Object-Oriented Formal Modeling and Analysis of Interacting Hybrid Systems in HI-Maude

Muhammad Fadlisyah[1], Peter Csaba Ölveczky[1], and Erika Ábrahám[2]

[1] Department of Informatics, University of Oslo, Norway
[2] Computer Science Department, RWTH Aachen University, Germany

Abstract. This paper introduces the HI-Maude tool that supports the formal modeling, simulation, and model checking of *interacting hybrid systems* in rewriting logic. Interacting hybrid systems exhibit both discrete and continuous behaviors, and are composed of components that influence each other's continuous dynamics. HI-Maude supports the *compositional* modeling of such systems, where the user only needs to describe the dynamics of *single* components and interactions, instead of having to explicitly define the continuous dynamics of the entire system. HI-Maude provides an intuitive, expressive, object-oriented, and algebraic modeling language, as well as simulation and LTL model checking with reasonably precise approximations of continuous behaviors for interacting hybrid systems. We introduce the tool and its formal analysis features, define its formal semantics in Real-Time Maude, and exemplify its use on the human thermoregulatory system.

1 Introduction

Modelers typically have to choose between *simulation tools* that provide intuitive and expressive modeling languages but only support system simulation, and *model checking* tools that can do powerful formal analyses but only provide quite restrictive modeling formalisms to ensure that key properties are decidable. For *real-time systems*, the rewriting-logic-based *Real-Time Maude* tool [17, 18] tries to bridge this gap by providing an intuitive and expressive object-oriented modeling formalism as well as both simulation, reachability, and LTL and TCTL model checking. Although the price of this expressiveness is that properties are in general undecidable, useful formal analyses can often be performed on very complex systems; furthermore, Real-Time Maude model checking is sound and complete model checking for many systems encountered in practice [16]. Real-Time Maude has proved to be useful for analyzing a wide range of advanced applications that are beyond the scope of timed automata, including large communication protocols [19, 14], wireless sensor network algorithms [12, 20], and scheduling algorithms that need unbounded queues [15].

This paper introduces the *HI-Maude* tool that extends Real-Time Maude to support the modeling, simulation, and model checking of *hybrid systems* with combined discrete and continuous behaviors. We target complex hybrid systems in which multiple physical entities interact and influence each other's continuous behavior. For example, a hot cup of coffee in a room interacts with the room through different kinds of heat transfer, leading to a decrease of the coffee's temperature and to a slight increase of the room's temperature. One distinguishing feature of HI-Maude is the *modularity* and *compositionality* of the specification of the system's continuous dynamics. Non-compositional specification of the *whole* system is very hard, as it involves combining the ordinary differential equations (ODEs) that specify the dynamics of its components; it also requires redefining the system's continuous dynamics for each new configuration of interacting physical components. To achieve the desired modularity and compositionality, HI-Maude offers an object-oriented modeling methodology [5], based on the *effort/flow approach* [24], that allows us to specify the continuous dynamics of *single physical entities* (such as the cup of coffee and the room) and of *single physical interactions* (such as thermal conduction and convection). To analyze the system, HI-Maude uses adaptations of different numerical methods (the Euler method and the Runge-Kutta method of different orders) to give fairly precise approximate solutions to coupled ordinary differential equations.

Several simulation tools for hybrid systems, such as MATLAB/Simulink [23], HyVisual [13], and CHARON [4], are based on numerical methods. In contrast to these tools, HI-Maude also supports reachability analysis and temporal logic model checking. Of course, the results of such model checking must be seen in light of the approximation inaccuracies of the continuous behaviors. Our approach also differs from model checkers for hybrid systems, such as CheckMate [1], PHAVer [8], d/dt [3], and HYPERTECH [10], in that we do not use abstraction or over-approximation. Whereas other formal tools use hybrid automata, chart or block models, or formulas for modeling, a main advantage of HI-Maude is that it is based on the intuitive yet expressive *rewriting logic* formalism as the underlying modeling formalism.

To summarize, HI-Maude provides:

1. a modeling framework for hybrid systems that is (i) intuitive and expressive, (ii) object-oriented, with support for advanced features such as inheritance and dynamic creation and deletion of objects, (iii) algebraic, and that (iv) makes it easy to specify the continuous dynamics in a simple and compositional way;
2. a simulation, reachability analysis, and LTL model checking tool for such models based on fairly precise approximations of the system's continuous behavior.

We exemplify the use of HI-Maude with a small example of the coffee in the room, as well as with the formal modeling and analysis of the human thermoregulatory system. Both these systems are outside the decidable fragment of hybrid automata, in part due to their complex non-linear continuous dynamics.

The paper is structured as follows: Section 2 gives an overview of Real-Time Maude. Section 3 briefly explains our effort/flow-based method proposed in [5] for modeling hybrid systems in rewriting logic. Section 5 describes the HI-Maude tool, Section 6 briefly outlines its semantics, and Section 7 gives an overview of the modeling and analysis of the human thermoregulatory system in HI-Maude. Finally, some concluding remarks are given in Section 8.

The HI-Maude tool, together with some examples and a longer technical report, is available at `http://folk.uio.no/mohamf/HI-Maude`.

2 Real-Time Maude

A Real-Time Maude [17] *timed module* specifies a *real-time rewrite theory* of the form (Σ, E, IR, TR), where:

– (Σ, E) is a *membership equational logic* [2] theory with Σ a signature[3] and E a set of *confluent and terminating conditional equations.* (Σ, E) specifies the system's state space as an algebraic data type, and must contain a specification of a sort `Time` modeling the (discrete or dense) time domain.
– *IR* is a set of (possibly conditional) *labeled instantaneous rewrite rules* specifying the system's *instantaneous* (i.e., zero-time) local transitions, written with syntax `rl [`l`] : `t` => `t', where l is a *label.* Such a rule specifies a *one-step transition* from an instance of t to the corresponding instance of t'. The rules are applied *modulo* the equations E.[4]
– *TR* is a set of (usually conditional) *tick rewrite rules*, written with syntax `crl [`l`] : {`u`} => {`v`} in time `τ` if `*cond*, that model time elapse. `{_}` encloses the global state, and τ is a term that denotes the *duration* of the rewrite.

