Modellazione di un sistema ibrido per la rilevazione di incendio e controllo temperatura

Chiara Zandonà - VR407847

***Sommario*—Questo report spiega la modellazione di un sistema ibrido formato da due stanze in cui in momenti diversi viene rilevato un incendio (la temperatura ha raggiunto i $600°$). L'obbiettivo è stato quello di implementare un sistema per riuscire a riportare la temperatura di entrambe le stanze a condizioni accettabili (circa $20°$) nello stesso momento.**

I. Introduzione

Il flusso di verifica formale nei sistemi hardware e software si propone di modellare il sistema e provare o smentire, tramite una risposta o a un controesempio, la correttezza degli algoritmi di esso controllando le proprietà di interesse. Per fare ciò bisogna fornire, quindi, un modello matematico astratto del sistema che lo rappresenti ad esempio le macchine a stati finiti o le reti di Petri. Ci sono, però, sistemi con una doppia natura che evolvono nel sistema continuo ma i loro comportamenti vengono controllati da un sistema discreto. Questi sistemi, chiamati automi ibridi, permettono di rappresentare al meglio fenomeni presenti in natura grazie a un potere espressivo maggiore degli altri modelli matematici. Per verificare se un sistema dinamico soddisfa certe proprietà, bisogna descrivere il suo comportamento con un insieme di stati raggiungibili e osservare l'evoluzione del sistema. Per i sistemi non lineari questo insieme di stati raggiungibili non è computabile e viene quindi approssimato.

Il caso di applicazione qui preso in analisi viene trattato come un fenomeno presente in natura: sono presenti due stanze in cui accidentalmente il termostato si rompe e la temperatura inizia a salire con velocità diverse scatenando due incendi. Appena il sensore di temperatura rileva l'incendio ($600°$) si apre la valvola per portare acqua e raffreddare la stanza fino al completo estinguersi del fuoco. All'inizio l'incendio è stato rilevato in una sola stanza quindi la valvola è aperta al massimo portando tutta l'acqua possibile. Successivamente anche nell'altra stanza scatta l'incendio e quindi l'acqua ora deve essere divisa tra le due stanze in modo proporzionale facendo così poi decrescere la temperatura insieme. Il processo termina quando entrambe le stanze raggiungono contemporaneamente circa i $20°$.

Per descrivere questo sistema ci si è avvalsi della libreria C++ *Ariadne* che utilizza il formalismo degli automi ibridi per descrivere sistemi non lineari a tempo continuo. Permette quindi di fare verifica formale utilizzando tecniche di analisi numerica per avere l'evoluzione in modo approssimato.

Il report è organizzato nel seguente modo: nella sezione II verrà introdotto il sistema con le formule utilizzate, come sono state costruite le componenti e modellato l'intero sistema, nella sezione III si descrivono le problematiche riscontrate, nella sezione IV i risultati e quindi i grafici ottenuti, nella sezione V le conclusioni e nella sezione VI la sitografia del progetto.

II. Modello

Il progetto è stato sviluppato prendendo spunto dall'articolo scientifico "Modellazione dell'incendio e delle esplosioni" scritto dall'ex responsabile dell'Ufficio Operativo Interventistico del Servizio Antincendi e Protezione civile della Provincia autonoma di Trento, Architetto Roberto Lenzi. In questo documento viene spiegato il processo di innescamento ed estinzione di un incendio nelle varie modalità. Il caso preso in esame non considera agenti esterni che accelerano l'innescamento dell'incendio. Le formule, quindi prese ad esempio sono le seguenti:

PRIMA FORMULA DI RISCALDAMENTO per t $<$ 21s

$$temp = 154t^{0.25} + 20$$

SECONDA FORMULA DI RISCALDAMENTO per t $\geq$ 21s

$$temp = 345log_{10}[8(t-20)+1)] + 20$$

PRIMA FORMULA DI RAFFREDDAMENTO per t $\leq$ 0.5

$$temp = tempmax - 625(t - tfire * x)$$

SECONDA FORMULA DI RAFFREDDAMENTO per 0.5 $<$ t $<$ 2

$$temp = tempmax - 250(3-t)(t - tfire * x)$$

TERZA FORMULA DI RAFFREDDAMENTO per t $\geq$ 2

$$temp = tempmax - 250(t - tfire * x)$$

Per semplificare le formule è stata eliminata la dipendenza dal tempo t inserendo delle costanti che indicano il tempo in cui si è entrati nella locazione e sono stati modificati i fattori moltiplicativi per avere due stanze con caratteristiche diverse.

