Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 274 (2011) 17–32

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Adaptive-Step-Size Numerical Methods in Rewriting-Logic-Based Formal Analysis of Interacting Hybrid Systems[⋆]

Muhammad Fadlisyah Peter Csaba Ölveczky

Department of Informatics, University of Oslo, Norway

Erika Ábrahám

Computer Science Department, RWTH Aachen University, Germany

Abstract

This paper focuses on the formal modeling, simulation, and analysis of *interacting* hybrid systems that influence each other's continuous behaviors. We define in the rewriting-logic-based Real-Time Maude tool a method for the numerical approximation of the continuous dynamics specified by ordinary differential equations. We adapt the Runge-Kutta-Fehlberg 4/5 method to define an *adaptive-step-size* technique that allows a more accurate approximation with less computational effort than fixed-step-size techniques. We also present experimental results for two thermal systems using different error tolerances.

Keywords: formal modeling, interacting physical systems, simulation, formal analysis, rewriting logic, Runge-Kutta-Fehlberg method

1 Introduction

Real-Time Maude [15] is a high-performance tool that extends the rewriting-logic-based Maude system [4] to support the formal modeling, simulation, and analysis of object-based real-time systems. Real-Time Maude emphasizes ease and expressiveness of specification, and has proved to be useful for analyzing a wide range of advanced applications that are beyond the scope of timed automata, such as communication protocols [16,13], wireless sensor network algorithms [11,17], and scheduling algorithms that need unbounded data structures [14].

This paper is part of an investigation into how Real-Time Maude can be used to formally model, simulate, and analyze *hybrid systems* with both discrete and

⋆ This work was partially supported by the Research Council of Norway through the Rhythm project and the DAAD ppp project HySmart.

doi:10.1016/j.entcs.2011.07.004

continuous behavior. In particular, we consider *interacting physical entities*, whose continuous behavior can be described by ordinary differential equations (ODEs). The physical entities interact and may 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.

Our goal is to develop a technique to generate *executable models* of such systems in Real-Time Maude. For the continuous behavior of physical systems, which is described by ODEs, in our previous work [7] the execution was based on *fixed-step-size* numerical methods giving approximate solutions to ordinary differential equations. That is, to approximate a system's behavior for a given time duration we approximate the behavior for a series a small time steps, each of them having a fixed duration. We used the Euler and the Runge-Kutta 2nd and 4th order methods for the small-step approximation.

In this paper we describe the integration of *adaptive-step-size* numerical methods, where the duration of the small time steps is chosen dynamically. Adaptive-step-size approximations have the advantages of: (i) making 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) making 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.

We develop an adaptation of the Runge-Kutta-Fehlberg 4/5 method (see e.g. [8]) that allows us to approximate the continuous behavior of our models with dynamic step-size, based on the error tolerance provided by the user. We describe the implementation of the adapted method in Real-Time Maude. Furthermore, we compare the results and execution times of both simulation and model checking using different error tolerances on two *thermal* systems with realistic parameters.

There are several simulation tools for hybrid systems based on numerical methods. MATLAB/Simulink [20] offers a wide range of numerical methods for simulation. HyVisual [12] considers linear multi-step and Runge-Kutta methods. CHARON [6] also uses linear multi-step methods with adaptive step size. In contrast to these tools, our approach supports, besides modeling and simulation, also the *formal analysis*, such as temporal logic model checking, of hybrid systems. Our approach also differs from model checkers for hybrid systems, such as CheckMate [3], PHAVer [9], d/dt [5], and HYPERTECH [10] in that we do not use abstraction or over-approximation, but still support the modeling, reachability analysis, and LTL model checking of the *full class* of hybrid systems, describing the continuous dynamics by (possibly non-linear) ODEs. Whereas other formal tools use hybrid automata, chart or block models, or formulas for modeling, we use *rewriting logic* as the underlying modeling formalism. The models of our approach are *compositional*, where the continuous dynamics of a component may depend on explicitly modeled *interactions* with other components. Since the logic also supports classes and objects as well as the definition of any computable data type, physical systems controlled by some programs can be intuitively modeled and analyzed.

The paper is structured as follows: Section 2 gives an overview of Real-Time Maude. Section 3 briefly explains our approach for modeling hybrid systems in rewriting logic. Section 4 presents the adaptation of the Runge-Kutta-Fehlberg 4/5 method for our purposes, and Section 5 describes its implementation in Real-Time Maude. The case studies are summarized in Section 6 and concluding remarks are given in Section 7.

2 Real-Time Maude

A Real-Time Maude *timed module* specifies a *real-time rewrite theory* (Σ, E, IR, TR), where:

- (Σ, E) is a *membership equational logic* [4] theory with Σ a signature[1] and E a set of confluent and terminating *conditional equations.* (Σ, E) specifies the state space as an algebraic data type, and contains a specification of a sort `Time` modeling the time domain.
- IR is a set of (possibly conditional) *labeled instantaneous rewrite rules* specifying the system's *instantaneous* (i.e., zero-time) one-step transitions. The rules are applied *modulo* the equations E.[2]
- TR is a set of (possibly conditional) *tick rewrite rules* that model time elapse, written with syntax

```
rl [l] :  {t} => {t'} in time τ
crl [l] : {t} => {t'} in time τ if cond
```

  where τ is a term of sort `Time` denoting the *duration* of the rewrite.

The global states of a system are terms of the form `{`t`}`; the form of the tick rules then ensures that time advances uniformly in a system. The Real-Time Maude syntax is fairly intuitive (see [4]). For example, a function f with arguments of sorts $s_1 \ldots s_n$ and value of sort s is declared `op` f `:` $s_1 \ldots s_n$ `->` s. Equations are written `eq` t `=` t', and `ceq` t `=` t' `if` *cond* for conditional equations. Variables are declared with the keywords `var` and `vars`.