The Real-Time Maude syntax is fairly intuitive. For example, a function symbol f is declared with the syntax `op `f` : `$s_1 \ldots s_n$` -> `s, where $s_1 \ldots s_n$ are the sorts of its arguments, and s is its (value) *sort.* Equations are written with syntax `eq` t `=` t', and `ceq` t `=` t' `if` *cond* for conditional equations. The mathematical variables in such statements are declared with the keywords `var` and `vars`. An equation $f(t_1, \ldots, t_n) = t$ with the `owise` (for "otherwise") attribute can be applied to a subterm $f(\ldots)$ only if no other equation with left-hand side $f(u_1, \ldots, u_n)$ can be applied. We refer to [2] for more details on the syntax of Real-Time Maude.

In *object-oriented* Real-Time Maude modules, a *class* declaration

`class `C` | `att_1` : `s_1`, ... , `att_n` : `s_n` .`

declares a class C with attributes att_1 to att_n of sorts s_1 to s_n. An *object* of class C in a state is represented as a term `<` $O : C \mid att_1 : val_1, ..., att_n : val_n$ `>` of sort `Object`, where O, of sort `Oid`, is the object's *identifier*, and where val_1 to val_n are the current values of the attributes att_1 to att_n. In a *concurrent* object-oriented system, the state is a term of sort `Configuration`. It has the structure of a *multiset* made up of objects and messages. Multiset union for configurations is denoted by a juxtaposition operator (empty syntax) that is declared associative and commutative, so that rewriting is *multiset rewriting* supported directly in Real-Time Maude.

The dynamic behavior of concurrent object systems is axiomatized by specifying each of its transition patterns by a rewrite rule. For example, the rule

```
rl [l] :  < O : C | a1 : 0, a2 : y, a3 : w, a4 : z >  =>
          < O : C | a1 : T, a2 : y, a3 : y + w, a4 : z >
```

defines a parametrized family of transitions which can be applied whenever the attribute `a1` of an object `O` of class `C` has the value `0`, with the effect of altering the attributes `a1` and `a3` of the object. "Irrelevant" attributes (such as `a4`, and the *right-hand side occurrence* of `a2`) need not be mentioned in a rule (or equation).

A *subclass* inherits all the attributes and rules of its superclasses.

[3] i.e., Σ is a set of declarations of *sorts*, *subsorts*, and *function symbols*
[4] E is a union $E' \cup A$, where A is a set of equational axioms such as associativity, commutativity, and identity, so that deduction is performed *modulo* A. Operationally, a term is reduced to its E'-normal form modulo A before any rewrite rule is applied.

Formal Analysis. A Real-Time Maude specification is *executable*, and the tool offers a variety of formal analysis methods. The *rewrite* command simulates *one* fair behavior of the system *up to a certain duration.* The *search* command uses a breadth-first strategy to analyze all possible behaviors of the system, by checking whether a state matching a *pattern* and satisfying a *condition* can be reached from the initial state. The timed search command which searches for n such states reachable within time τ has syntax `(tsearch [`*n*`]` *t* `=>*` *pattern* `such that` *cond* `in time <=` τ `.)`.

Real-Time Maude also extends Maude's *linear temporal logic model checker* to check whether each behavior, possibly up to a certain time bound, satisfies a temporal logic formula. *State propositions*, possibly parametrized, can be predicates characterizing properties of the state and/or properties of the global time of the system. A temporal logic *formula* is constructed by state propositions and temporal logic operators such as `True`, `False`, `~` (negation), `/\`, `\/`, `->` (implication), `[]` ("always"), `<>` ("eventually"), `U` ("until"), and `W` ("weak until").

3 Effort/Flow Modeling of Interacting Hybrid Systems

Our tool is based on the modeling methodology developed in [5], which adapts the *effort/flow* method [24] to model a physical system as a network of *physical entities* and *physical interactions* between the entities.[5] This makes the models *modular* and *compositional*, in the sense that it is sufficient to define the continuous dynamics for each (class of) component(s) to define the dynamics of the entire system.

As shown in Fig. 1, a *physical entity* is described by a real-valued *effort*, a set of *attribute* values, and the entity's *continuous dynamics.* The attribute values describe discrete properties, e.g., the mass or the phase of a material, that can only be changed by discrete events. The effort variable represents a dynamic physical quantity, such as temperature, that evolves over time as given by the continuous dynamics in the form of an ordinary differential equation (ODE).

A *physical interaction* between two physical entities is described by a real-valued *flow*, a set of *attribute* values, and a *continuous dynamics.* The flow value describes the dynamic interaction between two entities, whose evolution over time is specified by the continuous dynamics. The values of the effort variables of the two physical entities are used in the definition of the continuous dynamics of the interaction. In an *open physical interaction* we abstract away from one of the entities and model a constant flow to a single entity.

Finally, the system may also exhibit discrete transitions, because of phase changes, explicit control, communication, or other factors.

Figure 1 illustrates our modeling methodology on a *thermal* system consisting of a cup of coffee in a room. Heat flows from a hot cup of coffee to the room through both *heat convection* and *heat conduction*, where the flow variable ($\dot{Q}$) denotes the heat flow rate of the physical interaction. The effort variables of the two physical entities (coffee and room) are the temperature T_r of the room and the temperature

[5] The approach using effort/flow variables is applicable to different areas of physical systems. In mechanical translation systems, the pair of effort and flow variables are force and velocity; in mechanical rotation systems, torque and angular velocity; in electrical systems, voltage and current; in fluidic systems, pressure and volume flow rate; in thermal systems, temperature and heat flow rate.

