring unit of thermal conductivity is *watts per meter kelvin* ($W/(mK)$), denoted for the constant `k` as `wpmk`.

4.1.5 Behavior Section

The `behavior` section of the HA subclause is used to specify the continuous behavior of the annotated AADL component in terms of concurrently-executing processes. Thus, a behavior declaration has process declarations, which in turn, may contain several predefined executing processes of various topologies (sequential, concurrent, repetitive, etc.). The process algebra notation that models reactive system behavior as communication flows is documented below.

```
behavior_declaration ::=
behavior_identifier ::=
process_declaration
{ & process_declaration }*

process_declaration ::=
skip | wait time_value | assignment
| communication | sequential_composition
| concurrent_composition | choice
| continuous_evolution | repetition
```

Behavior of a physical controlled variable of a hybrid system is specified by continuous evolution—a differential expression controlled optionally by a Boolean expression. Differential expressions consist of several derivative expressions combined with standard multiplication ($*$), addition ($+$) and subtraction operators ($-$). A derivative expression is indicated using the keyword `DE` followed by the order of the differential equation, then the dependent variable, and finally the independent variable. For example, the rate of change of variable y with respect to x, denoted $\frac{dy}{dx}$, a first order equation, is specified as `DE 1 y x`, while the second order equation $\frac{d^2y}{dx^2}$ is specified with `DE 2 y x`. A similar notation is defined for time derivation, a frequently encountered concept in real-time systems. Here the keyword is `DT`, and the independent variable, always being time, is not needed. Thus, the rate of change of y with respect to time t, $\frac{dy}{dt}$, is stated `DT 1 y`. The grammar for the continuous evolution process is defined as follows.

```
continuous_evolution ::=
'differential_expression =
differential_expression'
[ < boolean_expression > ] [ interrupt ]
```

Listing 2: AADL Air Component Implementation with Hybrid Annex Specifications

```
abstract implementation Air.impl
annex hybrid {**
  assert
    <<NORMAL: : (c@now < (u+Iso_Properties::Tolerance)) and  --air temp normal range
      (c@now > (l-Iso_Properties::Tolerance))>>
    <<EA_IS_1: : forall x:BLESS_Types::Time in 0.0 ,, now are  --limit rate of heating
      (c@now - c@x) <= Iso_Properties::Heat_Rate*(now-x)>>
    <<EA_IS_2: : forall x:BLESS_Types::Time in 0.0 ,, now are  --limit rate of cooling
      (c@now - c@x) >= Iso_Properties::Cool_Rate*(now-x)>>
  invariant
    <<NORMAL() and EA_IS_1() and EA_IS_2()>>
  variables
    h :  Iso_Variables::on_off  -- heat control command value
    q :  Iso_Variables::Heat  -- heat source energy value
    c :  Iso_Variables::Heat  -- current Air heat energy value
    l :  Iso_Variables::LdtTemp  -- lower desired temperature value
    u :  Iso_Variables::UdtTemp  -- upper desired temperature value
  constants
    r = 73.0 f  -- constant room temperature
    k = 0.026 wpmk  -- average thermal conductivity of air
  behavior
    Heating ::= ’DT 1 c = -k*(c - q)’ & ’DT 1 q = 1’ [[> tout!(c) ]]> Continue
    Cooling ::= ’DT 1 c = -k*(c - q)’ & ’DT 1 q = -1’ [[> tout!(c) ]]> Continue
    AirTemp ::= hss?(h) & (h=on) -> Heating [] (h=off) -> Cooling
    Continue ::= skip
    WorkingIsolette ::= repeat(AirTemp)
**};
end Air.impl;
```

```
differential_expression ::=
differential
| differential { * differential }+
| differential { + differential }+
| differential - differential

differential ::=
numeric_literal
| variable_identifier [^ numeric_literal ]
| derivative_expression
| derivative_time
| ( differential_expression )

derivative_expression ::=
DE order_integer_literal
dependent_variable_identifier
independent_variable_identifier

derivative_time ::=
DT order_integer_literal variable_identifier
```

