Design and Implementation of a C Code Generator Module for the Gamma Statechart Composition Framework

Nagy Levente Márk

Advisor: Graics Bence

Budapest University of Technology and Economics
Department of Measurement and Information Systems
ftsrg Research Group

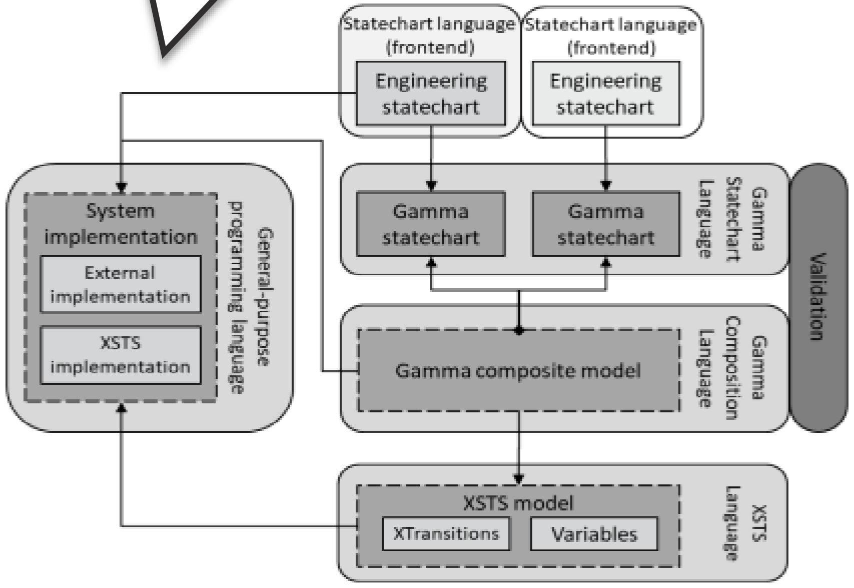
Fancy title, but..

- What is the Gamma Statechart Composition Framework?
- Why do we need Code Generators?
- From where do we generate code?
- What technologies are being used?
- How do we achieve platform independence?
  - Especially in timing
- How do we utilize the generated code?

The Gamma Framework

- Eclipse extension
- Composite Statecharts
  - From statecharts & interfaces designed in Yakindu
- Formal Verification of composite systems
  - Through a model-checker, UPPAAL, Theta, Spin
- Code Generation for various platforms
  - Gamma contains a code generator for java
- Verification of the generated code
  - Unit tests based on a model-checker should cover the state space

C, Java implementations



Code Generators

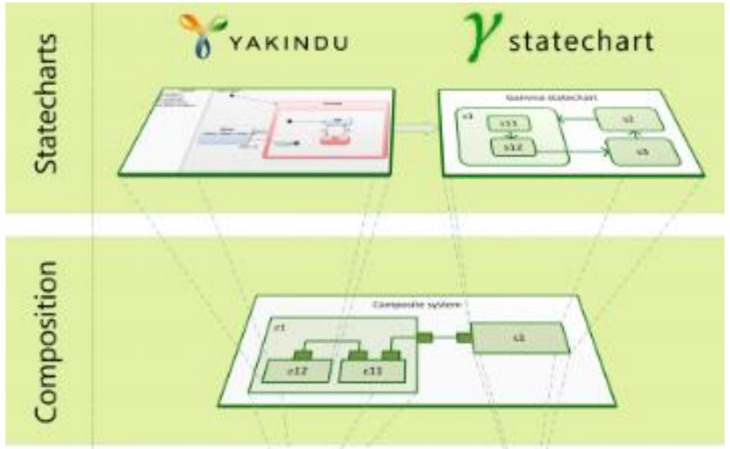
Why do we need?

- Efficiency - no need for manual coding
- Accuracy - reduce the risk of human error
- Consistency - enforcing model-code behavior consistency
- Maintainability - code maintenance by regenerating code
- Portability - can generate code for different platforms

Potential presence of bugs in code generators, could we verify the results?

Technologies used

What technologies?



- XSTS language
  - Intermediate representation of our model
- Eclipse
  - Gamma is an eclipse plugin
  - The XSTS language is implemented using EMF, serialized to XML
  - Dependencies integrated into Eclipse: Yakindu, PlantUML, etc..
- Xtend
  - Java dialect, java code is being generated in the background
  - Commonly used in code generators, serializers instead of instanceof

The XSTS Language

XSTS definitions

What technologies?

- Declarations
  - Type, Variable declarations
  - Variable groups, annotations, e.g.: input groups, clock variables

```
type Main_region_Controller : { __Inactive__, Operating, Interrupted }
var PoliceInterrupt_police_In_controller : boolean = false
var BlinkingYellowTimeout3_secondary : integer = 0
```

- Initializations
  - Initial values, initialize the component
  - Reset inputs, outputs between cycles
- Transitions - the internal mechanism of the model
- Consists of..