Il modello finale è formato da quattro componenti:due stanze in cui scoppia l'incendio in momenti diversi, una valvola proporzionale che una volta rilevato l'incendio inizia a spegnerlo in modo che entrambe le due stanze siano a una temperatura tollerabile insieme e un clock che funge da timer.

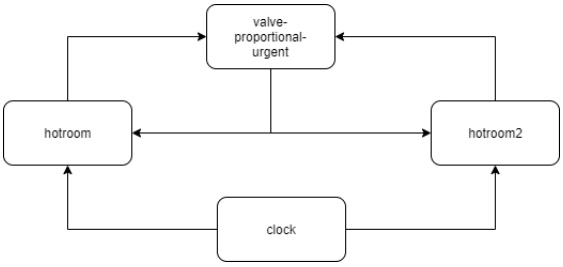

Figura 1: FSM del sistema Fire

La temperatura inizia a crescere contemporaneamente nella stanza *hot_room* e *hot_room2*, partendo da 20° gradi, a velocità diverse. All'istante temporale 1.0940 la stanza1 raggiunge i 600° quindi il sensore di temperatura rileva l'incendio e manda una richiesta urgente alla valvola dell'impianto antincendio di aprirsi per iniziare a raffreddare la stanza (valore della valvola *aperture1=1* mentre per la stanza2 il valore è 0). All'istante temporale 1.3034 anche nella stanza2 scoppia un incendio e ora la valvola proporzionale deve dividere il quantitativo di acqua tra le due stanze in modo da estinguere gli incendi contemporaneamente.

Si è partiti implementando una sola stanza nella quale a 600° scoppiava l'incendio e successivamente iniziava a raffreddarsi fino a raggiungere i 20° circa. La stanza ha due comportamenti di riscaldamento e tre di raffreddamento regolati da un timer che indica quanto tempo bisogna restare in una locazione. Inoltre è presente uno stato per delimitare la fine dell'esecuzione (*stopHeating*). Il sistema è quindi un automa temporizzato e delimitato superiormente dalla temperatura massima di incendio, 600° e inferiormente dalla temperatura minima raggiunta 20° circa.

Successivamente è stata implementata la seconda stanza *hot_room2* uguale alla prima se non che si raggiunge l'incendio più tardi (istante temporale 1.3034). Di seguito si possono vedere gli automi con le transizioni della *hot_room*, della *hot_room2* e del *clock* che viene utilizzato in entrambe.

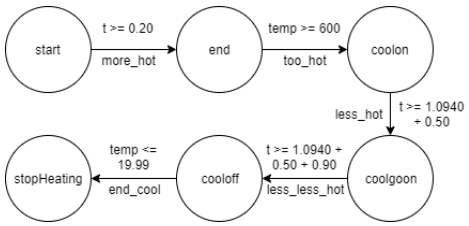

Figura 2: Automa della stanza1 hot_room

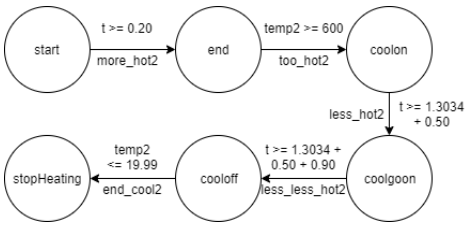

Figura 3: Automa della stanza2 hot_room2

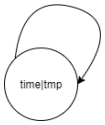

Figura 4: Automa del clock

Viene riportato qui di seguito nel listing 1 il codice che implementa la prima stanza, *hot_room*, mentre nel listing 2 quello della seconda, *hot_room2* con alcune costanti modificate rispetto all'articolo teorico per avere gli incendi nei tempo desiderati: 1.0940 e 1.3034.