Boolean expressions are composed of Boolean terms combined with the binary **and**, **or**, and **xor** operators, and may be negated with the unary **not** operator. A term must either be a Boolean value, **true** or **false**, or an expression or relation that evaluates to a Boolean value. A relation is defined using numeric expressions combined with the standard relational operators =, <>, >, <=, >=, and >. The complete grammar for Boolean expressions is given below.

```
boolean_expression ::=
boolean_term
| boolean_term { and boolean_term }+
| boolean_term { or boolean_term }+
| boolean_term { xor boolean_term }+

boolean_term ::=
[ not ] ( true | false |
( boolean_expression ) | relation )

relation ::=
[ numeric_expression relation_symbol
numeric_expression ]

numeric_expression ::=
numeric_term | numeric_term - numeric_term
| numeric_term / numeric_term
| numeric_term mod numeric_term
| numeric_term { + numeric_term }+
| numeric_term { * numeric_term }+

numeric_term::=
[-] ( numeric_literal | variable_identifier
| numeric_expression )

numeric_literal ::=
integer_literal | real_literal

relation_symbol ::= = | <> | > | <= | >= | >
```

In our running example, the continuous behavior of the Air component has been captured as a repeating `AirTemp` process in the `WorkingIsolette` process of the `behavior` section in Listing 2. On each iteration, process `AirTemp` obtains the status of the Heat Source through its `hss?(h)` communication event, and chooses between `Heating` or `Cooling` processes based on the communicated value of variable `h`. If the Boolean expression `(h=on)` is true then the behavior is as specified by process Heating, otherwise as specified by process Cooling.

Continuous evolution of current temperature c, when the Heat Source is **on**, is specified by 'DT 1 c = -k*(c - q)' & 'DT 1 q = 1' with the ampersand acting as a separator having no semantics. As explained in Section 3, changes in current temperature c depend on the amount of heat being generated by the Heat Source. This physical behavior is modeled by the 'DT 1 q = 1' term, where q is the rate of change in Heat Source output. When the Heat Source is switched **off** by the Thermostat the rate of change in q becomes negative and the Air behavior is then governed by the Cooling process, 'DT 1 c = -k*(c - q)' & 'DT 1 q = -1'. The Heating and Cooling processes define the continuous evolution of c under different conditions. Either can be preempted by a communication interrupt delivered along the Air *out data* port, tout.

In addition to modeling the constructs described above, HA supports both sequential and concurrent composition. Sequential composition defines consecutively-executing behaviors. For example, a sequentially composed process P;Q behaves as P first and after its successful termination, behaves as Q. A parallel compose S1||S2 behaves as if S1 and S2 were running independently, except that all interactions occur through communication events. The grammar for sequential and concurrent compositions, and choice and repetition constructs is as follows.

```
SequentialComposition ::=
{ behavior_identifier
{ ; behavior_identifier }+ }

ConcurrentComposition ::=
{ behavior_identifier
{ || behavior_identifier }+ }

choice ::=
alternative { [] alternative }*

alternative ::=
( boolean_expression ) -> process_identifier

repetition ::=
repeat [ [ ( integer_literal
| integer_variable_identifier ) ] ]
( process_identifier )
```

Several primitive processes like skip to model successful execution termination; x:=e to model variable assignment; and wait to model time delay, can also be specified using HA. The grammar for these primitive processes is quite simple and is not specified here.

4.2 Cyber-Physical Interaction Modeling

A computing unit's extensive interactions with, and strong dependence on its physical environment makes precise specification of the system's cyber-physical interaction (communication between computing units and the physical environment) an essential part of hybrid system modeling.

Extensive support for interaction and continuous evolution preemption due to timed and communication interrupts is a major innovation of our proposed HA. Communication between physical processes uses the channels declared in the channels section of the respective behavior specifications, while communication with an AADL component relies on the ports that are declared in the component's type. Communication channels must be paired in complementary directions, e.g., an *out* channel with an *in* channel. The grammar for the channels section and communication events is defined below.

```
channel ::=
channel_identifier {, channel_identifier }*
:  data_component_classifier_reference

communication ::=
port_communication | channel_communication

port_communication ::=
port_identifier (?|!)
( [variable_identifier] )

channel_communication ::=
channel_identifier (?|!)
[ variable_identifier ]
```

Communication events hss?(h) and tout!(c) of Listing 2 enable cyber-physical interactions involving data ports of AADL components and variables of the physical process. An event can result from either a port input (?) or a port output (!) action. In this case, the Air component type (Listing 1) declares input port hss and output port tout, while quantities h and c are declared in the variables section of the component's HA specification (Listing 2).

