



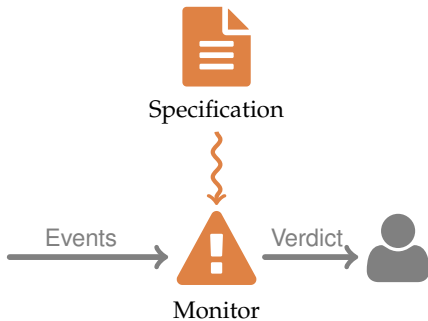
TeSSLa – An Ecosystem For Runtime Verification

Hannes Kallwies¹ **Martin Leucker**¹ **Malte Schmitz**¹ **Albert Schulz**²
Daniel Thoma¹ **Alexander Weiss**²

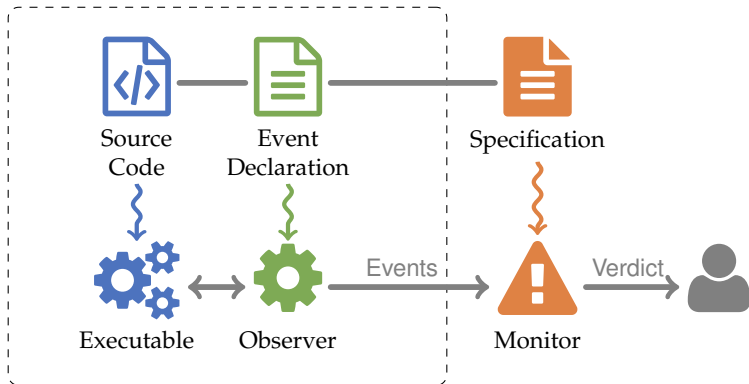
¹ Institute for Software Engineering and Programming Languages,
University of Lübeck, Lübeck, Germany

² Accemic Technologies GmbH, Kiefersfelden, Germany

22nd International Conference on Runtime Verification, September 2022



Runtime Verification Process



TeSSLa language features



- ▶ **Stream-based** specification language with **functional part**

- ▶ **Stream-based** specification language with **functional part**
- ▶ Based on six basic stream operators (**core operators**)
(unit, nil, lift, last, delay, time)

- ▶ **Stream-based** specification language with **functional part**
- ▶ Based on six basic stream operators (**core operators**)
(unit, nil, lift, last, delay, time)
- ▶ **Macro system** for definition of user defined stream operators
Lots of predefined macros in the stdlib

- ▶ **Stream-based** specification language with **functional part**
- ▶ Based on six basic stream operators (**core operators**)
(unit, nil, lift, last, delay, time)
- ▶ **Macro system** for definition of user defined stream operators
Lots of predefined macros in the stdlib
- ▶ **Type system** with support of generic types

- ▶ **Stream-based** specification language with **functional part**
- ▶ Based on six basic stream operators (**core operators**)
(unit, nil, lift, last, delay, time)
- ▶ **Macro system** for definition of user defined stream operators
Lots of predefined macros in the stdlib
- ▶ **Type system** with support of generic types
- ▶ **Annotation system** for steering of the RV tool chain

TeSSLa - Example Specification



```
# Inputs
@InstFunctionCall("read_brake_sensor")
in read_brake_sensor: Events[Unit]
@InstFunctionCall("activate_brakes")
in activate_brakes: Events[Unit]

# Trace Processing
def latency = measureLatency(read_brake_sensor,
                             activate_brakes)

def error = latency > 4ms
def high = filter(latency, error) - 4ms
def is_critical = count(high) > 10
def critical = filter(high, is_critical)

# Output
@VisDots out high
@VisEvents out critical

# Macro
def measureLatency[A, B](a: Events[A],
                        b: Events[B]) =
    time(b) - last(time(a), b)
```

TeSSLa - Example Specification



```
# Inputs
@InstFunctionCall("read_brake_sensor")
in read_brake_sensor: Events[Unit]
@InstFunctionCall("activate_brakes")
in activate_brakes: Events[Unit]

# Trace Processing
def latency = measureLatency(read_brake_sensor,
                             activate_brakes)

def error = latency > 4ms
def high = filter(latency, error) - 4ms
def is_critical = count(high) > 10
def critical = filter(high, is_critical)

# Output
@VisDots out high
@VisEvents out critical

# Macro
def measureLatency[A, B](a: Events[A],
                        b: Events[B]) =
  time(b) - last(time(a), b)
```