Listing 1: Mode hot_room

```
                                                                    1
hr.new_mode( heating| start ,                                       2
{dot(temp) = 945*pow(t,Nat(0.25))+200});                            3
hr.new_mode( heating|end,                                           4
{dot(temp) = 950*((log(8*(0.20-0.0005)+1))/log(10))+                5
             0.0005});                                              6
hr.new_mode( heating|coolon,                                        7
{dot(temp) = 600+((-400)*(1.0940-((1.0940+0.01)*(-2.0))))});        8
hr.new_mode( heating|coolgoon,                                      9
{dot(temp) = 600+((-330)*(2.1-(1.0940+0.50))*                       10
             (1.0940-((1.0940+0.50)*(-2.0))))});                    11
hr.new_mode( heating|cooloff,                                       12
{dot(temp) = 600+((-330)*(1.0940-((1.0940+0.50+0.90))*              13
             (-1.30)))});                                           14
hr.new_mode( heating|stopHeating,                                   15
{dot(temp) = 0});                                                   16
                                                                    17
hr. new_transition ( heating | start , more_hot, heating |end,     18
{next(temp)=temp}, t>=0.20, EventKind::URGENT );                    19
hr. new_transition ( heating |end, too_hot, heating |coolon,        20
{next(temp)=temp}, temp >=600, EventKind:: URGENT);                 21
hr. new_transition ( heating |coolon, less_hot , heating |coolgoon, 22
                    {next(temp)=temp},                              23
                    t>=1.0940+0.50, EventKind::URGENT);             24
hr. new_transition ( heating |coolgoon, less_less_hot , heating |   25
      cooloff , {next(temp)=temp}, t>=1.0940+0.50+0.90,
      EventKind::URGENT );
hr. new_transition ( heating | cooloff , end_cool, heating |       26
      stopHeating , {next(temp)=temp}, temp<=19.99,
      EventKind::URGENT );
```

Listing 2: Mode hot_room2

```
hr2.new_mode( heating2| start ,                                     1
{dot(temp2) = 700*pow(t,Nat(0.25))+200});                           2
hr2.new_mode( heating2|end,                                         3
{dot(temp2) = 875*((log(8*(0.20-0.0005)+1))/log(10))+               4
              0.0005});                                             5
hr2.new_mode( heating2|coolon,                                      6
{dot(temp2) = 600+((-450)*(1.3034-(1.3034+0.01)*(-1.10)))});        7
hr2.new_mode( heating2|coolgoon,                                    8
{dot(temp2) = 600+((-250)*(2.7-(1.3034+0.50)))*                     9
              (1.3034-(1.3034+0.50)*(-1.10))});                     10
hr2.new_mode( heating2|cooloff,                                     11
{dot(temp2) = 600+((-250)*(1.3034-(1.3034+0.50+0.90)*               12
              (-1.90)))});                                          13
hr2.new_mode( heating2|stopHeating,                                 14
{dot(temp2)= 0});                                                   15
                                                                    16
hr2. new_transition ( heating2| start , more_hot2, heating2|end,    17
      {next(temp2)=temp2}, t>=0.20, EventKind::URGENT );
hr2. new_transition ( heating2|end, too_hot2, heating2|coolon,      18
      {next(temp2)=temp2}, temp2 >=600, EventKind::
      URGENT);
hr2. new_transition ( heating2|coolon, less_hot2 , heating2 |       19
      coolgoon, {next(temp2)=temp2}, t>=1.3034+0.50,
      EventKind::URGENT );
hr2. new_transition ( heating2|coolgoon, less_less_hot2 , heating2  20
      | cooloff ,{next(temp2)=temp2}, t>=1.3034+0.50+0.90,
      EventKind::URGENT );
hr2. new_transition ( heating2| cooloff , end_cool2, heating2 |     21
      stopHeating , {next(temp2)=temp2}, temp2<=19.99,
      EventKind::URGENT );
```

Sono state quindi dichiarate le locazioni, le dinamiche (chiamate mode) di esse e le transizioni, tutte urgenti, da uno stato all'altro. Le stanze passano dal primo comportamento di riscaldamento al secondo dopo 0.20 istanti di tempo, nel secondo mode rimangono fino al raggiungimento dell'incendio, poi aspettano 0.50 istanti in *coolon*, 0.90 istanti in *coolgoon* ed infine rimangono in *cooloff* fino alla fine dell'esecuzione. Le due stanze evolvono con una dinamica indipendente l'una dall'altra in quanto condividono solo il *clock* facente funzione di timer. La classe *fire* si occupa quindi di fare la composizione tra *hot_room*, *hot_room2* e *clock*, di inizializzare l'insieme di partenza e di tenere traccia, con dei grafici, dell'evoluzione del sistema.