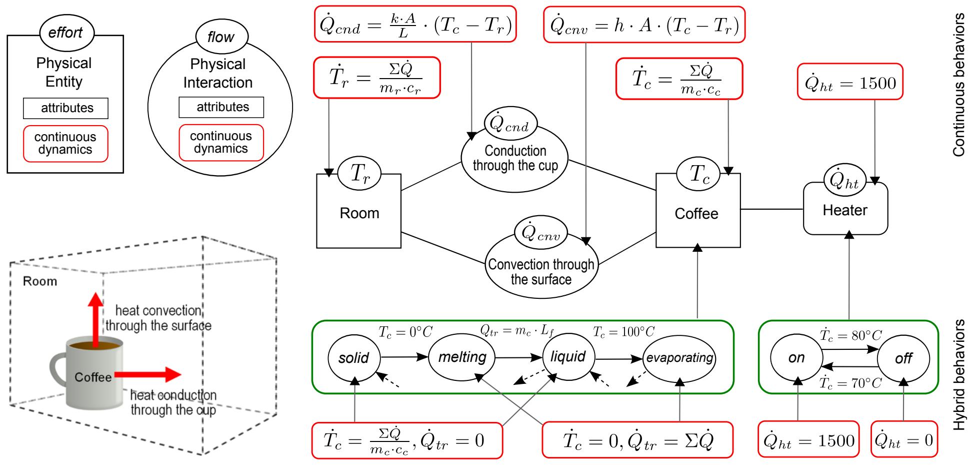
effort
Physical Entity
attributes
continuous dynamics
flow
Physical Interaction
attributes
continuous dynamics
$\dot{Q}_{cnd} = \frac{k \cdot A}{L} \cdot (T_c - T_r)$
$\dot{Q}_{cnv} = h \cdot A \cdot (T_c - T_r)$
$\dot{T}_r = \frac{\Sigma \dot{Q}}{m_r \cdot c_r}$
$\dot{T}_c = \frac{\Sigma \dot{Q}}{m_c \cdot c_c}$
$\dot{Q}_{ht} = 1500$
$\dot{Q}_{cnd}$
Conduction through the cup
T_r
Room
T_c
Coffee
$\dot{Q}_{ht}$
Heater
$\dot{Q}_{cnv}$
Convection through the surface
Continuous behaviors
Room
heat convection through the surface
Coffee
heat conduction through the cup
$T_c = 0°C$
$Q_{tr} = m_c \cdot L_f$
$T_c = 100°C$
solid
melting
liquid
evaporating
$\dot{T}_c = 80°C$
$\dot{T}_c = 70°C$
on
off
Hybrid behaviors
$\dot{T}_c = \frac{\Sigma \dot{Q}}{m_c \cdot c_c}, \dot{Q}_{tr} = 0$
$\dot{T}_c = 0, \dot{Q}_{tr} = \Sigma \dot{Q}$
$\dot{Q}_{ht} = 1500$
$\dot{Q}_{ht} = 0$

Fig. 1. Physical system components and their interaction in a simple thermal system.

T_c of the coffee. The values of the other attributes are parameters such as the mass and surface area of the cup. We also have a (flow-source) heater that adds a constant flow of heat to the cup of coffee. Finally, we have discrete behaviors, since the phase of the coffee could change instantaneously from, e.g., liquid to evaporating, and since the heater could be automatically turned off and on to keep the temperature of the coffee between 70 and 80 degrees.

4 Executing Interacting Hybrid Systems

The continuous dynamics of a physical entity is typically defined as an ordinary differential equation (ODE), where the time derivative of its effort is a function of both the entity's attribute values *and* the flows of connected interactions. Dually, the continuous dynamics of a physical interaction is an equation with the flow variable on the left-hand side and an expression possibly referring to the interaction's local attributes and the efforts of the connected entities on the right-hand side. This way the direct coupling of the ODEs of physical entities can be avoided.

The continuous behavior of a system is approximated by a sequence of discrete time steps. For each time step, first the current values of the flow variables are computed. These values are then used to compute the values of the efforts of the physical entities after the time step.

As we do not require *linear* ODEs, the continuous dynamics of a system is in general not analytically solvable. We therefore use numerical techniques to *approximate* the continuous behaviors by advancing time in small discrete time increments, and approximating the values of the continuous variables at each "visited" point in time. We have adapted the following numerical methods to our effort/flow framework: the *Euler* [5], the *Runge-Kutta 2nd order* (RK2), and the *Runge-Kutta 4th order* (RK4) methods [7] for fixed time increments. The papers [5, 7] explain in detail how we have adapted these numerical methods, and how they have been defined in Real-Time Maude. Furthermore, those papers also show the execution times and the relative errors for the different numerical methods on an example that does have an analytical solution. As expected, the Euler method is the fastest but least accurate method; while RK4 is the slowest but most accurate fixed-step method.

5 The HI-Maude Tool

The HI-Maude tool integrates the modeling techniques in Section 3 and the Real-Time Maude implementations of the numerical approximation algorithms Euler, RK2, and RK4 to support the rewriting-logic-based object-oriented formal modeling and simulation, reachability, and LTL model checking analysis of hybrid systems containing interacting physical components.

In particular, the HI-Maude tool makes it easy for the user to define the continuous dynamics of the effort and flow variables of single physical entities and physical interactions, respectively. These values *can* conveniently be defined as ODEs. Once the dynamics of the single physical components have been defined, the tool

1. automatically defines the continuous dynamics of the entire systems, and
2. provides the usual Real-Time Maude formal analysis commands, but where the desired built-in approximation algorithm and the desired time increments used by the approximations are additional parameters of the commands.

Furthermore, the tool provides infrastructure to define that instantaneous transitions (modeled as instantaneous rewrite rules) are applied in a timely manner.

HI-Maude is implemented in Maude as an extension of the Real-Time Maude tool, and is available at `http://folk.uio.no/mohamf/HI-Maude/`.

5.1 Representing Continuous Values

Maude provides a built-in data type for the *unbounded* rational numbers, and we first used these rationals for the effort and flow values. However, it quickly became apparent that it is inconvenient to use the rationals, for the following reasons:

– It is hard to read large rational numbers.
– The *size* of the rational numbers gets very large (both the numerator and the denominator are large numbers), which significantly slows down the execution.

HI-Maude now therefore uses the Maude's built-in IEEE-754 floating-point numbers to represent the values of the effort and flow variables.

5.2 Modeling

This section explains how hybrid systems can be specified in HI-Maude as a multiset of objects representing physical entities, physical interactions, other flow components, as well as other objects, representing, e.g., controllers, sensors, and actuators. We first show how the physical entities and interactions and their continuous dynamics should be defined, and then discuss instantaneous discrete transitions.

Modeling Physical Entities and Interactions. HI-Maude provides the following built-in classes for specifying physical entities and interactions, as well as flow-source components. Concrete physical entities (interactions, . . .) must then be defined as object instances of user-defined subclasses of these built-in classes:

```
class PhysicalEntity | effort : Float .
class PhysicalInteraction | flow : Float, entity1 : Oid, entity2 : Oid .
class FlowSource | flow : Float, entity : Oid .
```

The `effort` attribute denotes the effort value of the entity. For the `Physical-Interaction` class, the `flow` attribute denotes the flow between the physical entities given by the `entity1` and the `entity2` attribute values. In case of `FlowSource`, the `entity` attribute denotes the name of physical entity which receives the given flow.

