MultiMedia C# (MMC#)

QoS-Aware Programming with C#

Motivation

- Handle Quality of Service (QoS) and adaptive behavior within the common programming language C#.
  - Constraint Definition
  - Constraint Monitoring
  - Adaptivity

Motivation

- Temporal Constraints

QoS Constraint	QL Syntax
Throughput	$\forall n, \lvert\tau(\epsilon_r, n) - \tau(\epsilon_r, n-k)\rvert \leqslant \delta$
Latency	$\forall n, \lvert\tau(\epsilon_r, n) - \tau(\epsilon_r, n-1)\rvert \leqslant \delta$
Jitter	$\forall n, \delta_1 \leqslant \lvert\tau(\epsilon_r, n) - \tau(\epsilon_r, n-k)\rvert \leqslant \delta_2$
Bounded Execution Time	$\forall n, \lvert\tau(\epsilon_i, n) - \tau(\epsilon_o, n)\rvert \leqslant \delta$

- Non-temporal Constraints

$$(width \geqslant \lambda_{minWidth} \wedge height \geqslant \lambda_{minWidth})$$

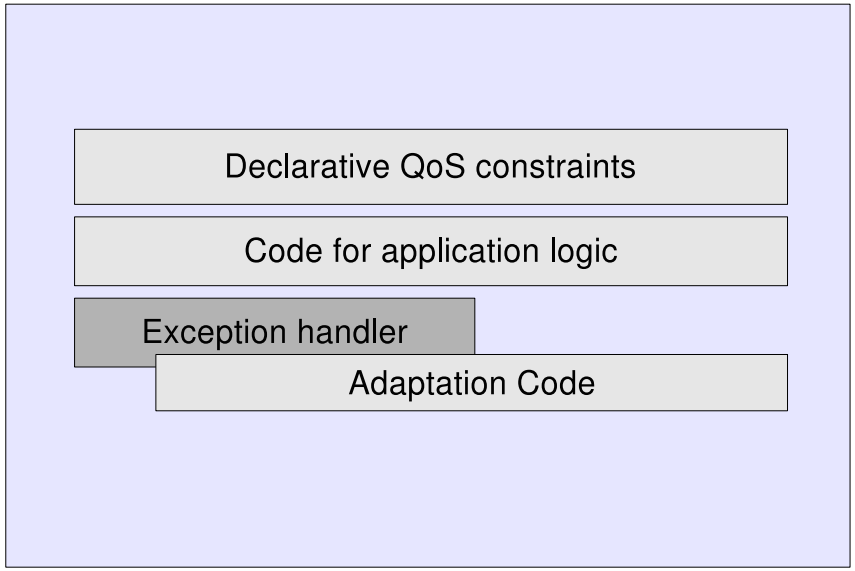
$$\wedge \quad (width \leqslant \lambda_{maxWidth} \wedge height \leqslant \lambda_{maxWidth})$$

Motivation

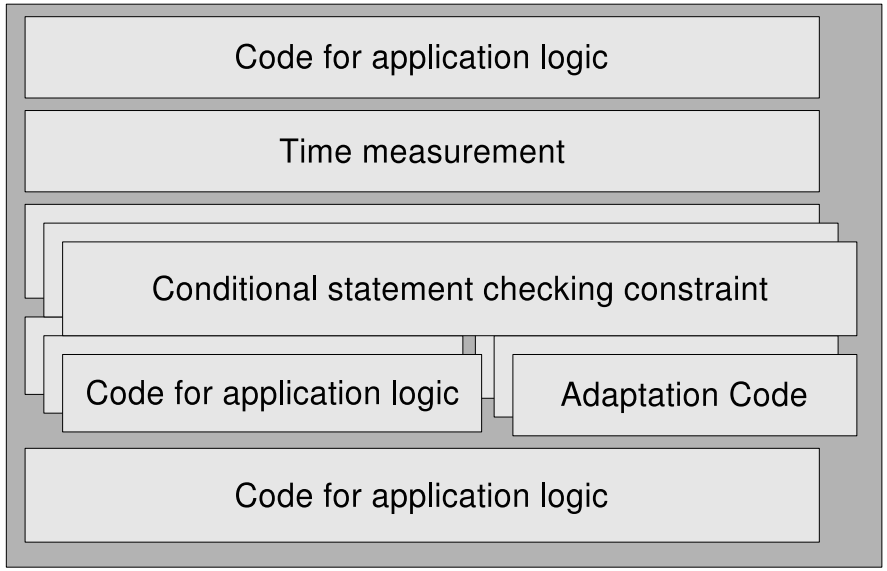
```
int n = 0;
long oneSecond = 1*1000*1000*10; // in 100-nanosec
DateTime[] timeStamps = new DateTime[26];

while (DataAvailable()) {
  timeStamps[n % 25] = DateTime.Now;

  Frame inData = FetchData();

  if (n >= 25
   && timeStamps[(n % 25)].ToFileTime()
      - timeStamps[(n + 1) % 25].ToFileTime() > oneSecond) {
    // constraint violated, do some adaptation
  } else {
    // process received data
  }
  n++;
}
```