  Actions, Expressions

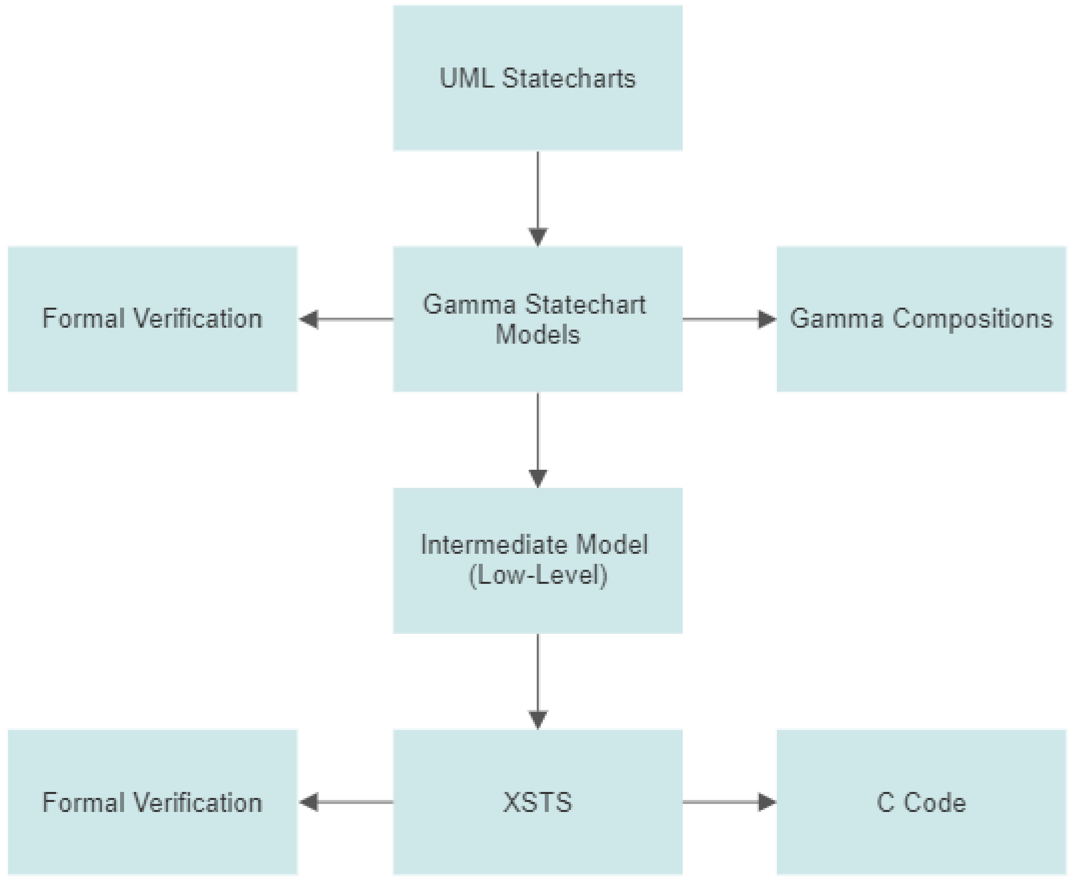
XSTS serialized to XML

```
<typeDeclarations name="Main_region_Controller">
   <type xsi:type="hu.bme.mit.gamma.expression:EnumerationTypeDefinition">
      <literals name="__Inactive__"/>
      <literals name="Operating"/>
      <literals name="Interrupted"/>
   </type>
</typeDeclarations>
```

Model Transformation

From where?

- UML Statecharts
  - Transformed to *gamma statechart language*
- Gamma Components
  - Transformed into XSTS with an extra intermediate step
- XSTS Model
  - Used to generate C code
  - Can be used to formally verify the component



Implementation

What technologies?