Listing 3: Composizione del sistema *fire*

```
//composizione del sistema fire
HybridAutomaton hotroom_system = create_heating_system();
HybridAutomaton clock_system = getClock();
HybridAutomaton hotroom2_system = create_heating_system2();
CompositeHybridAutomaton heating_system({clock_system,
      hotroom_system, hotroom2_system});

// Creato un GeneralHybridEvolver object
GeneralHybridEvolver evolver(heating_system);
evolver.verbosity = evolver_verbosity;

// Settati i parametri di evoluzione
evolver.configuration().set_maximum_enclosure_radius(0.01);
evolver.configuration().set_maximum_step_size(0.001);

// Dichiarato il tipo che viene usato per l'evoluzione del
      sistema
typedef GeneralHybridEvolver::OrbitType OrbitType;

TaylorSeriesIntegrator integrator(MaximumError(1e-1), Order
      (3));
evolver.set_integrator(integrator);

std::cout << "Computing evolution..." << std::flush;
```

Listing 4: Inizializzazione e plotting del sistema *fire*

```
// Inizializzazione dell'insieme di partenza
Real Tinitmin(Ti+r); Real Tinitmax(Ti+3*r); Real Cinitmin(0+r
      );
HybridSet initial_set({heating|start, time|tmp, heating2|
      start},{temp==20, t==0,temp2==20});

// Evoluzione del sistema
HybridTime evolution_time(2.8425,100);
OrbitType orbit = evolver.orbit(initial_set, evolution_time,
Semantics::UPPER);
std::cout << "done." << std::endl;

// Stampa dei grafici
std::cout << "Plotting trajectory ..."<<std::flush;
Axes2d time_temp_axes(0<=TimeVariable()<=evolution_time.
      continuous_time()+1,19<=temp<=1000);
plot("FireHotRoomTTemp",time_temp_axes, Colour(0.0,0.5,1.0),
      orbit);
std::cout << "Fine plot 1" << "\n";
Axes2d time_temp2_axes(0<=TimeVariable()<=evolution_time.
      continuous_time()+1,19<=temp2<=1000);
plot("FireHotRoomTTemp2",time_temp2_axes, Colour(0.0,0.5,1.0)
      , orbit);
std::cout << "Fine plot 2" << "\n";

std::cout << "done." << std::endl;
```

A questo punto è stata introdotta l'ultima componente del sistema: la valvola proporzionale, classe *valve-proportional-urgent*, che divide la quantità d'acqua durante il raffreddamento affinché le due stanze raggiungano una temperatura ragionevole insieme. La valvola proporzionale è stata modellata come un automa regolato dalle temperature e non dal timer come invece accade nelle stanze.

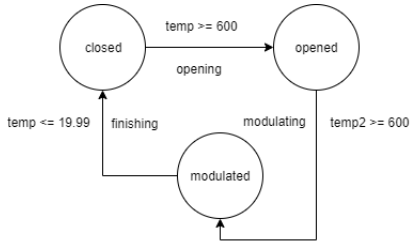

Figura 5: Automa valvola proporzionale

La valvola all'inizio è nello stato *closed*, infatti, non essendo stato rilevato l'incendio in nessuna delle due stanze, non deve portare acqua per ridurre le temperature. Non appena all'instante 1.0940 nella *hot_room* si raggiungono i 600° gradi, l'automa entra nella locazione *coolon* e quindi la dinamica viene influenzata dal valore della variabile *aperture1* che viene settata a 1 nello stato *opened*. Quando anche nella *hot_room2* a 1.3034 si percepisce l'incendio si passa alla locazione della valvola successiva, *modulated* e, per ogni instante di tempo successivo, viene calcolato il valore di *aperture1*, tra 0 e 1, date le temperature attuali nelle stanze in modo da portare la giusta quantità d'acqua:

VALORE VALVOLA PER HOT_ROOM

$$aperture1 = temp/(temp + temp2)$$

VALORE VALVOLA PER HOT_ROOM2

$$aperture2 = 1 - aperture1$$

La seconda formula nel codice non appare con la variabile *aperture2* infatti, considerata superflua, viene sostituita dall'espressione algebrica *1-aperture1*. Essa corrisponde come valore a calcolare per la temperatura della seconda stanza l'apertura della valvola *aperture2= temp2/(temp+temp2)*. Non appena si raggiunge la temperatura minima in entrambe le stanze, la valvola torna allo stato *closed* in quanto ha finito di raffreddare le stanze.

