DASH: A New Language for Declarative Behavioural Requirements with Control State Hierarchy

UNIVERSITY OF WATERLOO

Jose Serna, Nancy A. Day, and Sabria Farheen
David R. Cheriton School of Computer Science
{jserna, nday, sfarheen}@uwaterloo.ca

The Language

- A **declarative** language for describing *formal behavioural models* of requirements
- The name comes from **D**eclarative **A**bstract **S**tate **H**ierarchy
- Adds hierarchical, labelled control states to the *Alloy* Language
- Supports user-defined and uninterpreted types and operations, and first order logic formulae in the conditions and actions of *state machines*
- Supports new ways of factoring, patterning, and layering abstractions to describe and systematically organize transitions of a model

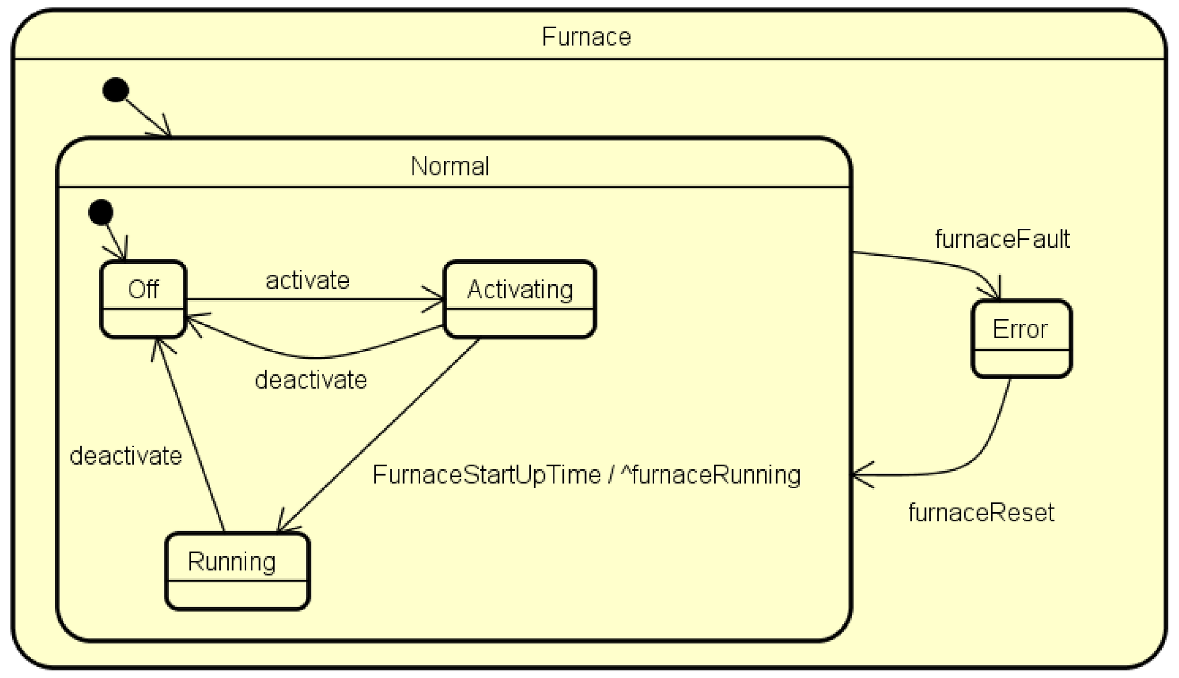
Transitions

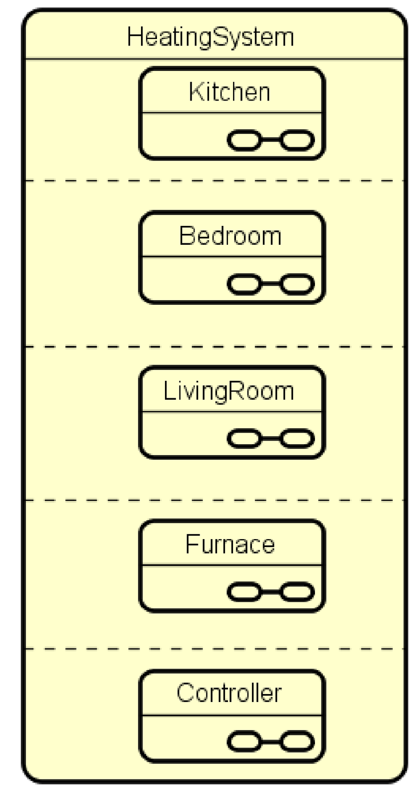
- Behavioural models are described using transition relations, DASH adds support for user-level abstractions and primitives to describe the transitions
- DASH has multiple ways of **factoring** transitions. They can be factored by states, events, actions and conditions

```
event deactivate {
    trans off1 {from Activating goto Off}
    trans off2 {from Running goto Off}
}
```

- **Patterning** defines a set of transitions in a single statement**.** In the `from` and `goto` parts of a transition, a list of state names can be provided. Additionally, * can be used to represent all states in the current scope
- **Layering** facilitates aspect-oriented modelling. Parts of transitions can be defined in different places, then the descriptions are merged together to create a complete description of the transitions

```
addon (do incErrorCounter)
    to (from * goto Error)
addon (do incErrorCounter) to t4
```






```
abstract sig ValvePosition {}
abstract sig Room {}
...
conc state HeatingSystem {
  valvePosition: Room -> ValvePosition
  desiredTemp: Room -> Int
  actualTemp: Room -> Int
  occupied: Room
  requestHeat: Room

  event activate {}
  event deactivate {}

  action adjValve [
    all r:occupied | r.actualTemp < r.desiredTemp =>
                     r.valvePosition' = OpenPosition
  ] {}

  condition roomsNeedHeat [
    some requestHeat
  ] {}

  init {
    all r: Room | r.valvePosition = ClosedPosition
  }

  conc state Controller {
    default state Off { }
    state Error { }
    state On {
      default state Idle{
        trans t1 { when roomsNeedHeat goto HeaterActive send activate }
        trans t2 { from HeaterActive when (not roomsNeedHeat) send deactivate }
      }
      state HeaterActive{
        default state ActivatingHeater{}
        state HeaterRunning{}
      }

      trans t3 { on heatSwitchOff goto Off send deactivate }
      trans t4 { on furnaceFault goto Error }
    }
  }

  conc state Bedroom {
    default state NoHeatRequestested {}
    state HeatRequested {
      default state  IdleHeating{}
      state WaitForCool{
        trans t5 { on waitedForCool do adjValve }
      }
    }
  }
}
```

State Hierarchy

- DAHS has direct support for control state hierarchy: **AND-**, **OR-** and basic states can be defined
- In each state, **declarations** of system elements can be defined using *Alloy* syntax
- DASH uses **primed** variables to refer to their values in the next state
- The state hierarchy is used as a **scoping mechanism** for creating partitioned **namespaces**
- The `init` and `default` keywords are used to define the initial state of the system and default states of the hierarchy
- **Actions** and **conditions** are expressed in *first order logic* including *quantifiers*

Semantics

- The definition and formalization of the semantics for DASH is work in progress
- A final set of transitions is obtained by flattening the effect of factoring, expanding the patterns, and combining layers to complete the definitions
- The meaning of a DASH model is determined from the final set of transitions that are combined to create a next state relation. Together, the next state relation and the predicates that determine initial conditions, form a symbolic *Kripke Structure*

Future Work

- Add more modularity to the language, including parameterized states and quantification over states
- Translate the models to Alloy and eventually to SMT solvers
- Explore model checking of DASH models