A *class* declaration `class` C `|` att_1 `:` $s_1, \ldots,$ att_n `:` s_n `.` declares a class C with attributes att_1 to att_n of sorts s_1 to s_n. A *subclass* inherits all attributes and rules of its superclasses. An *object* of class C is represented as a term `<` O `:` C `|` att_1 `:` $val_1, ..., att_n$ `:` val_n `>` of sort `Object`, where O, of sort `Oid`, is the object's *identifier*, and val_1 to val_n are the current values of the attributes att_1 to att_n. In a concurrent object-oriented system, a state is a term of the sort `Configuration`. It has the structure of a *multiset* made up of objects and possibly *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.

---

[1] i.e., declarations of *sorts*, *subsorts*, and *function symbols*

[2] E is a union $E' \cup A$, where A is a set of equational axioms (associativity, commutativity, identity, etc.). Deduction is performed *modulo* A. A term is reduced to its E'-normal form modulo A before any rewrite rule is applied.

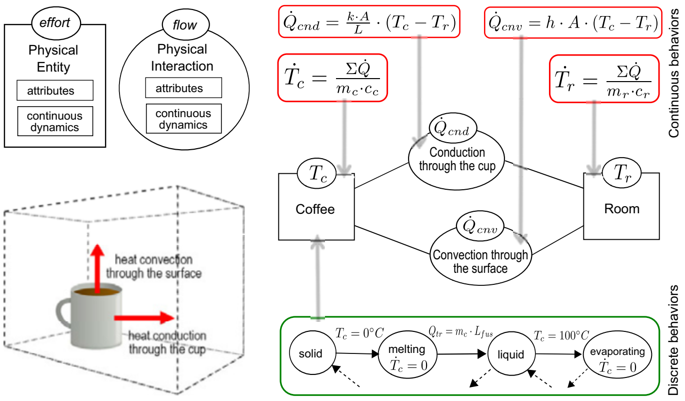
effort
Physical Entity
attributes
continuous dynamics
flow
Physical Interaction
attributes
continuous dynamics
$\dot{Q}_{cnd} = \frac{k \cdot A}{L} \cdot (T_c - T_r)$
$\dot{Q}_{cnv} = h \cdot A \cdot (T_c - T_r)$
$\dot{T}_c = \frac{\Sigma \dot{Q}}{m_c \cdot c_c}$
$\dot{T}_r = \frac{\Sigma \dot{Q}}{m_r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

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

Real-Time Maude specifications are *executable* under reasonable conditions, and the tool offers a variety of formal analysis methods. The *rewrite* command `(trew` t `in time <=` τ `.)` simulates *one* fair behavior of the system *up to duration* τ, where t is the initial state and τ is a term of sort `Time`. The *search* command uses breadth-first search to analyze all possible behaviors of the system and checks whether a state matching a *pattern* can be reached from the initial state such that a given *condition* is satisfied. 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 linear temporal logic formula. Finally, the `find earliest` command determines the shortest time needed to reach a desired state.

3 Modeling Physical Systems

In [7] we present a framework for the modeling and analysis of physical systems based on the *effort* and *flow* approach [21]. One key difference between our work and most other formal approaches to hybrid systems is that, instead of considering the continuous behavior of a component in isolation, we consider a hybrid system to consist of a set of physical components, where the continuous dynamics of a component may depend on the continuous dynamics of other components. The *physical interactions* between physical entities are therefore considered as first-class citizens, and a physical system is modeled as a network of *physical entities* and *physical interactions*, as shown in Figure 1.

A *physical entity* consists of a set of *attributes*, a real-valued *effort variable*, and a *continuous dynamics*. The attribute values can only be changed by discrete events. For example, the *phase* of a material (solid, liquid, gas, plasma) changes via discrete *phase transitions*. The effort variable represents a physical quantity, such as temperature, evolving over time. Its continuous dynamics is given as an ordinary differential equation (ODE). A physical entity can have one or more *physical interactions* with one or more physical entities. A physical interaction represents an interaction between two physical entities. It consists of a set of *attributes*, a real-valued *flow variable*, and a *continuous dynamics*. The flow variable represents a quantity describing the interaction between two entities, e.g., the heat flow rate in a thermal interaction. Its value is determined by the continuous dynamics in the form of an equation.

The continuous dynamics of a physical entity is an ODE with the time derivative of its effort on the left-hand side and an expression possibly referring to the entity's attributes *and* to the flows of connected interactions on the right-hand side. 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 [2] can be avoided.

Fig. 1 shows a *thermal system* representing a cup of of coffee in a room. In thermal systems, a physical entity is a *thermal entity*, whose effort variable (T) denotes the temperature of the entity and whose continuous dynamics defines the heat gained or lost by the entity as time evolves and its temperature changes. Likewise, a physical interaction is a *thermal interaction* whose flow variable ($\dot{Q}$) denotes the heat flow rate. Examples of thermal interactions are conduction, convection, and radiation. Their continuous dynamics are given by equations for the heat transfer rates.

The basic behavior of physical system components is their continuous behavior. We use *single-step*, *initial-value-problem numerical* methods [2] to approximate the continuous behaviors of physical system components by advancing time in small discrete time steps, and computing the values of the continuous variables at each "visited" point in time. In previous work, we have integrated the Euler, Runge-Kutta 2nd order, and Runge-Kutta 4th order methods to our modeling technique. However, in these methods, the size of the small time steps in the execution is constant.

4 Adaptive-Step-Size Numerical Methods

To approximate some continuous behavior with rapid variations or abrupt changes, for *fixed* step-size methods we have to choose a small step-size to get satisfactory results. However, small step-sizes come at a very high computational cost. For example, approximating the coffee system in Fig. 1 with our implementation of the Runge-Kutta 4th order method for 1000 time units took 38 minutes using step size 1, but took 285 minutes using step size 0.5. For systems with more stable dynamics, larger step-size can be used to get an adequate approximation more efficiently. The idea of *adaptive* step-size techniques is to adapt the trajectory of the approximation by estimating and controlling the error at each step. Such error estimates are used as a basis for dynamically increasing or decreasing the step size.