Continuous process evolution may be terminated after a specific time or on a communication event. These are invoked through timed and communication interrupts, respectively. A timed interrupt preempts continuous evolution after a given amount of time whereupon the process then assumes the behavior specified by the interrupt. A communication interrupt preempts continuous evolution whenever communication takes places along any one of the named ports or channels. The grammar for interrupts follows.

```
interrupt ::=
timed_interrupt | communication_interrupt

timed_interrupt ::=
[> time_value ]> { behavior_identifier }+

time_value ::=
time_variable_identifier |
real_literal time_unit

communication_interrupt ::=
[[> port_or_channel_identifier
{ , port_or_channel_identifier }*
]]> { process_identifier }+
```

Listing 2 defines a communication interrupt that preempts the continuous evolution of the current temperature process quantity, c with [[> tout!(c)]]> Continue. The only port named is tout, and no channels named. Thus, whenever a value is sent out of the Air's tout port, the evolution of c will cease, and the Continue process will be adopted as the subsequent behavior for the process.

5. BEHAVIOR CONSTRAINTS

In addition to continuous-time differential equations, HA accommodates the use of BLESS as a behavior interface specification language (BISL). Then, BLESS Assertions may be used to express constraints on any HA defined behavior. Assertion are more fully explained in [17].

Two sections of an HA subclause permit the application of BLESS Assertion constraints to a component's continuous-time behavior: the `invariant` section may contain a single Assertion that always hold true for aspects of the component's behavior; and the `assert` section may declare Assertions either for later inclusion as terms in the `invariant` section, thereby making it more concise, or for expressing exceptional conditions.

The HA specification in Listing 2 uses Assertions in both the `assert` and `invariant` sections to constrain the continuous behavior of the Air in the Isolette. The intent is to keep the Air temperature within a normal range, `NORMAL`, and to limit the rates of heating, `EA_IS_1`, and cooling, `EA_IS_2`. Statements within an AADL property set, `Iso_Properties` (not shown), puts the tolerance for the temperature range at 0.5°F, and sets the Air heating and cooling rates to be 1°F per minute.

6. RELATED WORK

In order to enhance its extensive capabilities for system modeling and analysis, AADL supports extensions to its core language by way of properties and annexes. The Hybrid Annex presented in this paper, takes advantage of the latter to extend AADL with capabilities suitable for modeling the continuous behavior of physical environments. Major existing work related to the language extension (i.e., annex definition) of AADL consists of numerous dedicated annexes defined to fulfill specific modeling and analysis needs. Some of these annexes have already been standardized while others are currently undergoing the standardization process.

Standardized annexes include the Error Model Annex [12] for conducting safety and reliability analyses that specify the fault behavior of components and connections on identified paths, or *flows*, along with their propensity for error propagation; the Data Modeling Annex [14] to enable the creation of complicated data types in an architectural model; and the ARINC653 Annex [15] used to enforce standard ARINC653 compliant modeling and analysis. One of the important annexes presently undergoing standardization is the Requirements Definition and Analysis Annex [3] for the association of system requirements with elements of an AADL model.

The works most closely related to ours, in terms of providing AADL with behavior modeling support, are the Behavior Annex (BA) and BLESS [13, 18]. Both BA and BLESS use state transition systems with guards and actions to model the discrete behavior of control systems. To prove correct behavior, BLESS includes a means for the automatic generation of proof obligations, as well as an interactive theorem proving tool based on temporal logic formulas, specified with BLESS Assertions.

The modeling of cyber-physical systems with AADL presented in [25] is based on networks of timed automata with use of the UPPAAL model checker for property analysis. Another approach to modeling hybrid systems with AADL has been proposed by Qian [23], but it is not expressive enough to model constants with measuring units, and has difficulty modeling complex continuous behavior expressed with differential equations. In [2], Banerjee, et al. discuss the modeling of Body Area Networks (BAN) with AADL based on intentional and un-intentional interactions between human body and the BAN devices. These interactions are modeled using the concepts of the region of impact and the region of interest, with identified parameters.