} Input decl.
&
annotations

TeSSLa - Example Specification



```
# Inputs
@InstFunctionCall("read_brake_sensor")
in read_brake_sensor: Events[Unit]
@InstFunctionCall("activate_brakes")
in activate_brakes: Events[Unit]

# Trace Processing
def latency = measureLatency(read_brake_sensor,
                             activate_brakes)

def error = latency > 4ms
def high = filter(latency, error) - 4ms
def is_critical = count(high) > 10
def critical = filter(high, is_critical)

# Output
@VisDots out high
@VisEvents out critical

# Macro
def measureLatency[A, B](a: Events[A],
                         b: Events[B]) =
  time(b) - last(time(a), b)
```

Input decl.
&
annotations

Monitoring
property

TeSSLa - Example Specification



```
# Inputs
@InstFunctionCall("read_brake_sensor")
in read_brake_sensor: Events[Unit]
@InstFunctionCall("activate_brakes")
in activate_brakes: Events[Unit]

# Trace Processing
def latency = measureLatency(read_brake_sensor,
                             activate_brakes)

def error = latency > 4ms
def high = filter(latency, error) - 4ms
def is_critical = count(high) > 10
def critical = filter(high, is_critical)

# Output
@VisDots out high
@VisEvents out critical

# Macro
def measureLatency[A, B](a: Events[A],
                        b: Events[B]) =
  time(b) - last(time(a), b)
```

} Input decl.
&
annotations

} Monitoring
property

} Output decl.
&
annotations

TeSSLa - Example Specification



```
# Inputs
@InstFunctionCall("read_brake_sensor")
in read_brake_sensor: Events[Unit]
@InstFunctionCall("activate_brakes")
in activate_brakes: Events[Unit]

# Trace Processing
def latency = measureLatency(read_brake_sensor,
                             activate_brakes)

def error = latency > 4ms
def high = filter(latency, error) - 4ms
def is_critical = count(high) > 10
def critical = filter(high, is_critical)

# Output
@VisDots out high
@VisEvents out critical

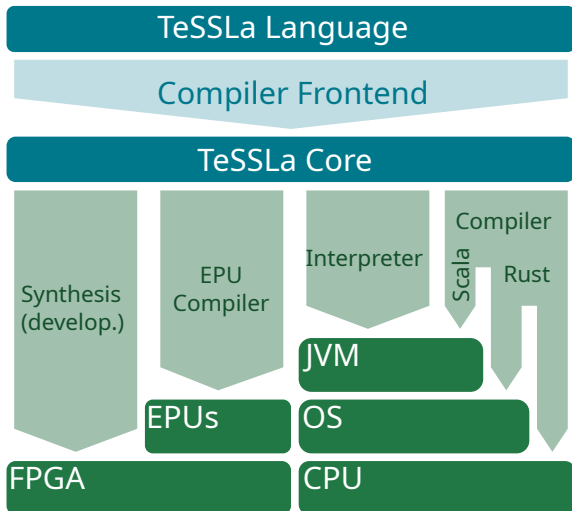
# Macro
def measureLatency[A, B](a: Events[A],
                        b: Events[B]) =
    time(b) - last(time(a), b)
```

} Input decl.
&
annotations

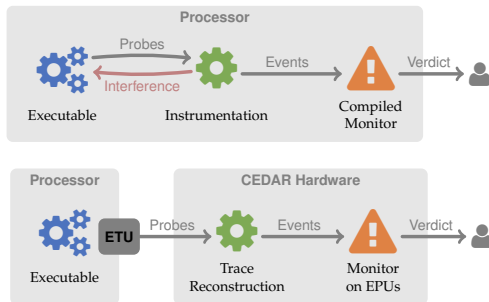
} Monitoring
property

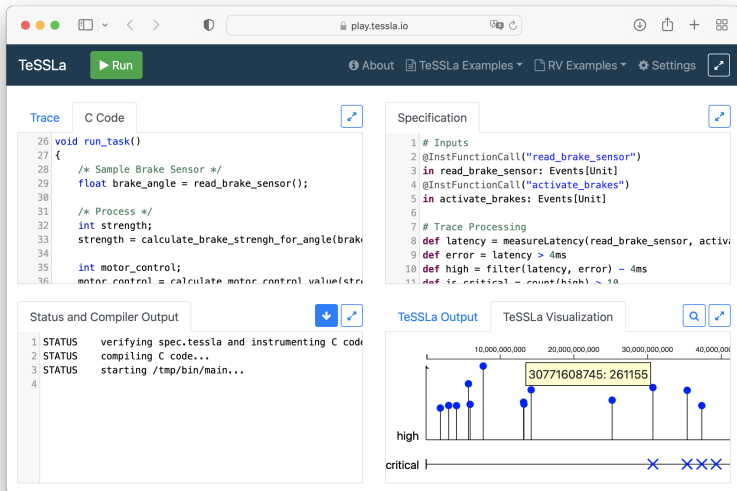
} Output decl.
&
annotations

} Macro
definitions



- ▶ Instrumenter for C code integrated in compiler
- ▶ Accemic's CEDARtools for non-intrusive hardware monitoring
- ▶ Connection to other instrumentation tools via generic annotation system