For example, if we want to define *thermal* systems such as the simple coffee example in Fig. 1, we can define a new class `ThermalEntity` (with attributes denoting the heat capacity and the mass of the entity), whose objects model thermal physical entities, such the cup of coffee and the room, as a subclass of `PhysicalEntity`:

```
class ThermalEntity | heatCap : Float, mass : Float .
subclass ThermalEntity < PhysicalEntity .
```

The `effort` attribute of the superclass denotes the temperature of the entity. Likewise, as shown in Fig. 1, the heat flow by *convection* through the surface of the coffee is characterized by the temperatures of the entities as well as of the *area* of the surface (A) and the convection coefficient h:

```
class Convection | area : Float, convCoeff : Float .
subclass Convection < PhysicalInteraction .
```

Thermal *conduction* and *radiation* can be defined similarly (see [5]). Finally, we can define a heater as a constant heat flow source (where this flow is 0 when the `status` of the heater is `off` and is 1500 when the `status` is `on`; we also monitor the temperature of the coffee, but abstract from the details of that sensing):

```
class Heater | status : OnOff, monitoredTemp : Float .
subclass Heater < FlowSource .
```

An initial state (in which the system has not yet computed the first values of the flows) of the coffee system could then be

```
{< coffee : ThermalEntity | effort : 70.0, heatCap : 4.2, mass : 0.4 >
 < room : ThermalEntity | effort : 20.0,  heatCap : 10.5, mass : 80.0 >
 < c-rCond : Conduction | flow : 0.0, entity1 : coffee, entity2 : room, ... >
 < c-rConv : Convection | flow : 0.0, entity1 : coffee, entity2 : room,
                          convCoeff : 0.02, area : 0.05 >
 < heater : Heater | flow : 1500.0, entity : coffee, status : on, monitoredTemp : 70.0 >}
```

Modeling the Continuous Dynamics. HI-Maude provides infrastructure that only requires the user to define the continuous dynamics of single physical components. As indicated in Fig. 1, the time derivative of the *effort* of an entity is a function of *the sum of the flows* to/from the entity as well as of other attribute values of the entity (e.g., $\dot{T}_r = \frac{\sum \dot{Q}}{m_r \cdot c_r}$). Likewise, the *flow* of an interaction is a function of the (previous) values of the efforts of the connecting entities and other attribute values of the interaction (e.g., $\dot{Q}_{cnv} = h \cdot A \cdot (T_c - T_r)$). The flows of the `FlowSource` components may depend on attribute (and effort) values of both the component and the connecting entity/entities (although in the coffee example this flow is simply defined as $\dot{Q}_{ht} =$ if `status` == `on` then 1500.0 else 0.0).

For each such physical component, the user must define the corresponding of the following functions:

```
op effortDyn : Float Object -> Float .
op flowDyn : Float Float Object -> Float .
op flowSourceDyn : Configuration -> Float .
```

The first argument of `effortDyn` is the sum of the values of the flows to/from the entity, and other argument is the entity object itself. `effortDyn(`$\sum \dot{Q}$, *entity*`)` therefore defines the time derivative of the effort variable of the *entity* object. In our coffee example, this is defined in the same way for both the coffee and the room ($\dot{T}_r = \frac{\sum \dot{Q}}{m_r \cdot c_r}$ and $\dot{T}_c = \frac{\sum \dot{Q}}{m_c \cdot c_c}$, for the different $\sum \dot{Q}$s)) and hence the user must define `effortDyn` as follows:

```
eq effortDyn(X, < O : ThermalEntity | mass : M, heatCap : C >) = X / (M * C) .
```

We follow here the convention that variables are written with (only) capital letters, and do not show the variable declarations.

To define the flow of an interaction, we must define `flowDyn(E1,E2,`*interaction*`)`, where `E`*i* denotes the (previous) value of the effort of the object referred to by the `entity`*i* attribute of the *interaction* object. For example, to define the heat flow through *convection* between the room and the coffee, a HI-Maude user defines

```
eq flowDyn(E1, E2, < O : Convection | convCoeff : CC, area : A >) = CC * A * (E1 - E2) .
```

The function `flowSourceDyn` defines the flow of the flow-source components, and its argument is the entire multiset of objects in the system. In our coffee example, the heat flow from the heater only depends on the state of the heater:

```
eq flowSourceDyn(< O : Heater | status : S >  REST) = if S == on then 1500.0 else 0.0 fi .
```

Discrete Transitions. Discrete transitions are modeled as instantaneous rewrite rules. In general, a rule may be applied whenever it is enabled, but nothing forces a rule to be taken in a timely manner. For example, we want a discrete transition that turns off the heater when it perceives that the temperature of the coffee has reached 80 degrees. This can be modeled with the following conditional rewrite rule:

```
crl [turnOffHeater] :
    < H : Heater | status : on, monitoredTemp : T >
   =>
    < H : Heater | status : off > if T >= 80.0 .
```

To force the application of this rule as soon as the temperature has reached 80 degrees, the user can define the function `timeCanAdvance` on heater objects. When `timeCanAdvance` of an object is `false`, time advance in a system stops, forcing the application of a suitable instantaneous rule. In the coffee example, the only discrete transitions that must be performed in a timely manner are those turning on and off the heater as soon as they are enabled. Therefore, the user can achieve this by only letting time advance when the heater can stay in a given state:

```
eq timeCanAdvance(< H : Heater | status : S, monitoredTemp : T >)
 = if S == on then T < 80.0 else T > 70.0 fi .
```

5.3 Formal Analysis in HI-Maude

HI-Maude extends Real-Time Maude's analysis commands by allowing the user to select (i) the numerical approximation technique used to approximate the continuous behaviors of the system, and (ii) the time increment used in the approximation.

For example, HI-Maude's hybrid *rewrite* command is used to simulate one behavior of system from a given initial state up to a certain duration and, possibly, up to a certain number of rewrites. The syntax of the hybrid rewrite command is:

```
(hrew [[n]] initState in time ∼ timeLimit using numMethod stepsize stepSize .)
```

where the '`[`n`]`' part is optional. The number n denotes the upper bound on the number of rewrite steps to perform; *initState* is the initial state; $\sim$ is either '`<=`' or '`<`'; *timeLimit* denotes an upper bound on the total duration of the rewrite sequence; *numMethod* $\in$ {`Euler`, `RK2`, `RK4`} is the numerical method used to approximate the continuous behaviors; and *stepSize* is the size of the time increment used in the approximation of the continuous behaviors.