4.1 Approximation Errors

Assume a continuous variable y with time derivative $y'(t) = f(t, y(t))$ and an initial value $y(0) = y_0$. To approximate $y(T)$ for some $T > 0$, small-step numerical methods compute a sequence of values $y_1, y_2, ..., y_N$ that approximate the exact

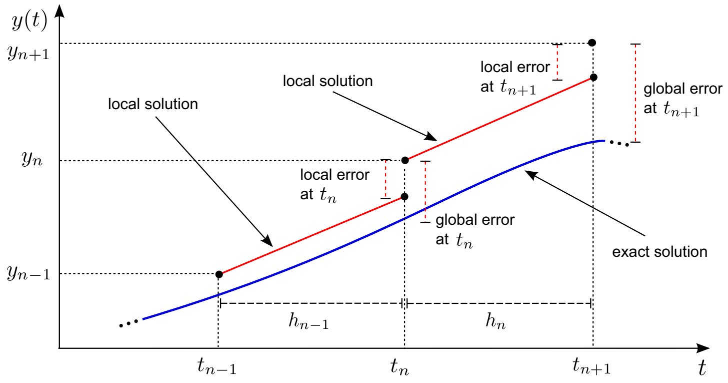

Fig. 2. The local and global errors in a numerical approximation.

values $y(t_1), y(t_2), ..., y(t_N)$ for some time points $t_1 < t_2 < ... < t_N$, $t_N = T$. Computing y_{n+1} is based on the value of y_n, and can be seen as taking a small time step with the *step size* $h_n = t_{n+1} - t_n$. For the methods with *fixed* step-size, all h_n, $n = 0, \ldots, N-1$, are equal. This is not the case for methods with *adaptive* step size.

There are two sources of errors in the above approach. *Round-off errors* are due to the limitations of computers in representing numbers. Discrete-time *approximation errors* originate from the fact that the approximations $y_1, y_2, ..., y_N$ deviate from the exact values $y(t_1), y(t_2), ..., y(t_N)$. If we assume that an exact arithmetic is used and thus there are no round-off errors, the deviation $\varepsilon_n^g = y(t_n) - y_n$ is called the *global error* at time point t_n (see Fig. 2).

The global error ε_{n+1}^g sums up from the global error at t_n and its propagation, and an additional error due to the last approximation. Let the *local solution* u be the solution of $u'(t) = f(t, u(t))$ for the initial value $u(t_n) = y_n$. Then the global error at t_{n+1} is $\varepsilon_{n+1}^g = (y(t_{n+1}) - u(t_{n+1})) + (u(t_{n+1}) - y_{n+1})$. The first summand is the global error $\varepsilon_n^g = y(t_n) - u(t_n)$ propagated during the time step from t_n to t_{n+1} yielding the error $y(t_{n+1}) - u(t_{n+1})$. With other words, it is the difference at time point t_{n+1} of two solutions of the ODE that differ by $y(t_n) - y_n$ at time point t_n. The second summand, which we call the *local error* $\varepsilon_{n+1}^l = u(t_{n+1}) - y_{n+1}$, is the approximation error of the last step.

4.2 Adjusting the Step Size

We cannot control the global error directly. However, we can control it *indirectly* by controlling the *local error* in each time step [19]. [3] We need to make the step size adaptive such that the local error in each step is bound by some error tolerance. The error tolerance is determined by a user-given value τ. Two commonly used definitions [18] are *error per step* requiring $|\varepsilon_{n+1}^l| \leq \tau$, and *error per unit step* defining $|\varepsilon_{n+1}^l| \leq \tau \cdot h_n$ as condition for each $n \geq 0$.

To check if these conditions are fulfilled we need to measure the local error, which

[3] Note that the global error cannot be controlled by the numerical methods. If the solutions of the ODEs are unstable, the global error can grow fast even for small local errors.

depends on the *order* of the method used. The order of a numerical method corresponds to how fast a sequence of approximations generated by a method converges toward the expected solution. The higher the order, the better the approximation. One way to estimate the local error ε^l_{n+1} for the approximation y_{n+1} by a method of order $p \geq 1$ is to compute the approximation $\hat{y}_{n+1}$ also with a higher order $\hat{p} > p$ method.[4] The local error of the method of order p can be estimated by comparing the results $\varepsilon^l_{n+1} \approx \epsilon_{n+1} = \hat{y}_{n+1} - y_{n+1}$. It can be shown that it is a correct asymptotic result when $h \to 0$ [19].

If the local error passes the test, this error is accepted, and the step size will be increased for the next step. For the error per step condition we define $h_{n+1} = \alpha \cdot h_n$ with $\alpha = \left(\frac{\tau}{|\epsilon_{n+1}|}\right)^{\frac{1}{p+1}}$. Note that $|\epsilon_{n+1}| \leq \tau$ implies $\alpha \geq 1$. If the local error does not pass the test, this step size is rejected and will be decreased. For the error per step condition we define $h'_n = \alpha \cdot h_n$. Note that, since the condition was not satisfied, we have $\alpha < 1$.

4.3 The Runge-Kutta-Fehlberg Order 4/5 Method

Given a numerical method of a certain order, any other numerical method of a higher order can be used to obtain an estimate of the local error. However, computing a second approximation using a second methods in each time step may be computationally expensive [1]. This problem can be avoided by using methods that share function values which are known as *embedded pairs*.

The *Runge-Kutta-Fehlberg* order 4/5 method (RKF45) [8] makes use of such embedded pairs. It uses a 5th order method to estimate the local error of a 4th order method. In each step, for the approximation with the 4th order method 5 values (slopes) must be calculated that are used to compute y_{n+1} as a weighted sum. The 5th order method, used for the error estimation, needs 6 slope values. However, 5 of them are already computed by the 4th order method, thus only one slope must additionally be determined.

The 5th order approximation $\hat{y}_{n+1}$ is computed to estimate the local error of the 4th order approximation y_{n+1}. However, since higher order methods yield more exact results than lower order ones, we use $\hat{y}_{n+1}$ instead of y_{n+1} as approximation result. This technique is called *local extrapolation*.

5 Integrating the Adaptive-Step-Size Method for Modeling Thermal Systems

This section gives an overview on the integration of the adaptive-step-size technique based on the RKF45 method into our modeling framework to support the formal modeling and analysis of hybrid systems with interacting components in Real-Time Maude. Fig. 3 shows the global framework.