The screenshot shows the TeSSLa web IDE interface. The browser address bar is `play.tessla.io`. The interface has a dark blue header with the TeSSLa logo, a green 'Run' button, and navigation links for 'About', 'TeSSLa Examples', 'RV Examples', and 'Settings'.

The main content area is divided into four panels:

- Trace / C Code:** Shows C code for a task. Lines 26-36 are visible:

```
26 void run_task()
27 {
28     /* Sample Brake Sensor */
29     float brake_angle = read_brake_sensor();
30
31     /* Process */
32     int strength;
33     strength = calculate_brake_strength_for_angle(brake_angle);
34
35     int motor_control;
36     motor_control = calculate_motor_control_value(strength);
```
- Specification:** Shows a specification for the task. Lines 1-11 are visible:

```
1 # Inputs
2 @InstFunctionCall("read_brake_sensor")
3 in read_brake_sensor: Events[Unit]
4 @InstFunctionCall("activate_brakes")
5 in activate_brakes: Events[Unit]
6
7 # Trace Processing
8 def latency = measureLatency(read_brake_sensor, activate_brakes)
9 def error = latency > 4ms
10 def high = filter(latency, error) - 4ms
11 def is_critical = count(high) > 10
```
- Status and Compiler Output:** Shows the output of the compiler. Lines 1-4 are visible:

```
1 STATUS verifying spec.tessla and instrumenting C code...
2 STATUS compiling C code...
3 STATUS starting /tmp/bin/main...
4
```
- TeSSLa Output / TeSSLa Visualization:** Shows a visualization of the task's execution. The x-axis represents time in nanoseconds, ranging from 0 to 40,000,000,000. The y-axis has two levels: 'high' and 'critical'. Blue dots represent high events, and blue 'x' marks represent critical events. A yellow box highlights a specific high event at time 30771608745:261155.

constIf

```
constIf[T](value: T, condition: Events[Bool]): Events[T]
```

Produce an event with the given value every time that the condition is met

Usage example:

```
in condition: Events[Bool]
def result = constIf(42, condition)
out result
```

Trace example:



Source

```
def constIf[T](value: T, condition: Events[Bool]): Events[T] =
  filter(const(value, condition), condition)
```

count

```
count[T](x: Events[T]): Events[Int]
```

Count the number of events on `x`. Provides for every input event an output event whose value is the number of events seen so far. See [resetcount](#) for a counting macro with an external reset.

► User Libraries

Macro system allows definition of application-specific libraries

E.g. AUTOSAR Timex, Past LTL libraries...

- ▶ **User Libraries**

Macro system allows definition of application-specific libraries
E.g. AUTOSAR Timex, Past LTL libraries...

- ▶ **Tutorials**

Extensive tutorials about the usage of the TeSSLa language and tools.

- ▶ **User Libraries**

Macro system allows definition of application-specific libraries
E.g. AUTOSAR Timex, Past LTL libraries...

- ▶ **Tutorials**

Extensive tutorials about the usage of the TeSSLa language and tools.

- ▶ **Open-Source availability**

Free availability of most parts of the tool chain.
Community-driven project.

Future plans



The development of TeSSLa is still in progress...

Future plans



The development of TeSSLa is still in progress...

- ▶ Improvement of the TeSSLa language and compilers

The development of TeSSLa is still in progress...

- ▶ Improvement of the TeSSLa language and compilers
- ▶ Extension of the tool chain: Further backends, integration with other tools

The development of TeSSLa is still in progress...

- ▶ Improvement of the TeSSLa language and compilers
- ▶ Extension of the tool chain: Further backends, integration with other tools
- ▶ Development of further libraries for specific RV applications

Find out more



TeSSLa Website:
<https://www.tessla.io/>

TeSSLa Playground:
<https://play.tessla.io/>

TeSSLa Sourcecode:
<https://git.tessla.io/>

Contact:
info@tessla.io