Real-Time Maude's timed search command—which searches for states that are matched by a search pattern with a substitution that satisfies an (optional) condition and that can be reached from an initial state within a given time interval—has been extended to the hybrid setting in the same way:

```
(hsearch [[n]] initState =>* searchPattern [such that cond] in time ∼ timeLimit
                using numMethod stepsize stepSize .)
```

where $\sim \in$ {`<`, `<=`, `>`, `>=`}, and *cond* is a condition on the variables appearing in the search pattern. The arrow '`=>!`' is used to search for states which cannot be further rewritten. We can also search without time bounds by writing '`with no time limit`' instead of '`in time` $\sim$ *timeLimit*'.

The following command finds the shortest time needed to reach a state:

```
(hfind earliest init =>* pattern [such that cond] using numMethod stepsize stepSize .)
```

Finally, HI-Maude's model checker extends Real-Time Maude's explicit-state time-bounded linear temporal logic model checker in the same way. The time-bounded hybrid model checking command is written with syntax

```
(hmc initState |=t formula in time ∼ timeLimit using numMethod stepsize stepSize .)
```

where *formula* is an LTL formula and $\sim$ is either '`<`' or '`<=`'. The model checker return '`true`' if the property holds, and returns a counterexample otherwise.

5.4 Soundness and Completeness of HI-Maude Analyses

There is a trade-off between expressiveness and analytic power for hybrid systems. On the one hand, model checkers and reachability analysis tools only deal with very restricted fragments of hybrid systems, such as initialized rectangular hybrid automata (see [9] for an early survey on decidable fragments of hybrid automata), to ensure that key properties are decidable. On the other hand, simulation tools have much more expressive modeling languages, but only provide simulation capabilities.

HI-Maude is as expressive as simulation tools, yet provides reachability and LTL model checking analysis in addition to simulation. The price to pay is that reachability and satisfaction of LTL properties are in general no longer decidable.

HI-Maude only analyzes those behaviors that are possible with the given strategy for executing hybrid behaviors; namely, those behaviors possible when time is always increased with the given time increment, and where the given numerical method is used to approximate the continuous behaviors. Therefore, the results of search and model checking in HI-Maude may not be correct. If a counterexample is found in LTL model checking, or a desired state is found in a search, these are indeed valid counterexamples *up to the approximation errors due to the use of numerical approximations and round-off errors due to the use of floating-point numbers.* However, since only a *subset* of all possible behaviors are analyzed, the fact that a state is not found during a search or that LTL model checking returns `true` does not necessarily imply that the state cannot be reached or that the LTL property holds.

6 The Real-Time Maude Semantics of HI-Maude

This section gives a brief overview of the Real-Time Maude semantics of HI-Maude.

A HI-Maude *hybrid module* automatically imports a library of numerical approximations methods, the built-in classes described above, other functions, etc. For any HI-Maude command, the tool adds a *system manager* object

```
  < sysMan : SysMan | numMethod : numerical method, stepSize : step size >
```

to the initial state and then transforms a HI-Maude analysis command to the corresponding Real-Time Maude command. For example, the hybrid search command

```
(hsearch {init} =>* pattern in time <= 100 using rk4 stepsize 2 .)
```

is executed by executing the Real-Time Maude timed search command

```
(tsearch {init < sysMan : SysMan | numMethod : rk4, stepSize : 2 >} =>* pattern in time <= 100 .)
```

Furthermore, to any hybrid module, the following tick rule is added. This rule advances time by the time increment given in the HI-Maude command, and updates the continuous effort and flow variables according to the numerical method used:

```
crl [tick] :
    {< SM : SysMan | stepSize : SS > REST}
   =>
    {computeEF(< SM : SysMan | > REST)}  in time SS
   if timeCanAdvance(< SM : SysMan | > REST) .
```

The function `timeCanAdvance` is used, as mentioned, to allow the user to specify that time cannot advance in certain states, to ensure timeliness of discrete transitions. It is defined as follows: it distributes over each object in the state, and the '`owise`' equation ensures that time advance is not impeded by those objects for which the user has *not* defined a `timeCanAdvance`-equation:

```
vars NEC NEC1 NEC2 : NEConfiguration .
op timeCanAdvance : Configuration -> Bool [frozen (1)] .
eq timeCanAdvance(NEC1 NEC2) = timeCanAdvance(NEC1) and timeCanAdvance(NEC2) .
eq timeCanAdvance(NEC) = true [owise] .
```

The main function is the function `computeEF`, that computes the new values of the effort and flow values in the specification. We refer to the executable specification of HI-Maude for its precise definition, and to [5, 7] for an explanation of how to implement the numerical approximation algorithms in Real-Time Maude.

7 Case Study: The Human Thermoregulatory System

In addition to the coffee-in-the-room system, we have also used HI-Maude and its effort/flow-based modeling methodology on a more ambitious case study modeling and analyzing the human thermoregulatory system. Since the model is fairly large, we can only present a brief overview of our modeling and analysis efforts in this paper, and refer to our longer technical report (and the formal specification) at `http://folk.uio.no/mohamf/HI-Maude/` for a thorough exposition of this case study.

7.1 The Human Thermoregulatory System

Human thermoregulation is a complex mechanism regulating heat production within the body and regulating heat exchange between the body and the environment in order to maintain an internal body temperature of around $37°C$. Heat is produced within the body by the metabolism process, while the interaction with the environment causes heat loss or gain through physical processes such as radiation, evaporation, and convection [11]. *Hyperthermia* occurs when the body is unable to maintain a normal temperature, which increases significantly above normal. *Hypothermia* occurs when the body temperature decreases significantly below normal [22].

The thermoregulatory system is controlled by the *hypothalamus*, which enables mechanisms to support heat loss from the body when the body temperature is increasing above normal levels. These mechanisms include: increasing the diameter of blood vessels to let more blood flow underneath the skin (*vasodilation*), which promotes heat loss by radiation, convection, and conduction; and increasing sweat production, which promotes heat loss by evaporation. When the body temperature is decreasing, the hypothalamus enables the following mechanisms to reduce heat loss and increase heat production: decreasing the diameter of blood vessels to let less blood flows underneath the skin (*vasoconstriction*), and stimulating the skeletal muscles to cause shivering, which increases heat production by the body [21].