Listing 5: Mode e transizioni valvola proporzionale

```
valve_automaton.new_mode(opened,{let(aperture1)=1});
valve_automaton.new_mode(closed,{let(aperture1)=0});
valve_automaton.new_mode(modulated,
{let(aperture1)=temp/(temp+temp2)});

valve_automaton.new_transition(closed,opening,opened,
temp>=600,EventKind::URGENT);

valve_automaton.new_transition(opened,modulating,modulated,
temp2>=600,EventKind::URGENT);
valve_automaton.new_transition(modulated, finishing,closed,
temp<=19.99_decimal,EventKind::URGENT);
```

A questo punto sono state modificate anche le costanti moltiplicative delle varie locazioni nelle stanze per permettere la terminazione del programma contemporaneamente. Senza questa modifica, essendo variato il fattore correttivo dei tre comportamenti rispetto alla prima versione (da una costante moltiplicativa alla stessa divisa per il valore della valvola nella

stanza), l'esecuzione finiva troppo rapidamente non permettendo un tempo di raffreddamento dato dalla somma degli istanti passati nelle locazioni di raffreddamento: 0.50 in *coolon* + 0.90 in *coolgoon* + tempo per arrivare a circa 20° in *cooloff*.

Listing 6: Mode e transizioni hot_room con valvola proporzionale

```
hr.new_mode( heating|start,
{dot(temp) = 945*pow(t,Nat(0.25))+200});
hr.new_mode( heating|end,
{dot(temp) = 950*((log(8*(0.20-0.0005)+1))/log(10))+0.0005});
hr.new_mode( heating|coolon,
{dot(temp) = 600+((-400)*(aperture1)*(1.0940-
        ((1.0940+0.01)*(-2.7))))});
hr.new_mode( heating|coolgoon,
{dot(temp) = 600+((-330)*(aperture1)*(2.1-(1.0940+0.50))*
        (1.0940-((1.0940+0.50)*(-2.7/aperture1))))});
hr.new_mode( heating|cooloff,
{dot(temp) = 600+((-330)*(aperture1)*(1.0940-
        ((1.0940+0.50+0.90))*(-1.302/aperture1)))});
hr.new_mode( heating|stopHeating,
{dot(temp) = 0});

hr.new_transition( heating|start, more_hot, heating|end,
{next(temp)=temp}, t>=0.20, EventKind::URGENT );
hr.new_transition( heating|end, too_hot, heating|coolon,
{next(temp)=temp}, temp >=600, EventKind:: URGENT);
hr.new_transition( heating|coolon, less_hot, heating|coolgoon,
    {next(temp)=temp}, t>=1.0940+0.50, EventKind::
    URGENT );
hr.new_transition( heating|coolgoon, less_less_hot, heating|
    cooloff, {next(temp)=temp}, t>=1.0940+0.50+0.90,
    EventKind::URGENT );
hr.new_transition( heating|cooloff, end_cool, heating|
    stopHeating, {next(temp)=temp}, temp<=19.99,
    EventKind::URGENT );
```

Listing 7: Mode hot_room2 con valvola proporzionale

```
hr2.new_mode( heating2|start,
{dot(temp2) = 700*pow(t,Nat(0.25))+200});
hr2.new_mode( heating2|end,
{dot(temp2) = 875*((log(8*(0.20-0.0005)+1))/log(10))+
        0.0005});
hr2.new_mode( heating2|coolon,
{dot(temp2) = 600+((-500)*(1-aperture1)*(1.3034-
        (1.3034+0.01)*(-1.30/1-aperture1)))});
hr2.new_mode( heating2|coolgoon,
{dot(temp2) = 600+((-490)*(1-aperture1)*(2.7-
        (1.3034+0.50)))*(1.3034-(1.3034+0.50)*
        (-1.30/1-aperture1));
hr2.new_mode( heating2|cooloff,
{dot(temp2) = 600+((-490)*(1-aperture1)*(1.3034-
        (1.3034+0.50+0.90)*(-1.90/1-aperture1)))});
hr2.new_mode( heating2|stopHeating,
{dot(temp2)= 0});

hr2.new_transition( heating2|start, more_hot2, heating2|end,
    {next(temp2)=temp2}, t>=0.20, EventKind::URGENT);
hr2.new_transition( heating2|end, too_hot2, heating2|coolon,
    {next(temp2)=temp2}, temp2 >=600, EventKind::
    URGENT);
hr2.new_transition( heating2|coolon, less_hot2, heating2|
    coolgoon, {next(temp2)=temp2}, t>=1.3034+0.50,
    EventKind::URGENT );
hr2.new_transition( heating2|coolgoon, less_less_hot2,
    heating2|cooloff, {next(temp2)=temp2},
    t>=1.3034+0.50+0.90, EventKind::URGENT );
hr2.new_transition( heating2|cooloff, end_cool2, heating2|
    stopHeating, {next(temp2)=temp2}, temp2<=19.99,
    EventKind::URGENT );
```