- For each component..
  - 2 source & 2 header files are being generated
- Statechart component
  - Implements the behavior of its model
  - Resets in-, outputs between cycles
  - Declares all required structure
- Wrapper component
  - Handles timing
  - Hides the internal mechanism
  - Provides Ports (in the form of setters/getters)

Regions as enums

```
enum Main_Region_Controller {
    __Inactive___main_region_controller,
    Operating_main_region_controller,
    Interrupted_main_region_controller
} main_region_controller;
```

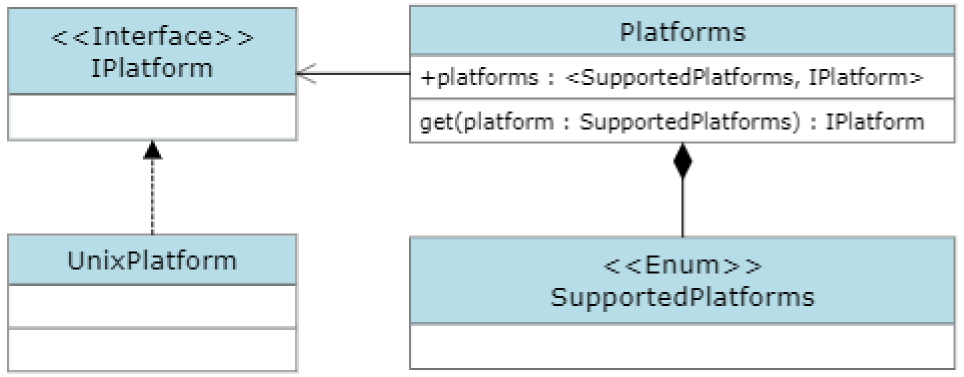
```
typedef struct {
    bool PoliceInterrupt_police_In_controller;
    enum Main_Region_Controller main_region_controller;
    enum Operating_Controller operating_controller;
    /* ... */
    bool __id_Green_9_Yellow__secondary;
    bool __id_Normal_18_Interrupted__secondary;
    unsigned int BlinkingYellowTimeout3_secondary;
} CrossroadStatechart;
```

Components as structs

Platform independence

How do we achieve?

- ISO Standard - standards for the C programming language
  - Extensive support from compilers
- Timing - measure time elapsed
  - Little-to-no overlap between platforms



<<Interface>>
IPlatform
Platforms
+platforms : <SupportedPlatforms, IPlatform>
get(platform : SupportedPlatforms) : IPlatform
UnixPlatform
<<Enum>>
SupportedPlatforms

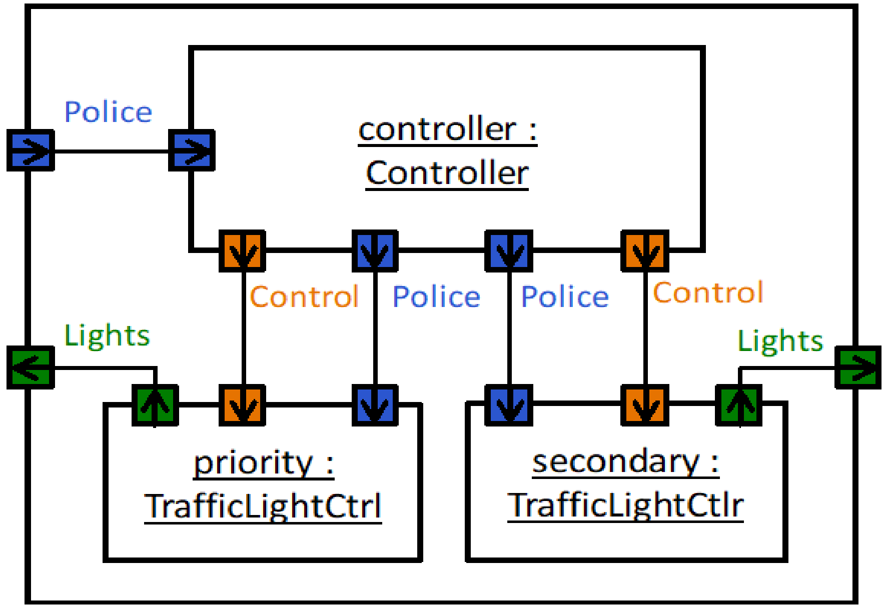
Solution

Introduce a **package** for customized platform & timing handling

- Similar structure between platforms
- One class for each platform, contains the implementation
- Central platform manager

Case Study

The Crossroad Example



Utilization

```
CrossroadWrapper statechart;
initializeCrossroadWrapper(&statechart);
```

Initialize statechart

- Initialization
  - Internal variables
- Ports
  - Setters for input ports
  - Getters for output ports
  - Fallback to a default value after each cycle

Ports as setters & getters

```
/* setters as input ports */
setPoliceInterrupt_police_In_controller(&statechart, detect(POLICE));
/* getters as output ports */
getLightCommands_displayRed_Out_prior(&statechart);
getLightCommands_displayRed_Out_secondary(&statechart);
```

- Timing
  - At least 500 us of sleep required
  - Variables used for timing can overflow

