SENG441 Assignment 1

Jack Orchard, Sam Annand

Demo and Questions at end of presentation

Seven Segment Display Support

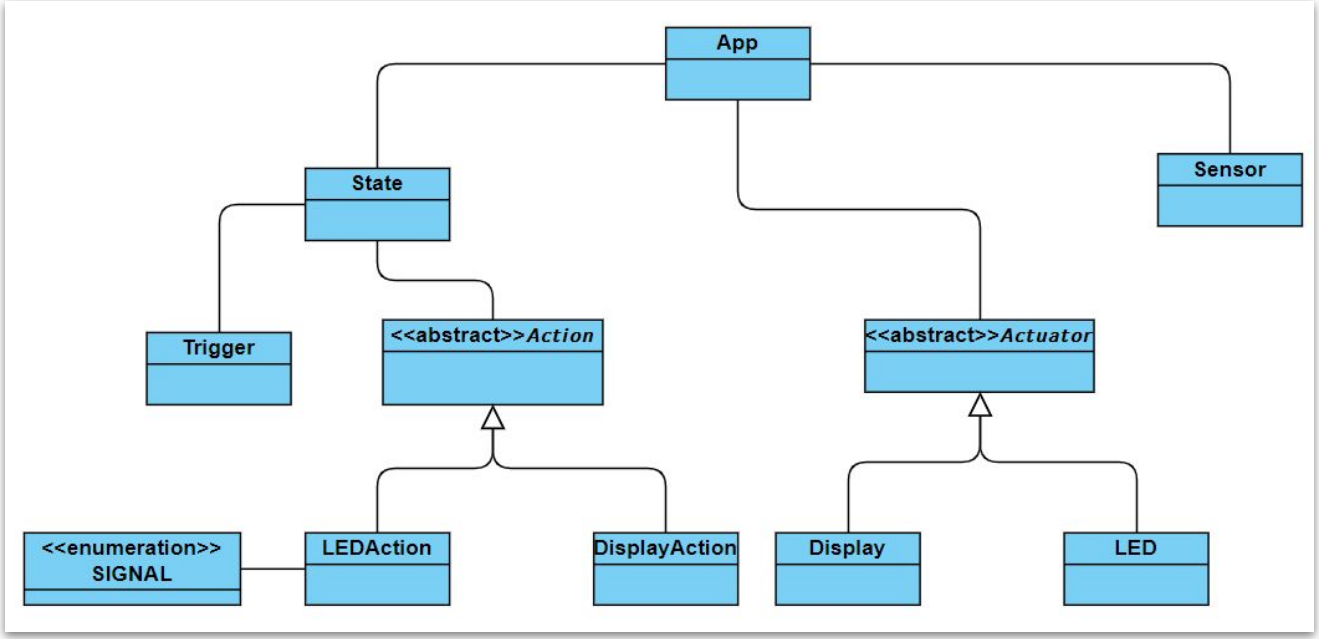
Process Followed

To implement support for the SevSeg in the ArduinoML we:

- Implemented the starter code from the lab
- Began trying to extend the existing Action and Actuator concepts to support the SevSeg
- Realised the limitations / drawbacks of trying to go too abstract so changed implementation (some early syntax analysis)
- Removed implementations from existing Action and Actuator concepts, and made them abstract, then implemented concrete child classes for SevSeg and SevSeg actions

Challenges Faced

- Difficult to plan out *how* we would implement, then came problems when we just starting hacking
- Trade-off with early design versus over-engineering and rigidity
- Difficult decision to abstract classes out and then use specific child classes
- MPS restrictions
- Using library based SevSeg implementation meant adding things in more than one place throughout generated text



UML Class Diagram - Language Structure

```
appliction <no name>

  no sensors defined
  <no name> :      initial: false
    no actions defined
    <no trigger>
    going to <no next>
```

```
appliction LED

  led: theLed on pin 13
  sensor: theButton on pin 10
  on :      initial: true
    theLed <= high
    theButton is low
    going to off

  off :      initial: false
    theLed <= low
    theButton is low
    going to on
```

```
appliction SevSeg

  Display sevseg
    A  1
    B  2
    C  3
    D  4
    E  5
    F  6
    G  7
    D1  8
    D2 9
  sensor: theButton on pin 10
  zero :      initial: true
    sevseg <= 0
    theButton is low
    going to one

  one :      initial: false
    sevseg <= 1
    theButton is low
    going to zero
```

Initial Syntax

Design justification

Abstract model with specific child implementations:

- More potential to take full advantage of library
- Easy to understand, e.g. user of language sees “ahh, okay, in this section I can add either a display, or an LED, and in this section a button”
- Open to expansion in future
- Learnt from implementing translation language in Java that generic implementations put much more work onto the user

Further syntax analysis

Syntax Analysis - First Model Positives

- The language is simple because it has few concepts
- The way an led and sensor is defined is very readable, similar to pseudocode
- The language is familiar to an arduino developer, for example setting an led to “low”

```
appliction LED

  led: theLed on pin 13
  sensor: theButton on pin 10
  on :       initial: true
    theLed <= high
    theButton is low
    going to off

  off :      initial: false
    theLed <= low
    theButton is low
    going to on
```

Syntax Analysis - First Model Negatives

- For actions and triggers to be readable, the user has to choose a readable name
- The symbols and constants used do not clearly afford the functions that they intend, they could read more clearly with more information.
- Each state has to be set as initial or not initial in every declaration, even though there is only allowed to be one initial state
- Inconsistent spacing affecting logical separation and readability.
- The models are not readable without previous experience writing in the language e.g what does “the button is low” mean.

```
appliction SevSeg

  Display sevseg
    A  1
    B  2
    C  3
    D  4
    E  5
    F  6
    G  7
    D1  8
    D2 9
  sensor: theButton on pin 10
  zero :       initial: true
    sevseg <= 0
    theButton is low
    going to one

  one :       initial: false
    sevseg <= 1
    theButton is low
    going to zero
```

Syntax Analysis - First Model Tradeoffs

- Overall the language is very restrictive because the user only has display and LED support. Users aren't able to define their own execution logic or types.

  However,

- Allowing user defined execution logic and types makes the language very similar to directly writing arduino C code, which begs the question, why would you not just write arduino C code at that point

Implementing new syntax

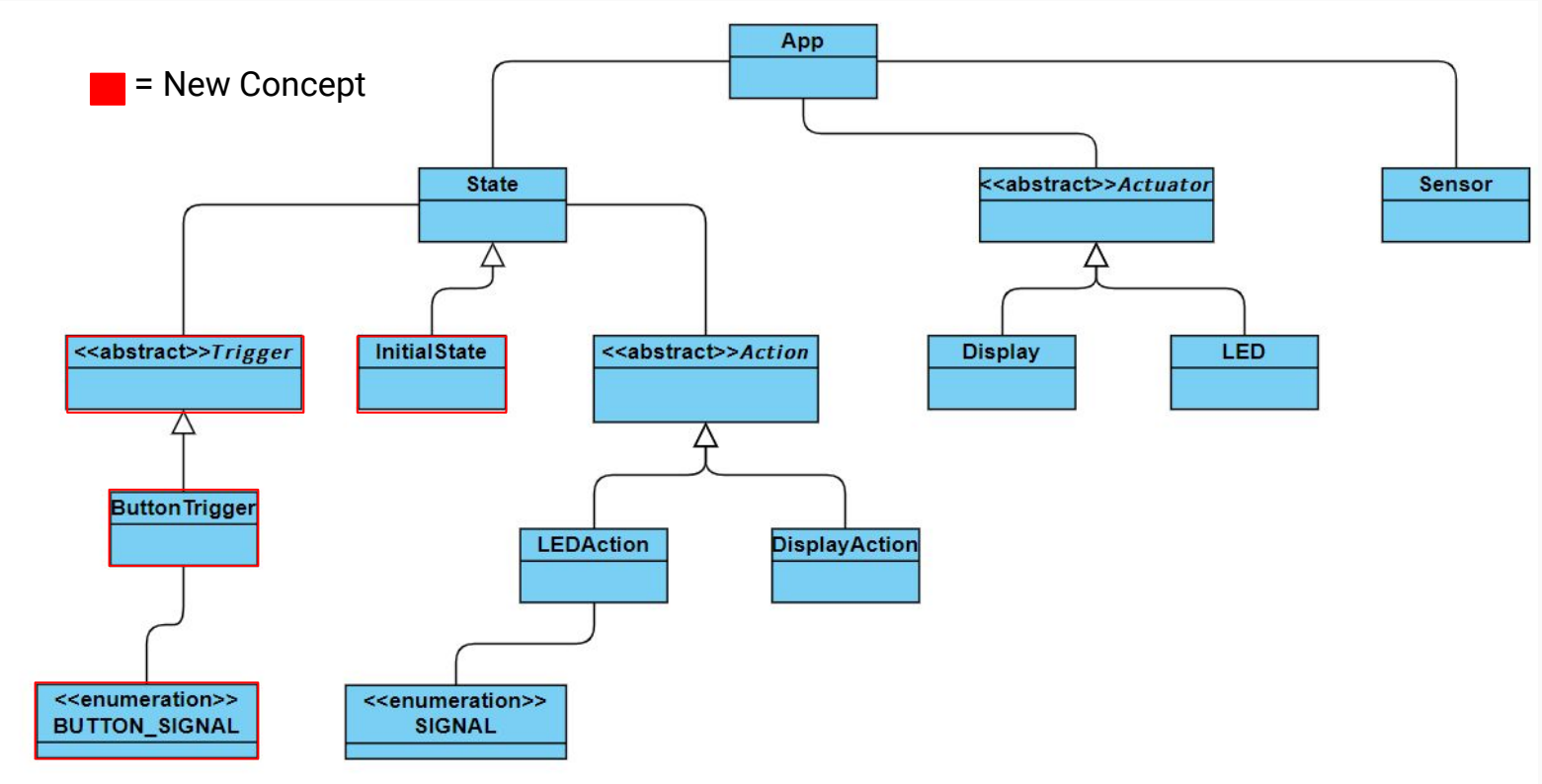
```
application <no name>

  actuators:

    no actuators defined

  sensors:

    no sensors defined

  states:

    initial state <no name>
      no actions defined
      transition to state <no next> <no trigger>

    no states defined
```

```
application LED

  actuators:

    led theLed on pin 13

  sensors:

    button theButton on pin 10

  states:

    initial state on
      turn on led theLed
      transition to state off if button theButton is pressed

    state off
      turn off led theLed
      transition to state on if button theButton is pressed
```

```
application SevSeg

  actuators:

    1 digit display sevseg
      A  1
      B  2
      C  3
      D  4
      E  5
      F  6
      G  7
      D1 8
      D2 9

  sensors:

    button theButton on pin 10

  states:

    initial state zero
      show 0 on display sevseg
      transition to state one if button theButton is pressed

    state one
      show 1 on display sevseg
      transition to state zero if button theButton is pressed
```

