Modelling and Transformation of Non-Functional Annotations

Jorge Ibarra Delgado
School of Computer Science
University of St. Andrews

June 24, 2015
Plan

1. Context
   - Introduction
   - Objectives

2. PlantNF

3. PlantNF-to-UPPAAL Transformation Tool
Modelling

- An abstract description of an artefact
- Key elements of a new or existing system
- Different stakeholders can verify and evaluate requirements
- helps enabling the reuse of components
Modelling and Transformation of Non-Functional Annotations

Context

Introduction

Modelling

Informal Methods

- Easy to learn and apply
- Too much expressiveness
- UML, boxes connected by lines
Modelling

Informal Methods

- How the customer explained it
- How the project leader understood it
- How the engineer designed it
- How the programmer wrote it
- How the sales executive described it
- How the project was documented
- What operations installed
- How the customer was billed
- How the helpdesk supported it
- What the customer really needed
Modelling and Transformation of Non-Functional Annotations

Context

Introduction

Modelling

Formal Methods

- Requires more intellectual effort
- Construction by applying semantical rules
- Event-B, PROMELA, UPPAAL, MoDeST
Modelling

Formal Methods

We have no idea what you're talking about.

Trust me, it's science.
Model Transformation

- Used to generate models of different kind
- Use of a target model which can be formally verified
- Examples of target models are Petri nets and timed automata
The Project

The problem:

- Specify a choice of non-functional properties of a system as model annotations
- Implement a transformation between the annotated model and a formal model
The Project

The problem:

- Specify a choice of non-functional properties of a system as model annotations
- Implement a transformation between the annotated model and a formal model

Our approach:

- Explore the use of different languages and modelling tools
- Create an annotation model based on observations
- Implement a transformation tool between the created language and a formal language
Plan

1. Context

2. PlantNF

3. PlantNF-to-UPPAAL Transformation Tool
PlantUML

A language that allows to write UML diagrams

Graphical representation of

- Sequence diagrams
- Use Case diagrams
- Class diagrams
- Activity diagrams
- Component diagrams
- State diagrams
- Object diagrams
@startuml

Closed : The door is closed & is not in use

[*] --> Closed
Closed --> Heating : set timer and press start
Heating --> Closed : finish heating or press stop
Closed --> Opened : press open
Opened --> Closed : close door

@enduml
Modelling and Transformation of Non-Functional Annotations

PlantUML

- Closed
  - The door is closed & is not in use
  - Transition to Heating
  - Transition to Opened

- Heating
  - set timer and press start

- Opened
  - finish heating or press stop
  - press open
  - close door
A tool suite for simulation and automatic verification of real-time systems

Systems are modelled as timed automata
Modelling and Transformation of Non-Functional Annotations

UPPAAL

[Diagram of a UPPAAL model with a state machine and labels such as "idle", "pushed!", "\text{!activated}", and \( w=0 \).]
MoDeST

The Modelling and Description Language for Stochastic Timed Systems

A formal language that aims to support:

- The modular description of reactive systems’ behaviour while covering
- Functional system aspects
- Non-functional system aspects
Modelling and Transformation of Non-Functional Annotations

MoDeST

\[ P ::= \text{stop} \mid \text{abort} \mid \text{break} \mid \text{act} \mid \text{when}(b) \ P \mid \text{urgent}(b) \ P \mid P_1; P_2 \mid \text{alt}\{::P_1 \ldots ::P_k\} \mid \text{do}\{::P_1 \ldots ::P_k\} \mid \text{par}\{::P_1 \ldots ::P_k\} \mid \text{act palt}\ \{::w_1::\text{asgn}_1; \ldots ::w_k::\text{asgn}_k; \ P_k\} \mid \text{ProcName}(e_1, \ldots, e_k) \mid \text{throw}(\text{excp}) \mid \text{try}\{P\} \text{ catch } \text{excp}_1 \ {\{P_1\} \ldots \text{ catch } \text{excp}_k \ {\{P_k\}} \mid \text{relabel } \{I\} \text{ by } \{G\} \ P \mid \text{extend } \{H\} \ P \]
PlantNF