Motivation

Adaptive Quality-Aware Programming Model



Common Model for Adaptive Programming



MMC# - Overview

- Constraint Definition
  - Representation of time
  - Declarative syntax for QoS constraint definition
  - Semantic analysis to provide correctness and utilize optimization potential

- Timed data types – Time as n+1st dimension

```
int [~] dynamicQoSVariable
    = new int[~(IQoSObject)constraint];
int [~constraint] staticQoSVariable;
```

MMC# - Overview

- Declarative Constraint Definition

```
private constraint FrameRateAndJitter
    (object[~] i, int rate, int jitter
   , int timeRange) =>
       @n{i[n] - i[n-rate]
           < QoS.Units.MSec(timeRange)}
    && @n{i[n] - i[n-1]
           > rate/QoS.Units.MSec(timeRange)
                 -QoS.Units.MSec(jitter)
    &&    i[n] - i[n-1]
           < rate/QoS.Units.MSec(timeRange)
                 +QoS.Units.MSec(jitter)};
```

- Attach Constraint to timed variable

```
object[~FrameRateAndJitter(o, 25, 10, 1000)] o;
```

MMC# - Overview

- Constraint Monitoring
  - Implicit constraint monitoring with the help of a new value assignment operation
  - Compiler-supported, automatic exception generation in case of QoS violations

- Timed assignment operation (~: :~)

```
int[~] value;
value ~: someValue;
```

MMC# - Overview

- Adaptivity
  - Place adaptation code inside exception handler

```
int [~@n {value[n] - value[n-1]
          <= Units.MSec(10)}] value;

try {
  // Timed writing; Left-hand must be a timed variable.
  value ~: someValue;
  Console.WriteLine("Value: " + value);
} catch (QoSException e) {
  Console.WriteLine("QoS Violation!");
}
```

MMC# - Overview

```
// constraint declaration
// timed data type with constraint definition
Frame[~@n{inData[n] - inData[n-25] < Units.Sec(1)}]
    inData;
while (DataAvailable()) {
  try {
    inData ~: FetchData(); // constraint monitoring
    // process received data
  } catch (QoSException) { // exception caused in case of
    constraint violation
    // constraint violated, do some adaptation
  }
}
```

Semantic Constraint Analysis

- Detect inconsistent constraint definition
- Optimize constraint representation

1. Split constraint definition into conditional blocks.
2. Convert condition blocks into a normalized form.
3. Compare each normalized constraint condition to each other and fill evaluation matrix.
4. Decide constraint reconstruction based on the evaluation matrix to provide optimized code or warning and error messages.

Semantic Constraint Analysis

- Constraint declaration

```
@n{frame[n] - frame[n-25] < QoS.Units.Sec(1)} &&
@n{frame[n] - frame[n-1] < QoS.Units.MSec(100)} &&
@n{   frame[n] - frame[n-1] > QoS.Units.MSec(20)
   && frame[n] < frame[n-1] + QoS.Units.MSec(60)} &&
@n{frame[n] - frame[n-5] < QoS.Units.MSec(500)}
```

- Constraint condition normalization

Part	Condition
C_1	frame[n] - frame[n-25] < QoS.Units.Sec(1)
C_2	frame[n] - frame[n-1] < QoS.Units.MSec(100)
C_{3a}	frame[n] - frame[n-1] > QoS.Units.MSec(20)
C_{3b}	frame[n] - frame[n-1] < QoS.Units.MSec(60)
C_4	frame[n] - frame[n-5] < QoS.Units.MSec(500)

Semantic Constraint Analysis

- Constraint condition comparison

$$R_{A\rightarrow B} = \begin{cases} E, & \text{if } \delta_A = \delta_B \wedge k_A = k_B \\ W, & \text{if } op_A = op_B = \begin{cases} <, & \text{if } \frac{\delta_A}{k_A} - \frac{\delta_B}{k_B} \geqslant 0 \wedge k_A \geqslant k_B \\ =, & \text{if } \frac{\delta_A}{k_A} > \frac{\delta_B}{k_B} \\ >, & \text{if } \frac{\delta_A}{k_A} - \frac{\delta_B}{k_B} < 0 \wedge k_A < k_B \end{cases} \\ S, & \text{if } op_A = op_B = \begin{cases} <, & \text{if } \frac{\delta_A}{k_A} - \frac{\delta_B}{k_B} \leqslant 0 \wedge k_A \leqslant k_B \\ =, & \text{if } \frac{\delta_A}{k_A} < \frac{\delta_B}{k_B} \\ >, & \text{if } \frac{\delta_A}{k_A} - \frac{\delta_B}{k_B} > 0 \wedge k_A > k_B \end{cases} \\ D, & \text{if } \delta_A \vee \delta_B \vee k_A \vee k_B \text{ are dynamic} \\ I, & \text{if } op_A \neq op_B \wedge \begin{cases} <>, & \text{if } \frac{\delta_A}{k_A} - \frac{\delta_B}{k_B} < 0 \\ ><, & \text{if } \frac{\delta_A}{k_A} - \frac{\delta_B}{k_B} > 0 \end{cases} \\ -, & \text{else} \end{cases}$$