[4] Note that the computation for $\hat{p}$ can re-use most of the computations for p, see Section 4.3.

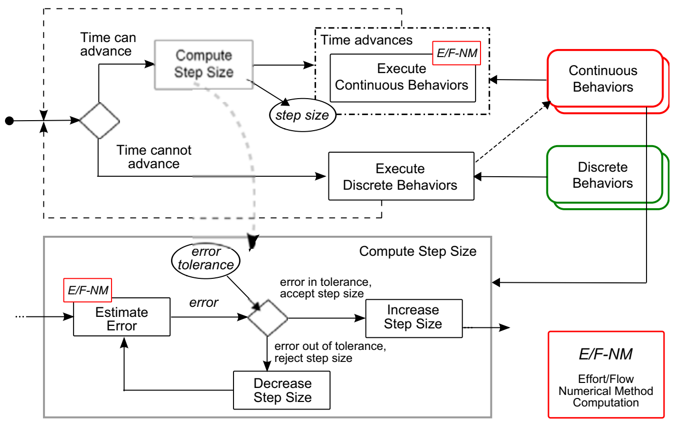

Fig. 3. The execution of a physical system model with adaptive step size.

5.1 *Modeling Thermal Systems in Real-Time Maude*

We illustrate our approach thermal systems as the one shown in Fig. 1. Physical entities in thermal systems are *thermal entities*, and physical interactions are *thermal interactions*. The temperature of a thermal entity changes according to $\dot{T} = \frac{\Sigma\dot{Q}}{m \cdot c}$, where $\Sigma\dot{Q}$ is the sum of the heat flow rates of the connected thermal interactions, m is the entity's mass, and c is its heat capacity. Likewise, the heat flow rate $\dot{Q}$ between two entities through *conduction* is defined by $\dot{Q} = \frac{k \cdot A}{L} \cdot (T_1 - T_2)$, where T_1 and T_2 are the current temperatures of the interacting entities, and k, L, and A are, respectively, the thermal conductivity, the thickness, and the area of the entity from which heat flows by conductivity.

We model thermal entities in Real-Time Maude as objects of the following class `ThermalEntity`:

```
class ThermalEntity | temperature : Rat,  mode : CompMode,  mass : PosRat,  heatCap : PosRat,
                      temp-p : Rat,  temp-o1 : Rat,  temp-o2 : Rat .
```

The effort variable `temperature` represents the entity's temperature. The attribute `mode` is used to distinguish between different modes for the continuous dynamics (see below). The attributes `mass` and `heatCap` denote the mass and the heat capacity of the entity, respectively. `temp-p`, `temp-o1`, `temp-o2` are auxiliary attributes used for the computation by the RKF45 method. The entity's continuous dynamics, described below, specifies the evolution of the temperature, depending on the heat transfer from or to the object.

We can define more specific types of thermal entities as subclasses of the base class `ThermalEntity`. For example, the following class `WaterEntity` represents water substance:

```
class WaterEntity | phase : MatterState,  heatTrans : Rat, heatTrans-p : Rat,
                    heatTrans-o1 : Rat, heatTrans-o2 : Rat .
subclass WaterEntity < ThermalEntity .
sort MatterState .
ops liquid solid gas melting evaporating condensing freezing : -> MatterState .
ops default phaseTrans : -> CompMode .
```

The attribute `phase` represents the phase of the water substance, which can be one of the main phases solid, liquid, gas, or one of the phase transitions melting, freezing, evaporating, or condensing. The change from a main phase to a phase transition occurs when the temperature reaches a given value, whereas a change from a phase transition to a main phase happens when the value of the heat accumulated during the phase transition divided by the mass of the entity reaches a given value called the latent heat. The attribute `heatTrans` stores the accumulated heat of the water in the phase transitions. The remaining attributes are needed for the computation of the approximations.

The `mode` determines the computation mode for the continuous dynamics. For water, the continuous dynamics of the temperature is the same in all three main phases, whereas the temperature does not change during phase transitions. In addition to the `default` computation mode for the main phases, we add the mode `phaseTrans` for phase transitions.

Each phase change is modeled by an instantaneous rewrite rule. We show two of the eight such rules for water:[5]

```
crl [solid-to-melting] :
    < E : WaterEntity | temp : T,  phase : solid >
    => < E : WaterEntity | phase : melting,  mode : phaseTrans,  heatTrans : 0 > if T >= 0 .
crl [melting-to-liquid] :
    < E : WaterEntity  | phase : melting,  heatTrans : QT,  mass : M >
    => < E : WaterEntity  | phase : liquid,  mode : default > if QT / M >= latentHeatFusion .
```

Thermal Interactions model the heat transfer between thermal entities. Examples are conduction, convection, and radiation. We define a class for general thermal interactions and subclasses for the three heat transfer mechanisms:

```
class ThermalInteraction | entity1 : Oid,  entity2 : Oid,  hfr : Rat,  area : PosRat,
                           hfr-p1 : Rat,  hfr-p2 : Rat,  hfr-p3 : Rat,  hfr-p4 : Rat,  hfr-p5 : Rat .
class Conduction | thermCond : PosRat,  thickness : PosRat .
class Convection | convCoeff : PosRat .
class Radiation  | emissive  : PosRat .
subclass Conduction Convection Radiation < ThermalInteraction .
```

The `ThermalInteraction` class contains common attributes of thermal interactions: `entity1` and `entity2` are the object identifiers of the two interacting thermal entities; the flow variable `hfr` specifies the heat flow rate $\dot{Q}$ of the thermal interaction; `area` is the area of the interaction; `hfr-p1` to `hfr-p5` are auxiliary attributes used for the computation by the RKF45 method. The subclasses have additional interaction-specific attributes. For conductivity, `thermCond` is the thermal conductivity of the material and the `thickness` is the thickness of the material through which the conduction occurs.