Compared to the above mentioned related works, our proposed Hybrid Annex is more expressive in specifying the primitives of hybrid system models, e.g., variables with data types, constants with measuring units, and behavior with complex Boolean expressions. It also provides extensive support for cyber-physical interaction modeling through use of timed and communication interrupts—an essential element of hybrid system modeling not provided for to such an extent by related efforts. Exclusive support for behavior constraints and the definition of component invariants with BLESS Assertions is a novel feature of our Hybrid Annex.

7. CONCLUSION AND FUTURE WORK

To facilitate the continuous behavior modeling of the physical portion of a hybrid system, and to integrate this with discrete behavior modeling activities, we have proposed a Hybrid Annex for AADL. We have demonstrated its use by modeling the physical environment of the FAA's well-known isolette example, and illustrated how this continuous model interacts with a more conventional AADL model of the system's discrete components, thus opening the door to truly integrated modeling of cross-domain interactions within cyber-physical systems. The EBNF grammar of the proposed sublanguage was also provided.

The Hybrid Annex language extension provides a means for the AADL community to engage in the full modeling of hybrid systems: One which includes the behavior of critical environmental and continuous-time elements, like, for example, the process model required for proper functioning of a control system. And furthermore, to do so in an integrated manner where the artificial separation between discrete and continuous domains has been erased. Finally, we showed how the modeled behavior can be constrained through the use of BLESS Assertions and invariants in two sections of a Hybrid Annex subclause.

Being a first step towards continuous behavior and cyber-physical interaction modeling with AADL, this study has opened up new opportunities for research and development. An important future contribution will be the implementation of a Hybrid Annex plug-in for the Open-Source AADL Tool Environment (OSATE) modeler. A plug-in planned to verify the correctness of Hybrid Annex specifications will leverage ongoing work with an in-house Hybrid Hoare Logic (HHL) theorem prover [27]. Formalizing the semantics of AADL models augmented with Hybrid Annex specifications is another topic for immediate research [1].

Acknowledgements

The first and fourth authors are supported by the National Basic Research Program of China under Grant No. 2014CB340700, by Natural Science Foundation of China under Grant No. NSFC-91118007 and NSFC-6110006, and by the CAS/SAFEA International Partnership Program for Creative Research Teams, and the last author is supported by the National Infrastructure Software Plan under Grant

No.2012ZX01041-002-003. The second and third authors are supported in part by the US National Science Foundation (NSF) (#0932289, #1239543), and the NSF US Food and Drug Administration Scholar-in-Residence Program (#1065887, #1238431).

8. REFERENCES

[1] Ehsan Ahmad, Yunwei Dong, Shuling Wang, Naijun Zhan, and Liang Zou, *Adding formal meanings to aadl with hybrid annex*, accepted for publication, The 11th International Symposium on Formal Aspects of Component Software, FACS'14, 2014.

[2] Ayan Banerjee, Sailesh Kandula, Tridib Mukherjee, and Sandeep K. S. Gupta, *Band-aide: A tool for cyber-physical oriented analysis and design of body area networks and devices*, ACM Transactions on Embedded Computing Systems vol:11, no. S2, pp. 49:1–49:29, ACM, 2012.

[3] Dominique Blouin, Eric Senn, and Skander Turki, *Defining an annex language to the architecture analysis and design language for requirements engineering activities support*, Model-Driven Requirements Engineering Workshop (MoDRE), pp. 11–20, 2011.

[4] Denis Buzdalov, Alexey Khoroshilov, and Eugene Kornykhin, *Unit relations annex*, (draft, progress update) https://wiki.sei.cmu.edu/aadl/images/c/c5/201309-ispras-unit-relations-annex.pdf, 2013.

[5] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ives, Reed Little, Robert Nord, and Judith Stafford, *Documenting software architecture: Views and beyond*, SEI Series in Software Engineering, Pearson Education, Inc., Boston, MA, 2003.

[6] AADL Standard Committee, *Aadl user days website*, https://wiki.sei.cmu.edu/aadl/index.php/AADL_User_Days, 2014.