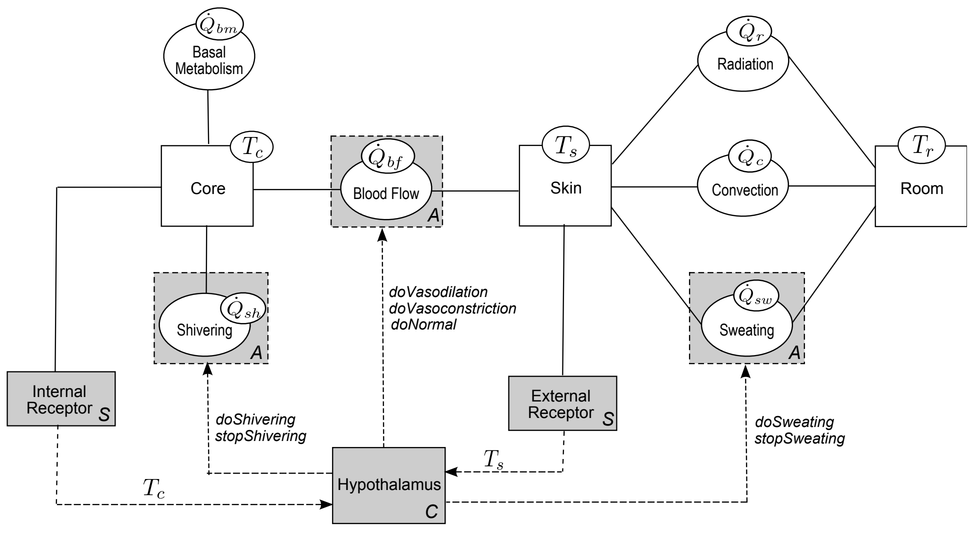

Fig. 2. Modeling human thermoregulatory using effort/flow technique.

7.2 Effort/Flow Modeling of the Human Thermoregulatory System

To reason about the thermoregulatory system, we can think of a person in a room as a *thermal* system, where the body core, the body skin, and the room are *thermal entities*, and where the heat flow between the body core and the skin and between the skin and the room are *thermal interactions*, as shown in Fig. 2. Heat flows between the body core and the skin through blood vessels, and between the skin and the room through radiation and convection. The *effort* variable of a thermal entity denotes its *temperature*, and the *flow* variable of the thermal interaction is the *heat flow rate*. The heat production inside the body through basal metabolism and shivering are represented as *flow-source* components. The heat loss from the skin by sweating is represented as a physical interaction, since the consequence of this process is heat gain in the room.

The controlling aspect of the thermoregulatory system is modeled by defining the hypothalamus as a *controller* component. Two *sensor* components are used to model the internal receptor and the external receptor for the core and the skin, respectively. Since increasing or decreasing blood flow, shivering, and sweating are the processes being controlled by the hypothalamus, the components representing these three controlling actions can be seen as *actuator* components.

Modeling the Human Thermal Entity Components. Each thermal entity is defined as in the coffee example by extending the built-in class `PhysicalEntity` with the entity's heat capacity and mass:

```
class ThermalEntity | mass : Float, heatCap : Float .
subclass ThermalEntity < PhysicalEntity .
```

The body core, the skin, and the room share the same thermal dynamics, where the temperature change ΔT is derived from $\Delta Q = m \cdot c \cdot \Delta T$, where ΔQ is the amount of heat transferred per time unit. We therefore define `effortDyn`, specifying the dynamics of an entity's effort value, for thermal entities as in the coffee example:

```
eq effortDyn(SF, < TE : ThermalEntity | mass : MASS, heatCap : HC >) = SF / (MASS * HC) .
```

where `SF` is the sum of the heat flows of the entity, which is computed by the tool.

To model the temperature-related states of the body core (*normal*, *moderate* and *severe hyperthermia*, *moderate* and *severe hypothermia*, and *dead*), we define a new subclass of `ThermalEntity` with a new attribute `coreState`:

```
sort CoreStateType .
ops normal modHyperthermia sevHyperthermia modHypothermia sevHypothermia
    dead : -> CoreStateType [ctor] .
class CoreHumanBody | coreState : CoreStateType .
subclass CoreHumanBody < ThermalEntity .
```

For example, the following rewrite rule models the the body core state change from *normal* to *moderate hyperthermia* when the temperature exceeds $38°C$:

```
crl [normal-to-modhyperthermia] :
    < CORE : CoreHumanBody | effort : TEMP, coreState : normal >
   =>
    < CORE : CoreHumanBody | coreState : modHyperthermia >  if TEMP > 38.0 .
```

The core state changes from *moderate hyperthermia* to *severe hyperthermia* if the core temperature exceeds $41°C$. Severe hyperthermia causes the sweating to stop, which is modeled by changing the state of the sweating component to *off*:

```
crl [modhyperthermia-to-sevhyperthermia] :
    < CORE : CoreHumanBody | effort : TEMP, coreState : modHyperthermia >
    < BFCS : BloodFlowCoreSkin | entity1 : CORE, entity2 : SKIN >
    < SKIN : SkinHumanBody | >
    < SWEAT : Sweating | entity1 : SKIN >
   =>
    < CORE : CoreHumanBody | coreState : sevHyperthermia >
    < BFCS : BloodFlowCoreSkin | >
    < SKIN : SkinHumanBody | >
    < SWEAT : Sweating | status : off >    if TEMP > 41.0 .
```

To ensure that the above rules (and all the other instantaneous rules) are applied as soon as they are enabled, we use the built-in `timeCanAdvance` function to define, for each core state, when time can advance *without* a rule having to be taken, e.g.:

```
eq timeCanAdvance(< CORE : CoreHumanBody | effort : TEMP, coreState : normal >)
 = TEMP > moderateHypothermiaPoint and TEMP <= moderateHyperthermiaPoint .
```

The *basal metabolism* and the *shivering* components connected that (may) provide heat to the body core are modeled as `FlowSource` components.