5.2 Computing the Step Size

We define the following class to manage the numerical method computation:

```
ops euler mp rk4 rkf45 : -> NumMethod [ctor] .
ops adj1 adj2 : -> CompStepSize [ctor] .
ops static dynamic : -> StepSizeType [ctor] .
ops eps epus : -> ErrorControlType [ctor] .

class SysMan | numMethod : NumMethod, stepSizeDef : Rat, stepSizeCur : Rat,
               stepSizeType : StepSizeType, errorTol : Rat, compStepSize : CompStepSize,
               safetyFactor : Rat, limitStepSize : Bool, stepSizeMin : Rat, stepSizeMax : Rat,
               limitAdjustRate : Bool, adjustRateMin : Rat, adjustRateMax : Rat,
```

[5] We do not show the variable declarations; instead we follow the Maude convention that variables are written in capital letters, and that function symbols (including constants) start with a lower-case letter.

```
                errorControl : ErrorControlType, localExtrapolation : Bool .
```

The attribute `numMethod` specifies which numerical method is used. The attribute `stepSizeDef` stores the initial step-size, and `stepSizeCur` the current step-size. The `stepSizeType` determines if fixed or adaptive step-size is used. The `errorTol` defines the error tolerance in adaptive-step-size computation (assuming that we use a single tolerance value). The `compStepSize` defines which step-size computation technique is used. The `safetyFactor` defines a fraction of the locally optimal step size which may be used to reduce the approximation error. The `limitStepSize`, `stepSizeMin`, and `stepSizeMax` limit the value of the step size. The `limitAdjustRate`, `adjustRateMin`, and `adjustRateMax` are used to limit increasing or decreasing rate of the step size. The `errorControl` chooses either *error per step* or *error per unit step* for controlling the step size. The `localExtrapolation` specifies whether to use the extrapolation technique in the numerical computation.

The RKF45 method stores the approximations for the temperature values by the 4th and 5th order methods in the attributes `temp-o1` and `temp-o2` of the thermal entities. The function `maxError` computes the maximal local error estimate over all thermal entities in the system:

```
op maxError : Configuration -> Rat .
eq maxError(
    < E : ThermalEntity | temp-o1 : TEMP-O1, temp-o2 : TEMP-O2, mode : default >
    < SM : SysMan | errorControl : eps > REST) =
    max(abs(TEMP-O1 - TEMP-O2), maxError(< SM : SysMan | > REST)) .
eq maxError(CONFIG) = 0 [owise] .
```

The `adjustRate` function computes the factor of the step size adjustment using error per step (we have a similar function for error per unit step):

```
op adjustRate : CompStepSize Rat Rat Rat ErrorControlType -> Rat .
eq adjustRate(adj1, ERR, ERRTOL, SAF, eps) =  SAF * root5(ERRTOL / ERR) .
```

The function `root5(X)` computes $X^{\frac{1}{5}}$ as $exp(1/5 \cdot \ln(X))$.

The function `stepSizeRKF` computes the step size based on the RKF45 method. If the maximal local error is below the tolerance value, it returns a pair of values, consisting of the current step size and the step size for the next time step. If not, the function is recursively called after computing new approximations with a smaller step-size. The following equation defines this function when putting no limitations on the step-size modifications:

```
op stepSizeRKF : Configuration -> ErrorStepSize .
ceq stepSizeRKF(
    < SM : SysMan | compStepSize : ADJ, stepSizeCur : SSCUR, errorTol : ERRTOL, safetyFactor : SAF,
                limitStepSize : false, limitAdjustRate : false, errorControl : ERRCTR > REST ) =
    if ERR <= ERRTOL then SSCUR ; SSRKF
    else stepSizeRKF(compute-EF-RKF45(< SM : SysMan | stepSizeCur : SSRKF > REST)) fi
if ERR := maxError(< SM : SysMan | > REST) /\ SSRKF := adjustRate(ADJ, ERR, ERRTOL, SAF, ERRCTR) * SSCUR .
```

We also implemented a similar function that limits the rate of step-size change.

The following *tick rule* advances time in the system by the step size computed by `stepSizeRKF`, and computes the new values of the effort variables for all thermal entities. These values are the 5th order approximations if the extrapolation is used, and the 4th order approximations otherwise.

```
crl [tick-adaptive-stepsize] :
    < SM : SysMan | stepSizeDef : SS, stepSizeType : dynamic, errorTol : ERRTOL > REST
```

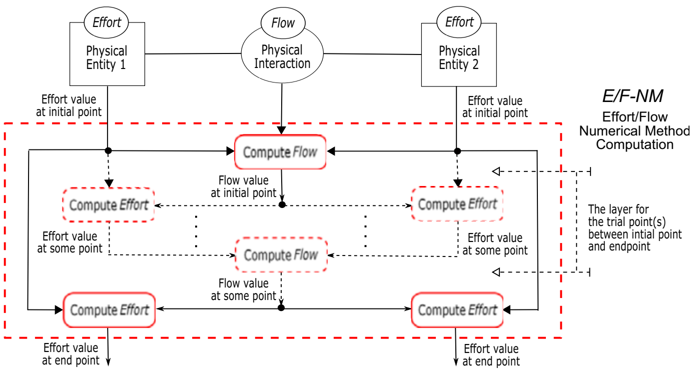
Effort
Physical Entity 1
Flow
Physical Interaction
Effort
Physical Entity 2
Effort value at initial point
Effort value at initial point
E/F-NM
Effort/Flow Numerical Method Computation
Compute Flow
Flow value at initial point
Compute Effort
Compute Effort
Effort value at some point
Compute Flow
Effort value at some point
The layer for the trial point(s) between intial point and endpoint
Flow value at some point
Compute Effort
Compute Effort
Effort value at end point
Effort value at end point

Fig. 4. The general model of the adaptation of numerical methods in `effort/flow` computation.

```
    =>
    delta(< SM : SysMan |  stepSizeCur : firstES(SSPAIR), stepSizeDef : secondES(SSPAIR)>
           REST) in time firstES(SSPAIR)
if TimeCanAdvance(< SM : SysMan | > REST)
   /\ SSPAIR := stepSizeRKF(compute-EF-RKF45(< SM : SysMan | stepSizeCur : SS > REST)) .

eq delta(< SM : SysMan | numMethod : rkf45, localExtrapolation : true > REST) =
   compute-EF-RKF45-Order5(< SM : SysMan | > REST) .
eq delta(< SM : SysMan | numMethod : rkf45, localExtrapolation : false > REST) =
   compute-EF-RKF45-Order4(< SM : SysMan | > REST) .
```