Run cycles, sleep

```
runCycleCrossroadWrapper(&statechart);
usleep(2500);
```

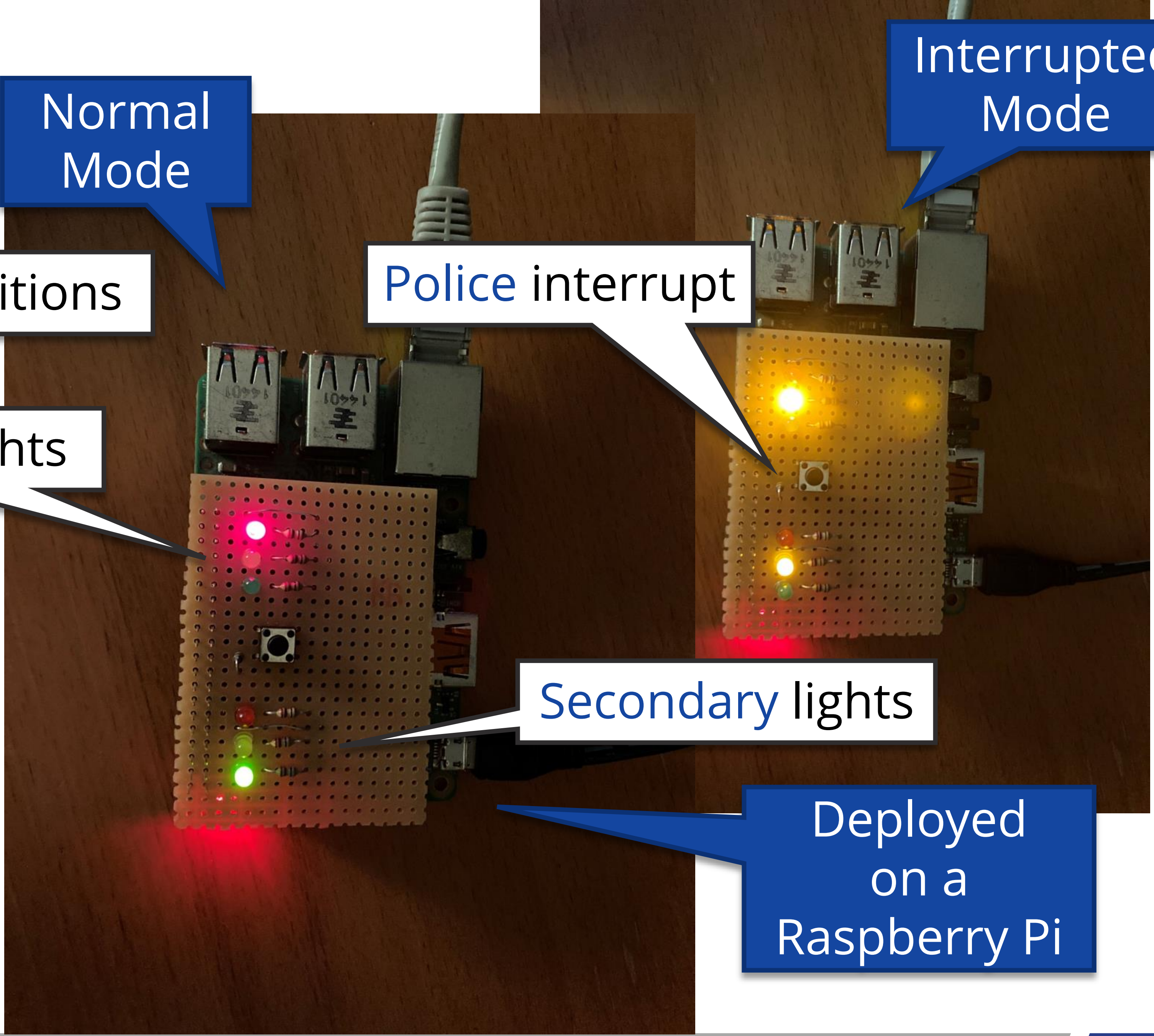
- Cycle - check whether a change in state is due

Results

```
#define PRIMARY_RED    8
#define PRIMARY_YELLOW 7
#define PRIMARY_GREEN  0

#define SECONDARY_RED    21
#define SECONDARY_YELLOW 23
#define SECONDARY_GREEN  25
```

- Environment
  - GPIO pins
  - Could be something else
- Compiled without
  - Errors, warnings
- Timing for UNIX platforms
  - Works as intended



Summary

- C Code Generation from XSTS Models
  - Efficiency, accuracy, consistency, etc..
  - Ensure platform independent code
- Case Study: The Crossroad Example

Consistency: the generated code will reflect the behavior of its models

Generation

Generate **accurate** and **consistent** code from gamma statecharts and composite models

Accuracy: fewer logical mistakes during coding

- Future work
  - Support more platforms
  - Formal verification of the generated code using model-checkers