New Syntax

Implementation of New Syntax



Design Choices

- Initial State is its own concept so that it is enforced as a required state and is programmed separately by the user. This means normal states do not have to concern themselves with being initial.
- Make Trigger abstract to separate the syntax from behaviour with the app, this allows the button trigger to be more readable

Rationale for New Syntax

- For actions and triggers to be readable, the user has to choose a name that fits the surrounding syntax and,
- The symbols and constants used do not clearly afford the functions that they intend, they could read more clearly with more information.

Fixes:

- Use pseudocode style syntax to explain user actions in english

- The lack of spacing means that sections of the model are not logically separated.

- Introduce headed sections with colons

- Each state has to be set as initial or not initial in every declaration

- Introduce new concept InitialState, separate to other states

Current Drawbacks

- Remains cumbersome to declare many repeated states
- Doesn’t capture restraints and reservations of underlying C language
- Assuming the use of one Display max, even though more can be created
- No constraints on numbers to show on display, even though only 0-99 supported
- Does not support advanced logic, such as multiple triggers and transitions from one state

Extra features and future enhancements

Extra features we did add

- Early on added trigger and sensor support to allow the use of a button

Extra features we would add in future

- More concrete implementations for basic sensors and actuators
- Counter and variable support
- Hard constraints on typesystem
- Concept of transitions, and being able to have multiple different possible future states
- Ability to return to previous state

MPS Insights

Cons

- Restricted ability to type
- Had to ‘retype’ things for MPS to realise updates
- Difficult to pair program remotely as not just free text, also no copy paste
- Software in relatively early stage of life
- Buggy, often had to invalidate caches and restart
- When making the project, sometimes you think it made the project, but actually it didn’t because of some error but didn’t tell you
- Very challenging to sync work over git, often had to reclone when switching pair programming ‘drivers’
- Due to file structure being different to how the structure appears in MPS (it parses files and presents them differently)
- Inheritance is not intuitive

Pros

- Restricted typing reduces chance of input error
- Project structure and linking between concepts, editors and textgen made it intuitive to track flow from models to generated code
- Familiar editor style as have plenty of IntelliJ experience
- Applications models updating automatically, big ++
- Syntax is forced in application models

Thanks!

Now for a quick demo and then we will take questions