5.3 *Integrating the RKF45 Method*

The general model for adapting numerical methods in our *effort/flow* framework is depicted in Fig. 4. We use time discretization, and compute approximations for each small time-step. To compute the approximations by a numerical method, some slopes k_1 to k_n must be computed. [6] For the RKF45 method we need six slopes k_1 to k_6, as explained in Section 4. For each k_i we need to compute a linear approximation of the behavior for a small time-step, starting from some initial point. This is done by (1) first calculating the heat flow rates of all thermal interactions at the initial point, (2) summing up the heat flow rates for all connected interactions for each entity, and (3) linearly approximate the effort, i.e. the temperature, after a small time-step, assuming that the computed heat flow rates are constant. Due to lack of space, in the following we restrict ourselves to explain these computation steps for k_1 (up to k_n, the other cases are similar but use different auxiliary attributes).

The function `computeFlow-IP` computes the heat flow rate of each thermal interaction for the initial point according to the laws of physics as described in [7]. We only show the case for thermal conductions:

```
op computeFlow-IP : Configuration -> Configuration .
ceq computeFlow-IP(
   < E1 : ThermalEntity | temperature : T1 >
   < E2 : ThermalEntity | temperature : T2 >
   < TI : Conduction    | entity1 : E1, entity2 : E2, area : A, thermCond : K, thickness : L > REST) =
   < TI : Conduction    | hfr : QDOT-T >
   computeFlow-IP(< E1 : ThermalEntity | > < E2 : ThermalEntity | > REST)
if QDOT-T := Qdot-Conduction(K, L, A, T1, T2) .

eq computeFlow-IP(CONFIG) = CONFIG [owise] .
```

[6] In our previous works, we have applied this technique for the Euler, Runge-Kutta 2nd order, and Runge-Kutta 4th order methods.

The equation above computes the initial heat flow rate for a thermal interaction `TI` of type `Conduction` between two thermal entities `E1` and `E2`, and then recursively applies the function to the remaining configuration. The function `Qdot-Conduction` defines the dynamics as $\dot{Q} = \frac{K \cdot A}{L} \cdot (T_1 - T_2)$.

The function `sumFlows-IP` computes the sum of the initial heat flow rates of all thermal interactions connected to a thermal entity:

```
op sumFlows-IP : Configuration Oid -> Rat .
eq sumFlows-IP(
   < TI : ThermalInteraction | entity1  : E1, entity2 : E2, hfr : QDOT > REST, E) =
   if (E == E1 or E == E2) then
      (if E == E1 then -1 * QDOT + sumFlows-IP(REST, E)
       else QDOT + sumFlows-IP(REST, E) fi)
   else sumFlows-IP(REST, E) fi .

eq sumFlows-IP(CONFIG, E) = 0 [owise] .
```

The function `computeEffort-P1` linearly approximates the temperature of each thermal entity in the system after a time step, assuming constant flow rates over the time step. It invokes the function `Tdot` representing the continuous dynamics given by $\dot{T} = \frac{\sum \dot{Q}}{m \cdot c}$, where $\sum \dot{Q}$ is the sum of the heat flow rate values of the thermal interactions of the entity as computed by `sumFlows-IP`, m the mass, and c the heat capacity. The attributes `numMethod` and `stepSize` of `SM` determine the numerical method and time step size, respectively:

```
op computeEffort-P1 : Configuration -> Configuration .
ceq computeEffort-P1(
   < E  : ThermalEntity | temperature : TEMP, mode   : default, mass : M, heatCap : C >
   < SM : SysMan        | numMethod   : mp, stepSize : H > REST) =
   < E  : ThermalEntity | temp-p : TEMP-P1 >
   computeEffort-P1(< SM : SysMan | > REST)
if TEMP-P1 := TEMP + 1/4 * H * Tdot(sumFlows-IP(REST,E),M,C).

eq computeEffort-P1(CONFIG) = CONFIG [owise] .
```

6 Case Studies

This section investigates how the adaptation of the adaptive step size technique based on the RKF45 method affects the accuracy and performance of the simulation and time analysis of thermal systems. We start with a cup of hot coffee in a room. Then we add a heater giving a constant heat flow to the coffee.[7] The experiments are performed on a computer with an Intel(R) Pentium(R) 4 CPU 3.00 GHz and 3 GB of RAM. The executable formal models, as well as the simulation and analysis commands described below, are available at `http://www.ifi.uio.no/RealTimeMaude/Coffee/`.

6.1 Case Study 1: A Cup of Coffee in a Room

We first model a cup of hot coffee in a room, as shown in Fig. 1, with *conduction* and *convection* as thermal interactions, and with realistic physical parameters. The initial state consists of a `SysMan` object managing the numerical computation, the thermal entity objects `coffee` and `room`, and two thermal interaction objects that model the heat flow:

[7] In the analysis we use the *error per step*, and the *extrapolation* for the computation.