Modeling Human Thermal Interactions. The thermal interactions in this case study are *radiation*, *convection*, and *blood flow*. We show how to define radiation, whose dynamics represents the rate of heat radiation given by $\dot{Q} = \varepsilon \cdot \sigma \cdot A \cdot (T_1^4 - T_2^4)$, where ε is the emissivity of the surface, σ is the Stefan-Boltzmann constant, and A is the surface area through which radiation takes place. The class defining radiation interactions is therefore defined by adding attributes for emissivity and area to the built-in class `PhysicalInteraction`, and its continuous dynamics is specified using the built-in function `flowDyn`:

```
class Radiation | area : Float, emmissiv : Float .
subclass Radiation < PhysicalInteraction .
eq flowDyn(TEMP1, TEMP2, < TI : Radiation | area : AREA, emmissiv : EMMISSIV >)
 = EMMISSIV * stefBoltzConst * AREA * ((TEMP1 ^ 4.0) - (TEMP2 ^ 4.0)) .
```

Modeling the heat flow between the body core and the skin through the blood flow is a more challenging problem, since the heat flow rate depends on the blood flow rate, which again depends on the diameter of the blood vessels, which can be changed by vasodilation and vasoconstriction (see our technical report for details).

Modeling the Hypothalamus. The hypothalamus regulates the body temperature based on some set points determining temperature thresholds for hot and cold exposure to the body. In our model, the hypothalamus is modeled by an object that senses the core and skin temperatures, and manages the activation of the shivering, the sweating, and the blood flow according to the following table:

	Temperature is normal	Temperature is too hot	Temperature is too cold

Current action: none	idle	do Sweating
		do Vasodilation
			do Shivering
			do Vasoconstriction

Current action: hot handling	stop Sweating
	do Vasonormal
		do Sweating
		do Vasodilation
			stop Sweating
			do Shivering
			do Vasoconstriction

Current action: cold handling	stop Shivering
	do Vasonormal
		stop Shivering
		do Sweating
		do Vasodilation
			do Shivering
			do Vasoconstriction

7.3 Formal Analysis

We model the human body as a vertical tube with height $1.7m$, diameter $0.3m$, body mass $63.5kg$, and body heat capacity $3.47kJ/kg°C$. The person is resting, with basal metabolic rate $0.08kW$. The room is $5m$ long, $5m$ wide, and $3m$ high, and is filled with air with heat capacity $1.005kJ/kg°C$ and density $1.2kg/m^3$. Moderate hyperthermia starts at the core temperature $38°C$, the severe hyperthermia at $41°C$, and the dead happens when the core temperature exceeds $45°C$. The set point for the core temperature is $37°C$ and the one for the skin is $33°C$.

We consider three persons: a healthy person, a person having high fever, and a person with brain damage in the hypothalamus part. The fever condition is modeled by increasing the core temperature set point of the hypothalamus by $1.5°C$. We want to analyze the effect of these different conditions to survive an extreme condition, and set the initial temperature for the body core, the skin, and the room at $37°C$, $33°C$, and $80°C$, respectively. The experiments have been performed on a computer with an Intel Pentium 4 CPU 3.00GHz and 3GB of RAM.

The following state `cs1` defines an initial state of a system with a healthy person:

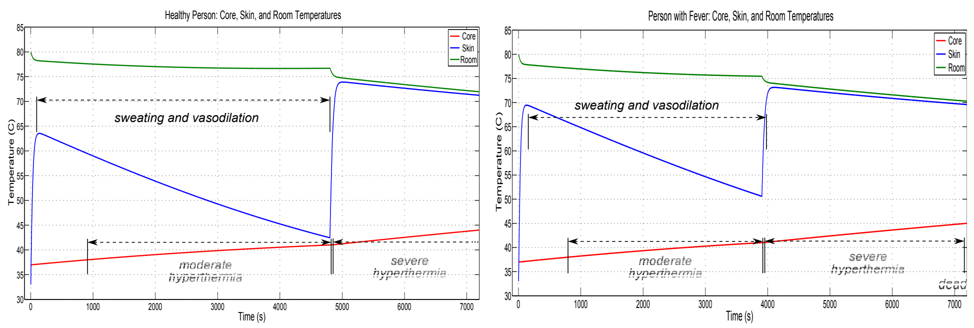

Fig. 3. The simulation results for the healthy and fever persons.

```
op cs1 : -> GlobalSystem
eq cs1 = {< core : CoreHumanBody | effort : 37.0 , mass : 0.85 * 63.5, heatCap : 0.85 * 3.47, coreState : normal >
          < skin : SkinHumanBody  | effort : 33.0 , mass : 0.15 * 63.5, heatCap : 0.85 * 3.47 >
          < room  : ThermalEntity  | effort : 80.0 , mass : 90.0, heatCap : 1.005 >
          < bloodf  : BloodFlow  | flow : 0.0, entity1 : core, entity2 : skin,  area : bodyArea,
                                   skinBloodFlowRate : 0.0315, bloodHeatCap : 3.85,  thermalCond : 0.00021,
                                   controller : hypothal, state : normal, . . . >
          < rad  : Radiation  | flow : 0.0, entity1 : skin, entity2 : room,  area : bodyArea, emmissiv : 0.97 >
          < convec : Convection | flow : 0.0, entity1 : skin, entity2 : room,  area : bodyArea, coeff : 0.0026 >
          < metabol  : BasalMetabol  | entity : core, flow : bodyMet, status : on >
          < shiver  : Shivering  | entity : core, flow : 0.0, status : off,  controller : hypothal  >
          < sweat  : Sweating  | entity1 : skin, entity2 : room, flow : 0.0, status : off , controller : hypothal >
          < hypothal  : Hypothalamus  | status : idle ,  hotSetPointCore : 37.5 , coldSetPointCore : 36.5,
                                       hotSetPointSkin : 34.5 , coldSetPointSkin : 12.0,  . . . >
          < intrecept  : InternalReceptor  |  entity : core, controller : hypothal, . . . >
          < extrecept  : ExternalReceptor  |  entity : skin, controller : hypothal, . . . >}  .
```

The state `cs2` adds an object which records key values in each step of the simulation [20] to `cs1`. We use hybrid rewriting to simulate the system for two hours:

```
Maude> (hrew cs2 in time 7200 using euler stepsize 1.0 .)
```

Fig. 3 shows the simulation results—the temperature of the body core, the skin, and the room temperatures as time advances—for a healthy person and for a person with fever. For the person with fever the sweating starts later since the shifting of the set point in the hypothalamus causing it sense the danger late.