Si mostra la nuova classe *fire* che, come nella prima formulazione del progetto, compone le varie parti e le fa funzionare insieme:

Listing 8: Composizione sistema *fire* precedente con valvola

```
// creo il sistema fire con valvola
HybridAutomaton hotroom_system=create_heating_system();
HybridAutomaton clock_system=getClock();
HybridAutomaton hotroom2_system=create_heating_system2();
AtomicHybridAutomaton valve_system = getValve();
CompositeHybridAutomaton heating_system({clock_system,
    hotroom_system, hotroom2_system, valve_system});

// Creato un GeneralHybridEvolver object
GeneralHybridEvolver evolver(heating_system);
evolver.verbosity = evolver_verbosity;

// Settati i parametri dell'evoluzione
evolver.configuration().set_maximum_enclosure_radius(0.01);
evolver.configuration().set_maximum_step_size(0.001);

// Dichiarato il tipo che viene usato per l'evoluzione del
    sistema
typedef GeneralHybridEvolver::OrbitType OrbitType;

TaylorSeriesIntegrator integrator(MaximumError(1e-1), Order
    (3));
evolver.set_integrator(integrator);

std::cout << "Computing evolution..." << std::flush;
```

Listing 9: Inizializzazione e plotting sistema *fire* precedente con valvola

```

// Inizializzazione dell'insieme di partenza
HybridSet initial_set({heating|start, time|tmp, valve|closed,
    heating2|start},{temp==20, t==0,temp2==20});

// Evoluzione del sistema
HybridTime evolution_time(2.7199,100);
OrbitType orbit = evolver.orbit(initial_set, evolution_time,
    Semantics::UPPER);
std::cout << "done." << std::endl;

// Stampa dei grafici
std::cout << "Plotting trajectory ..."<<std::flush;
Axes2d time_temp_axes(0<=TimeVariable()<=evolution_time.
    continuous_time()+1,19<=temp<=1000);
plot("FireHotRoomTTemp",time_temp_axes, Colour(0.0,0.5,1.0),
    orbit);
std::cout << "Fine plot 1" << "\n";
Axes2d time_temp2_axes(0<=TimeVariable()<=evolution_time.
    continuous_time()+1,19<=temp2<=1000);
plot("FireHotRoomTTemp2",time_temp2_axes, Colour(0.0,0.5,1.0)
    , orbit);
std::cout << "Fine plot 2" << "\n";
Axes2d time_valve_axes(0<=TimeVariable()<=evolution_time.
    continuous_time()+1,-0.1_decimal<=aperture1<=1.3
    _decimal);
plot("FireHotRoomTAperture1",time_valve_axes, Colour
    (0.0,0.5,1.0), orbit);
std::cout << "Fine plot 3" << "\n";
std::cout << "done." << std::endl;
```

III. Problematiche riscontrate

Si elencano le problematiche incontrate durante la modellazione del sistema, le cause e le soluzioni adottate:

PROBLEMATICA 1: La prima difficoltà è stata nel capire come implementare la dinamica delle stanze con un'equazione differenziale in quanto le formule sfruttano il *clock* come un timer. Con una sola stanza si poteva definire un *clock* interno ma con l'aggiunta di *hot_room2* il tempo doveva essere gestito esternamente da un altro file header e composto nel file main *fire.cpp* chiamando la variabile tempo t nello stesso modo in tutti i file.