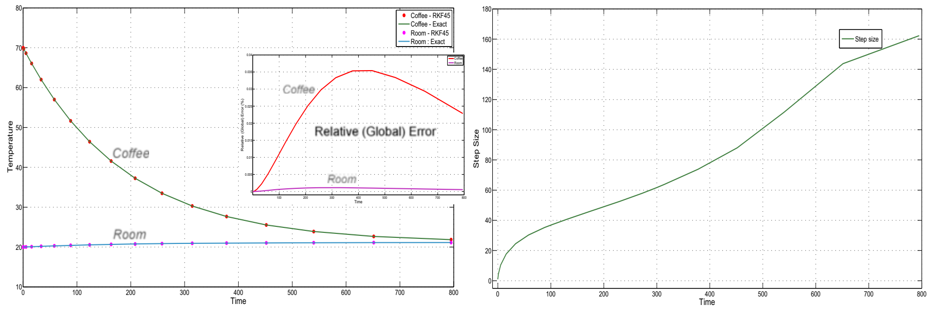

Fig. 5. The simulation results of Case Study 1 using error tolerance 10^{-3}.

```
eq cs1 =
{< coffee : WaterEntity | temperature : 70 , heatCap : coffeeHC, mass : coffeeMass, mode : default,
                           temp-p : 0, temp-o1 : 0, temp-o2 : 0, phase : liquid, heatTrans : 0,
                           heatTrans-p : 0, heatTrans-o1 : 0, heatTrans-o2 : 0 >
< room : ThermalEntity | temperature : 20 , heatCap : roomHC, mass : roomMass, mode : default,
                         temp-p : 0, temp-o1 : 0, temp-o2 : 0,
< crCond : Conduction | entity1 : scoffee, entity2 : room, hfr : 0, thermCond : k , area : condArea,
                        thickness : cupThick, hfr-p1 : 0, hfr-p2 : 0, hfr-p3 : 0, hfr-p4 : 0, hfr-p5 : 0 >
< crConv : Convection | entity1 : scoffee, entity2 : room, hfr : 0, convCoeff : h , area : convArea,
                        hfr-p1 : 0, hfr-p2 : 0, hfr-p3 : 0, hfr-p4 : 0, hfr-p5 : 0 >
< sm : SysMan | numMethod : rkf45, stepSizeCur : INIT-TIME-STEP, stepSizeDef : INIT-TIME-STEP,
                stepSizeType : dynamic, errorTol : 1/1000, compStepSize : adj1, safetyFactor : 9/10,
                limitStepSize : false, stepSizeMin : 1/10, stepSizeMax : 5, limitAdjustRate : false,
                adjustRateMin : 1/100, adjustRateMax : 1/4, errorControl : eps,
                localExtrapolation : true >} .
```

The behavior of the system until time 500 can be simulated using the following timed rewriting command:

```
Maude> (trew cs1 in time <= 500 .)
```

Fig. 5 shows the simulation results using the error tolerance 10^{-3}. The diagram on the left shows how the temperature of the coffee decreases and the one of the room increases as the heat flows. The small diagram inside shows the change of the relative values of the global error of the approximation of both temperatures at each time step.[8] . The diagram on the right shows the change of the step size which is getting larger as time advances.

The following table compares the simulation results using different error tolerances for the simulation time of 500:

Error Tolerance	*Effort*	**Error Abs (+ Rel %)**			**CPU Time (s)**
		Min	Max	Avg	
10^{-3}	T_c	3.4741e-06 (4.9815e-06)	0.0101 (0.0354)	0.0056 (0.0162)	4
	T_r	8.1800e-08 (4.0930e-07)	2.3796e-04 (0.0011)	1.3196e-04 (6.3503e-04)	
10^{-4}	T_c	3.4741e-06 (4.9815e-06)	0.0036 (0.0123)	0.0024 (0.0071)	75
	T_r	8.1800e-08 (4.0930e-07)	8.5957e-05 (4.1164e-04)	5.7330e-05 (2.7591e-04)	
10^{-5}	T_c	3.4741e-06 (4.9815e-06)	0.0012 (0.0041)	8.9203e-04 (0.0026)	3881
	T_r	8.1800e-08 (4.0930e-07)	2.8990e-05 (1.3895e-04)	2.1026e-05 (1.0123e-04)	

It shows the values of minimum, maximum, and average of absolute and relative values of global errors of both temperatures of the coffee T_c and the room T_r, including the execution time. The results show that by decreasing the error tolerance we increase *indirectly* the accuracy of the computation, with the consequent increase in the computation time. The values of maximum and average errors which are

[8] The relative approximation error percentage is computed using $\frac{|\mathrm{val}_{\mathrm{exact}} - \mathrm{val}_{\mathrm{approx}}|}{|\mathrm{val}_{\mathrm{exact}}|}$. Its percentage error is 100 times the relative error.

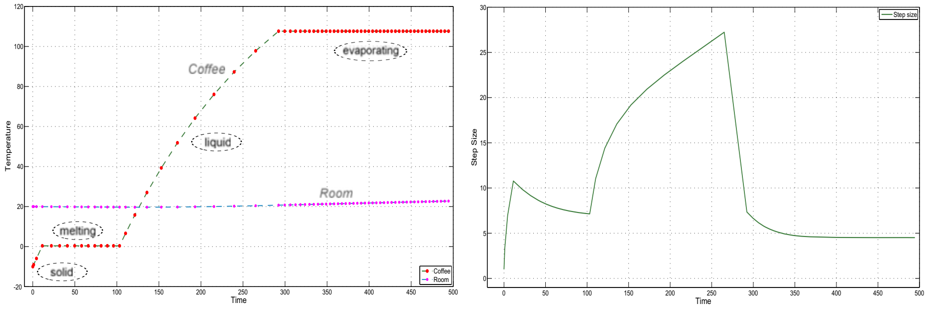

Fig. 6. The simulation results of Case Study 2 using error tolerance 10^{-3}.

greater than the error tolerances remind us again that the error tolerance does not control *directly* the global error. Note that the error values of the coffee are greater those of the room because the change rate of the temperature of the coffee is greater than the room.

6.2 Case Study 2: Keeping the Coffee Warm

To illustrate hybrid behavior in thermal systems, we add a heater providing a constant heat flow of 1.5 KW to the cup of coffee. We start with an initial coffee temperature of $-10°C$ to go over the phase transitions from solid to liquid through the melting phase transition. Due to lack of space, we refer to [7] for a detailed model description.

Fig. 6 shows the simulation results using the error tolerance of 10^{-3}. The diagram on the left shows how both temperatures of the coffee and the room increase as the coffee receives constant heat flow from the heater. It shows how the discrete behavior of the coffee, namely the changes from one physical state to the other (here *solid* to *melting*, *melting* to *liquid*, and *liquid* to *evaporating*), affect its continuous dynamics. The diagram on the right shows the changes of the step size, but unlike the previous case study, here the step size increases and decreases following the changes of the dynamics of the coffee.