We next use the *find earliest* command to analyze how long a person can stay in the sauna before he dies:

```
Maude> (hfind earliest
          cs1 =>* {C:Configuration < personCore : CoreHumanBody | coreState : dead >}
            using euler stepsize 1.0 .)
```

The following table shows how it long takes for each person to reach the various stages of discomfort (as well as the CPU time of the command execution):

	moderate hyperthermia	severe hyperthermia	dead
healthy	912 sec CPU: 14 sec	4802 sec CPU: 87 sec	8051 sec CPU: 251 sec
fever	803 sec CPU: 13 sec	3906 sec CPU: 90 sec	7197 sec CPU: 356 sec
brain damage	703 sec CPU: 2 sec	2880 sec CPU: 17 sec	6214 sec CPU: 83 sec

Many complex properties cannot be formulated as reachability problems, but may instead be defined as linear temporal logic (LTL) model checking problems. For example, the following command checks whether a moderately hyperthermic person will sweat and experience vasodilation until (s)he becomes severely hyperthermic:

```
Maude> (hmc cs1 |=t [] (modHyper -> (sweating-active /\ vasodilation-active) W sevHyper))
                     in time <= 7200 using euler stepsize 1.0 .)
```

where `modHyper`, `sweating-active`, `vasodilation-active`, and `sevHyper` are propositions denoting that the person is, respectively, moderately hyperthermic, sweating, vasodilating, and severely hyperthermic. For each person, the model checking returned the expected result (in 110 seconds of CPU time).

8 Concluding Remarks

We have introduced the HI-Maude tool that supports the formal modeling, simulation, and model checking of complex interacting hybrid systems in rewriting logic. The tool supports the compositional modeling of a complex system based our adaptation of the effort/flow approach developed in [5], and integrates the numerical approximation methods formalized in Real-Time Maude in [5, 7]. We have illustrated the use of the tool on the challenging human thermoregulatory system that features a set of interacting physical subsystems with complex continuous dynamics. Being based on rewriting logic, HI-Maude provides an intuitive yet expressive modeling language with support for concurrent objects, user-definable data types, different communication models, etc.

As usual much work remains. First of all, we should integrate techniques that dynamically adjust the step size used in the approximations to (i) make the analysis *more precise* by making the time step smaller when needed either to come close to a time instant when a discrete transition must be taken or when it is needed to maintain a desired precision of the approximation, and (ii) make the analysis *more efficient* by increasing the step size whenever the approximation allows it. In particular, adaptive step-size gives the user to possibility to define his/her own *error tolerance* to balance between desired precision and computational efficiency. Both of these features have been formalized in Real-Time Maude [6, 7] and must be integrated into HI-Maude. We should also study under what circumstances we can guarantee that the formal analyses are indeed sound and complete.

References

1. Clarke, E., Fehnker, A., Han, Z., Krogh, B., Stursberg, O., Theobald, M.: Verification of hybrid systems based on counterexample-guided abstraction refinement. In: Proc. TACAS'03. LNCS, vol. 2619. Springer (2003)
2. Clavel, M., Durán, F., Eker, S., Lincoln, P., Mart-Oliet, N., Meseguer, J., Talcott, C.: All About Maude - A High-Performance Logical Framework, LNCS, vol. 4350. Springer (2007)
3. Dang, T.: Verification and Synthesis of Hybrid Systems. Ph.D. thesis, INPG (2000)
4. Esposito, J., Kumar, V., Pappas, G.: Accurate event detection for simulating hybrid systems. In: Proc. of HSCC'01. LNCS, vol. 2034. Springer (2001)

5. Fadlisyah, M., Ábrahám, E., Lepri, D., Ölveczky, P.C.: A rewriting-logic-based technique for modeling thermal systems. In: Proc. RTRTS'10. Electronic Proceedings in Theoretical Computer Science, vol. 36 (2010)
6. Fadlisyah, M., Ábrahám, E., Ölveczky, P.C.: Adaptive-step-size numerical methods in rewriting-logic-based formal analysis of interacting hybrid systems. In: Proc. TTSS'10 (2010), to appear in *ENTCS*
7. Fadlisyah, M., Ölveczky, P.C., Ábrahám, E.: Formal modeling and analysis of hybrid systems in rewriting logic using higher order numerical methods and discrete-event detection. In: Proc. CSSE'11 (2011), to appear in *IEEE*
8. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. In: Proc. HSCC'05. LNCS, vol. 3414. Springer (2005)
9. Henzinger, T., Kopke, P., Puri, A., Varaiya, P.: What's decidable about hybrid automata? In: Journal of Computer and System Sciences. pp. 373–382. ACM Press (1995)
10. Henzinger, T.A., Horowitz, B., Majumdar, R., Wong-toi, H.: Beyond HYTECH: Hybrid systems analysis using interval numerical methods. In: Proc. of HSCC'00. LNCS, vol. 1790. Springer (2000)
11. Herman, I.: Physics of the Human Body. Springer (2007)
12. Katelman, M., Meseguer, J., Hou, J.: Redesign of the LMST wireless sensor protocol through formal modeling and statistical model checking. In: Proc. FMOODS'08. LNCS, vol. 5051. Springer (2008)
13. Lee, E., Zheng, H.: HyVisual: A hybrid system modeling framework based on Ptolemy II. In: IFAC Conference on Analysis and Design of Hybrid Systems (2006)
14. Lien, E., Ölveczky, P.C.: Formal modeling and analysis of an IETF multicast protocol. In: Proc. SEFM '09. IEEE (2009)
15. Ölveczky, P.C., Caccamo, M.: Formal simulation and analysis of the CASH scheduling algorithm in Real-Time Maude. In: Proc. FASE '06. LNCS, vol. 3922. Springer (2006)
16. Ölveczky, P.C., Meseguer, J.: Abstraction and completeness for Real-Time Maude. ENTCS 176(4), 5–27 (2007)
17. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20(1-2), 161–196 (2007)
18. Ölveczky, P.C., Meseguer, J.: The Real-Time Maude tool. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08). LNCS, vol. 4963, pp. 332–336. Springer (2008)
19. Ölveczky, P.C., Meseguer, J., Talcott, C.L.: Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. Formal Methods in System Design 29(3) (2006)
20. Ölveczky, P.C., Thorvaldsen, S.: Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in Real-Time Maude. Theoretical Computer Science 410(2-3) (2009)
21. Parsons, K.: Human Thermal Environments: The effects of hot, moderate, and cold environments on human health, comfort and performance. Taylor & Francis, London and New York (2003)
22. Plantadosi, C.: The Biology of Human Survival: Life and Death in Extreme Environments. Oxford University Press (2003)
23. *Simulink home page*. `http://www.mathworks.com/products/simulink`
24. Wellstead, P.E.: Introduction to physical system modelling. Academic Press (1979)