[7] Peter Feiler and David Gluch, *Model-based engineering with AADL: An introduction to the SAE architecture analysis & design language*, Addison-Wesley, 2012.

[8] Peter Feiler, Jörgen Hansson, Dionisio de Niz, and Lutz Wrage, *System architecture virtual integration: An industrial case study*, Tech. Report CMU/SEI-2009-TR-017, SEI, CMU, 2009.

[9] Jifeng He, *From CSP to hybrid systems*, A Classical Mind, Essays in Honour of C.A.R. Hoare, Prentice Hall International (UK) Ltd., pp. 171–189, 1994.

[10] Mats Heimdahl, Lian Duan, Anitha Murugesan, and Sanjai Rayadurgam, *Modeling and requirements on the physical side of cyber-physical systems*, Second International Wokshop on the Twin Peaks of Requirements and Architecture, ICSE'13, IEEE, 2013.

[11] Thomas A. Henzinger and Joseph Sifakis, *The embedded systems design challenge*, FM'06, LNCS, pp. 1–15, 2006.

[12] SAE International, *Architecture analysis & design language (AADL) annex volume 1: Annex e: Error model annex*, 2006.

[13] SAE International, *Architecture analysis & design language (AADL) annex volume 2: Annex d:behavior model annex*, 2011.

[14] SAE International, *Architecture analysis & design language (AADL) annex volume 2: Annex b:data modeling annex*, 2011.

[15] SAE International, *Architecture analysis & design language (AADL) annex volume 2: Annex f:arinc653 annex*, 2011.

[16] SAE International, *SAE as5506b, architecture analysis & design language (AADL)*, 2012.

[17] Brian R. Larson, *Behavior Language for Embedded Systems with Software: Language Reference Manual*, `info.santoslab.org/research/aadl/bless`, 2014.

[18] Brian R. Larson, Patrice Chalin, and John Hatcliff, *BLESS: Formal specification and verification of behaviors for embedded systems with software*, NASA Formal Methods, LNCS, vol. 7871, Springer Berlin Heidelberg, pp. 276–290, 2013.

[19] Brian R. Larson, John Hatcliff, Kim Fowler, and Julian Delange, *Illustrating the AADL error modeling annex (v.2) using a simple safety-critical medical device*, Proceedings of the 2013 ACM SIGAda Annual Conference on High Integrity Language Technology, HILT '13, ACM, pp. 65–84, 2013.

[20] Edward A. Lee, *What's ahead for embedded software?*, IEEE Computer, pp. 18–26, 2000.

[21] David L. Lempia and Steven P. Miller, *Requirement engineering management handbook*, Tech. Report DOT/FAA/AR-08/32, Federal Aviation Administration, 2009.

[22] Nancy Levenson, *Engineering a safer world*, MIT Press, Cambridge, MA, 2011.

[23] Qian Yuqing, Liu Jing, and Chen Xiaohong, *Hybrid aadl: A sublanguage extension to aadl*, Proceedings of the 5th Asia-Pacific Symposium on Internetware, Internetware '13, ACM, pp. 27:1–27:4, 2013.

[24] Naijun Zhan, Shuling Wang, and Hengjun Zhao, *Formal modelling, analysis and verification of hybrid systems*, Unifying Theories of Programming and Formal Engineering Methods, LNCS, pp. 207–281, 2013.

[25] Yu Zhang, Yunwei Dong, Fan Zhang, and Yunfeng Zhang, *Research on modeling and analysis of cps*, Proceedings of the 8th International Conference on Autonomic and Trusted Computing (Berlin, Heidelberg), ATC'11, Springer-Verlag, pp. 92–105, 2011.

[26] Chaochen Zhou, Ji Wang, and Anders P. Ravn, *A formal description of hybrid systems*, Hybrid systems, LNCS, vol. 1066, pp. 511–530, 1996.

[27] Liang Zou, Jidong Lv, Shuling Wang, Naijun Zhan, Tao Tang, Lei Yuan, and Yu Liu, *Verifying chinese train control system under a combined scenario by theorem proving*, VSTTE, LNCS, vol. 8164, pp. 262–280, 2013.