We model the changes in the phase transition phenomena as discrete events that change the dynamics of the physical entities. We can use the *find earliest* Real-Time Maude command to find out discrete changes in our model. For example, we use this command to check when our coffee starts melting:

```
Maude> (find earliest cs2 =>*
          C:Configuration < coffee : WaterEntity | phase : melting > .)
```

The following table compares the results and execution time of the above command using different error tolerances:

Error Tol	**Discrete Jump**		**CPU Time (s)**
	Time Point	T_c **approx**	
10^{-3}	11.3135531852	0.3858197647	0.6
10^{-4}	15.9889535171	4.5019701020	0.8
10^{-5}	11.6013479838	0.6425970213	1.2
10^{-6}	11.0388434197	0.1414667933	11.1

The results show an expected correlation between the error tolerance and the execution time. However, the results do not show the 'gradual' changes of the discrete jump, as we may expect. The coffee is supposed starting the melting process at temperature $0°C$. Thus we expect that as the error tolerance decreases, the approximate value of the coffee temperature will be closer to zero. But the results above cannot show our expectation. The following table shows the corresponding traces to the jump points from the simulation using different tolerances:

Time point	10^{-3}	10^{-4}	10^{-5}	10^{-6}
t_{j-2}	-9.0569907324	-4.0031039563	-2.9549415161	-1.0583268132
t_{j-1}	-5.9239797872	-0.0510841430	-1.1707648220	-0.4578893885
t_j	0.3858197647	4.5019701020	0.6425970213	0.1414667933

The results shows that the changes of the coffee temperature value from one time point to another correlate to the error tolerances. However, the execution of a discrete event depends on the execution strategy for hybrid behaviors. For the implementation presented in this paper, the check of the occurrence of a discrete event is performed before the time step is taken, but a discrete event that should occur between time t_n and t_{n+1} is executed at t_{n+1}.

7 Concluding Remarks

In this paper we describe how the adaptive-step-size technique based on the Runge-Kutta-Fehlberg 4/5 method can be adapted to an effort-flow-based modeling of *interacting* physical systems, and how the methods can be implemented in Real-Time Maude. We have compared the precision and execution times for some thermal systems, and showed that decreasing the error tolerances increases both the accuracy of the approximation of the continuous behavior and the computational effort. We also found a weakness in our execution strategy of the hybrid behavior when using adaptive step size techniques.

Making these methods, and a modeling framework, available within the Real-Time Maude rewriting logic tool should make it a suitable candidate for the object-based formal modeling, simulation, and model checking of advanced hybrid systems due to the tool's expressiveness, support for concurrent objects, user-definable data types, different communication models, etc. In future work, this should be validated this on more advanced systems.

Acknowledgement

We thank Lawrence F. Shampine for helpful discussions.

References

[1] Bradie, B., "A Friendly Introduction to Numerical Analysis," Pearson Education Int., 2006.

[2] Cheney, W. and D. Kincaid, "Numerical Mathematics and Computing," Brooks & Cole Publishing Co., 1994.

[3] Clarke, E., A. Fehnker, Z. Han, B. Krogh, O. Stursberg and M. Theobald, *Verification of hybrid systems based on counterexample-guided abstraction refinement*, in: *Proc. TACAS'03*, LNCS **2619** (2003).

[4] Clavel, M., F. Durán, S. Eker, P. Lincoln, N. Mart-Oliet, J. Meseguer and C. Talcott, "All About Maude - A High-Performance Logical Framework," LNCS **4350**, Springer, 2007.

[5] Dang, T., "Verification and Synthesis of Hybrid Systems," Ph.D. thesis, INPG (2000).

[6] Esposito, J., V. Kumar and G. Pappas, *Accurate event detection for simulating hybrid systems*, in: *Proc. of HSCC'01*, LNCS **2034** (2001).

[7] Fadlisyah, M., E. Ábrahám, D. Lepri and P. C. Ölveczky, *A rewriting-logic-based technique for modeling thermal systems*, in: *Proc. RTRTS'10*, EPTCS **36**, 2010.

[8] Fehlberg, E., *Klassische Runge-Kutta-formeln vierter und niedrigerer ordnung mit schrittweiten-kontrolle und ihre anwendung auf wrmeleitungsprobleme*, Computing (1970).

[9] Frehse, G., *PHAVer: Algorithmic verification of hybrid systems past HyTech*, in: *Proc. of HSCC'05*, LNCS **3414** (2005), pp. 258–273.

[10] Henzinger, T. A., B. Horowitz, R. Majumdar and H. Wong-toi, *Beyond HYTECH: Hybrid systems analysis using interval numerical methods*, in: *Proc. of HSCC'00*, LNCS **1790** (2000).

[11] Katelman, M., J. Meseguer and J. Hou, *Redesign of the LMST wireless sensor protocol through formal modeling and statistical model checking*, in: *Proc. FMOODS'08*, LNCS **5051** (2008).

[12] Lee, E. and H. Zheng, *HyVisual: A hybrid system modeling framework based on Ptolemy II*, in: *IFAC Conference on Analysis and Design of Hybrid Systems*, 2006.

[13] Lien, E. and P. C. Ölveczky, *Formal modeling and analysis of an IETF multicast protocol*, in: *Proc. SEFM '09* (2009).

[14] Ölveczky, P. C. and M. Caccamo, *Formal simulation and analysis of the CASH scheduling algorithm in Real-Time Maude*, in: *Proc. FASE '06*, LNCS **3922** (2006).

[15] Ölveczky, P. C. and J. Meseguer, *Semantics and pragmatics of Real-Time Maude*, Higher-Order and Symbolic Computation **20** (2007), pp. 161–196.

[16] Ölveczky, P. C., J. Meseguer and C. L. Talcott, *Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude*, Formal Methods in System Design **29** (2006).

[17] Ölveczky, P. C. and S. Thorvaldsen, *Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in Real-Time Maude*, Theoretical Computer Science **410** (2009).

[18] Shampine, L., "Numerical solution of ordinary differential equations," Chapman & Hall, 1994.

[19] Shampine, L. F., *Local error control in codes for ordinary differential equations*, Applied Mathematics and Computation **3** (1977), pp. 189 – 210.

[20] Simulink home page, `http://www.mathworks.com/products/simulink`.

[21] Wellstead, P. E., "Introduction to physical system modelling," Academic Press, 1979.