PROBLEMATICA 2: Quando si è passati dalla prima versione del codice (due stanze senza valvola proporzionale) alla versione finale con la valvola, è stato difficile capire perché con gli stessi mode la temperatura non iniziasse a decrescere raggiunto l'incendio. Praticamente finché il valore della valvola era implicitamente 1 (non essendo presente nelle dinamiche del primo caso) la temperatura in *coolon, coolgoon e cooloff* decresceva, mentre con un valore tra 0 e 1 essa in fase di raffreddamento cresceva e non decresceva meno velocemente, come ci si aspettava. Probabilmente questa problematica è stata dovuta alla poca stabilità delle formule usate che non prevedevano un caso simile. Si è comunque risolto il problema facendo dipendere il fattore correttivo dei comportamenti di raffreddamento dal valore della valvola *aperture1* e per questo, come descritto nella sezione precedente, sono state modificate le costanti moltiplicative.

PROBLEMATICA 3: Nel modello finale come si può vedere, nella prossima sezione, il grafico della valvola non mostra una connessione tra i vari comportamenti e l'esecuzione va in *core dumped* se viene fatta durare più a lungo. Il problema dalla descrizione dell'errore indica che il passo di integrazione è arrivato al limite inferiore e non può essere ridotto ulteriormente. Sono stati provati *maximum step_size* più piccoli, ma deve essere un problema legato anche alla valvola in quanto nella prima versione del codice non è presente. Purtroppo si sta ancora investigando su come si possa risolvere questo errore.

IV. Risultati e grafici

Si mostrano i grafici delle due stanze senza la valvola: come si può vedere la temperatura cresce fino a $600°$ con due comportamenti e poi decresce fino a circa $20°$ con altri tre comportamenti.

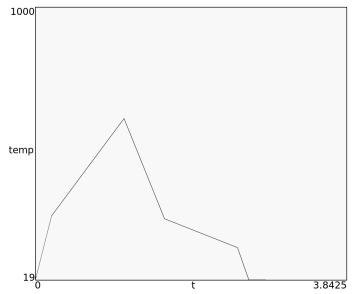

Figura 6: Grafico *hot_room*- temperatura rispetto al tempo

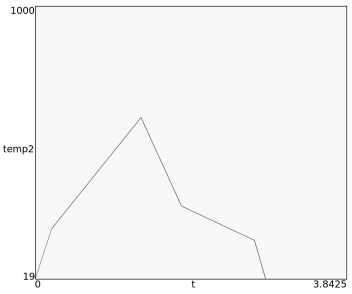

Figura 7: Grafico *hot_room2*- temperatura2 rispetto al tempo

Invece i seguenti sono i plot delle temperature nelle stanze e dell'apertura della valvola rispetto al tempo:

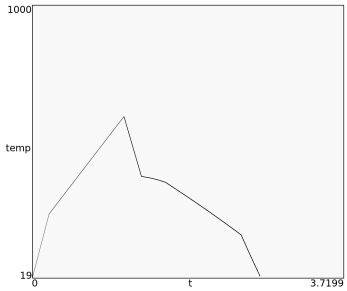

Figura 8: Grafico *hot_room*- temperatura rispetto al tempo con valvola

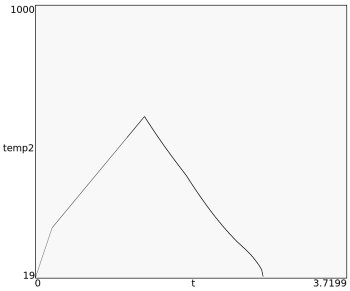

Figura 9: Grafico *hot_room2*- temperatura2 rispetto al tempo con valvola

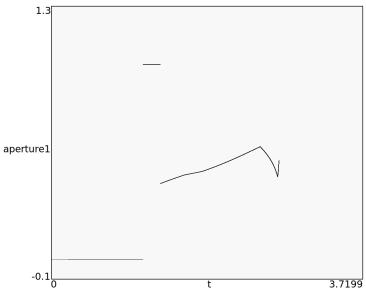

Figura 10: Grafico della *Aperture1* rispetto al tempo

V. Conclusioni

Questo progetto mi ha permesso di realizzare il mio primo sistema ibrido, precedentemente visto solamente nella teoria, mi ha insegnato a modellarlo, imparando a programmare con una nuova libreria C++, Ariadne. Grazie alla documentazione e ai codici di esempio forniti sono riuscita a capire come si progetta un sistema di questo tipo anche se ci sono state difficoltà con i codici di errore scritti in un linguaggio poco intuitivo perché segnalavano il problema nei file della libreria e non nel codice da me implementato.

VI. Sitografia

Link Libreria Ariadne:
Versione sperimentale libreria Ariadne
Link Progetto sistema Fire:
Progetto Chiara Zandonà