Language that provides:

- Ease of use, similar to PlantUML
- Capable to describe elements of timed automata
- Grammar combined with constructs from UPPAAL
- Incorporates elements of probability
The Grammar

Document ::= Declaration Template+ SystemExt

Declaration ::= 'Declarations' Declarations 'EndDeclarations'

Template ::= 'Template' ID ParameterExt? Declaration? State+ BranchPoint* Transition+ EndTemplate'

SystemExt ::= 'System' System 'EndSystem'
The Grammar

State ::= 'ini'? ('com' | 'urg')? 'loc' ID (';' ID)? Invariant? (':' ExponentialRate)? ';

BranchPoint ::= 'branch' ID ';

Transition ::= ID -> ID (';' Label (',' Label)*)? ';

The Grammar

Label ::= 'select :' SelectList
| 'guard :' Expression
| 'sync :' Expression ('!' | '?')
| 'assign :' Expression (',' Expression)*
| 'comments :' COMMENT
| 'probability :' Expression
Example

Template Soldier
Parameters const int delay
EndParameters

Declarations

clock y ; EndDeclarations
loc id0 ;
loc id1 , safe ;
loc id2 ;
ini loc id3 , unsafe ;
Example

id2 -> id3 :
guard : y >= delay ,
sync : release ! ;
id1 -> id2 :
guard : L == 1 , sync : take! ,
assign : y=0 ;
id0 -> id1 :
guard : y >= delay , sync : release ! ;
id3 -> id0 :
guard : L == 0 ,
sync : take! , assign : y=0 ;
EndTemplate
PlantNF Editor Tool

- Provides support to write files in our language
- Created using xtext plug-in
- Gives the possibility to work with the benefits of an eclipse environment
PlantNF Editor Tool
Plan

1. Context

2. PlantNF

3. PlantNF-to-UPPAAL Transformation Tool
The Tool

- Uses a PlantNF file as input
- Identify the elements that are part of the UPPAAL language meta-model
- Outputs an XML file that UPPAAL can read
Modelling and Transformation of Non-Functional Annotations

PlantNF-to-UPPAAL Transformation Tool

Language Metamodel
Architecture of the system

```
application
command

analyzer

util
language
```
Result From Past PlantNF Example

```xml
<template>
  <name>Soldier</name> <parameter>
    const int delay </parameter>
  <declaration> clock y ;
  </declaration>
  <location id="id0"/>
  <location id="id1"> <name>safe</name>
  </location> <location id="id2 />
  <location id="id3"> <name>unsafe</name>
  </location>
  <init ref="id3"/>
</template>
```
Result From Past PlantNF Example

```xml
<transition>
<source ref="id2"/> <target ref="id3"/>
<label kind="guard">y &gt;= delay</label>
<label kind="synchronisation">release!</label>
</transition> <transition>
<source ref="id1"/> <target ref="id2"/>
<label kind="guard">L == 1</label>
<label kind="synchronisation">take!</label>
<label kind="assignment">y = 0</label>
</transition> <transition>
<source ref="id0"/>
<target ref="id1"/>
<label kind="guard">y &gt;= delay</label>
<label kind="synchronisation">release!</label>
</transition>
```
Result From Past PlantNF Example
Future work

- Work on a graphic representation of our language
- Target other languages and tools
- Improve the grammar implemented in the PlantNF editor
- Give room to implement UPPAAL model checking queries
References


- Henrik Bohnenkamp *MODEST: A compositional modeling formalism for hard and softly timed systems* IEEE TRANSACTIONS ON SOFTWARE ENGINEERING

- Bogdoll, Jonathan and David *Mctau: Bridging the Gap Between Modest and UPPAAL* 2012

- Bettini, Lorenzo *A DSL for Writing Type Systems for Xtext Languages* 2011