Semantic Constraint Analysis

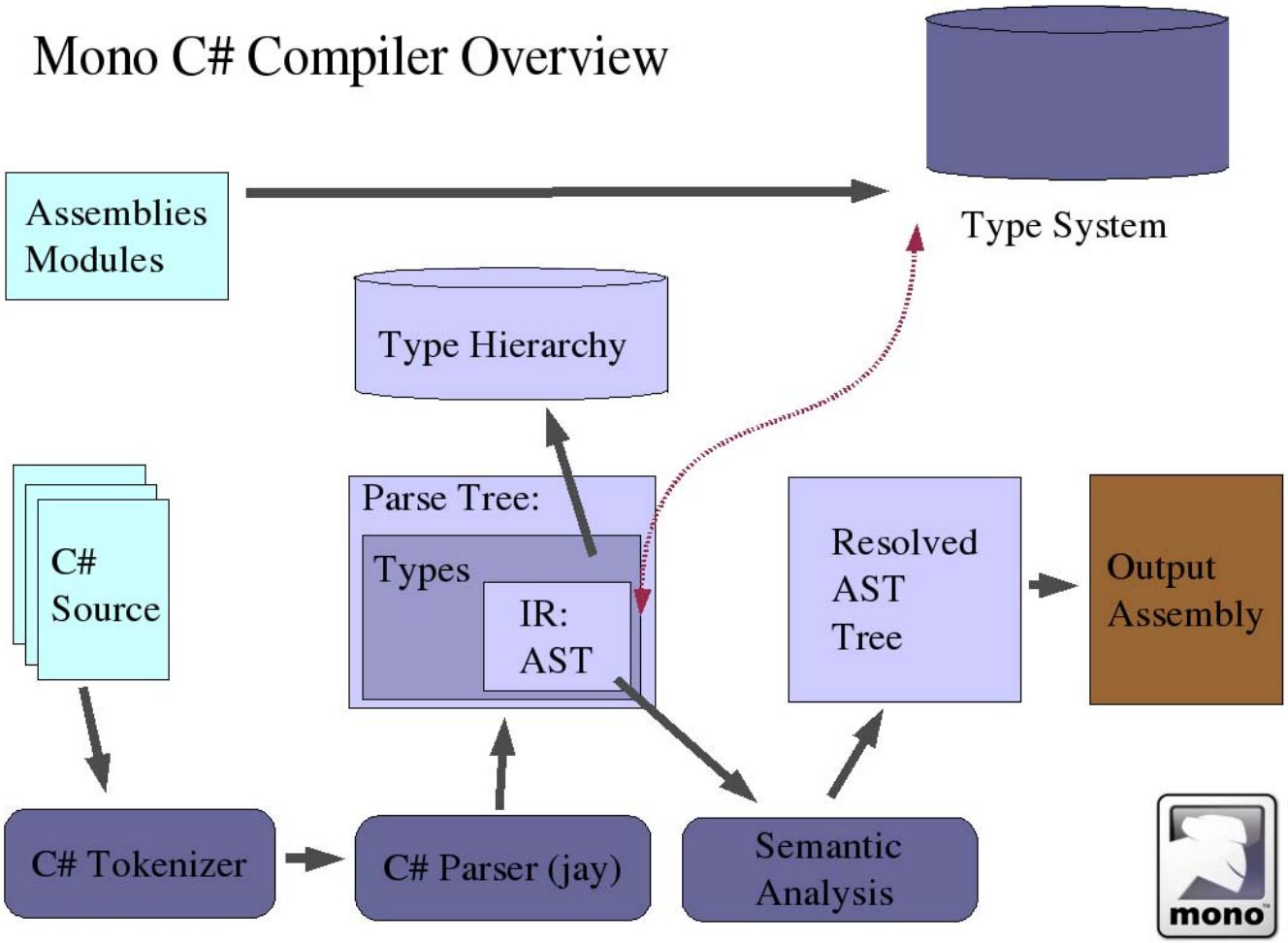
- Constraint reconstruction

```
@n{frame[n] - frame[n-25] < QoS.Units.Sec(1)}
   &&
@n{frame[n] - frame[n-1] > QoS.Units.MSec(20)
        && frame[n] < frame[n-1] + QoS.Units.MSec(60)}
```

Mono Project

- Common Language Infrastructure
  - CLI implementation containing a class-loader, JIT (just-in-time) compiler and GC (garbage collector) infrastructure.
- Class Library
  - The provided class library is .NET compatible and also includes Mono-provided classes.
- C# Compiler
  - The C# compiler is the main compiler of the Mono framework. There is already support for other languages like Visual Basic, Oberon or Object Pascal.

Mono C# Compiler



Mono C# Compiler Implementations

- **mcs** - targeting .NET 1.x framework.
- **gmcs** - targeting .NET 2.0 and 3.5 framework.
- **smcs** - targeting .NET 2.1 framework including Silverlight APIs.
- **dmcs** - a C# 4.0 compatible compiler (currently under development).

Summary

- Approach for handling QoS directly within C#
- Semantic analysis of temporal QoS constraints
- Short Overview about the Mono Project

Thank you for your attention!

Any questions?