

– MLCCA –  
Multi-Level Composability Check Architecture  
for Dependable Communication  
over Heterogeneous Networks

*Georg Lukas, Michael Schulze*  
glukas,mschulze@ovgu.de

Department of Distributed Systems  
Otto-von-Guericke University of Magdeburg

ETFA 2009  
Palma de Mallorca



# Outline

- 1 Motivation
- 2 MLCCA Concept
- 3 Case Study
- 4 Conclusion



# Motivation

## Industrial Automation: Communication Demands

- Real-Time, reliable communication
- Heterogeneous systems
- Wireless networks gaining popularity
- Plethora of platforms and protocols

# Motivation

## Industrial Automation: Communication Demands

- Real-Time, reliable communication
- Heterogeneous systems
- Wireless networks gaining popularity
- Plethora of platforms and protocols

### Main Question

How to ensure end-to-end application communication requirements?

- Latency, jitter, omission degree, data throughput
- Scalability, portability (sensors . . . servers)

# Motivation

## Industrial Automation: Communication Demands

- Real-Time, reliable communication
- Heterogeneous systems
- Wireless networks gaining popularity
- Plethora of platforms and protocols

### Main Question

How to ensure end-to-end application communication requirements?

- Latency, jitter, omission degree, data throughput
- Scalability, portability (sensors ... servers)

⇒ Growing demand in industrial and other scenarios

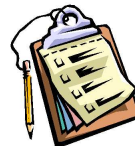
# MLCCA Approach

## Multi-Level Composability Check Architecture

- Automated component composability checking
  - Application requirements
  - System (network stack, OS) properties
- Functional as well as non-functional attributes
- Integration into a common (communication) middleware
- Detection of mismatches as early as possible
- Multi-Level checking
  - Design Time
  - Compile Time
  - Run Time

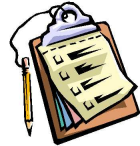
# Design-Time Checking

- System Design Tool
- Component Descriptions
  - Properties, requirements, ...
  - Machine processable; e.g. in XML
- What checks are possible on system compositions?
  - Completeness of specification
  - Correctness



# Design-Time Checking

- System Design Tool
- Component Descriptions
  - Properties, requirements, ...
  - Machine processable; e.g. in XML
- What checks are possible on system compositions?
  - Completeness of specification
  - Correctness



```
<Description>emergency switch</Description>
  <Attributes>
    <Attribute> <Name>Period</Name>    <Value>50</Value>
  </Attribute>
    <Attribute> <Name>Omission</Name>  <Value>2</Value>
  </Attribute>
  </Attributes>
```



# Design-Time Checking

- System Design Tool
- Component Descriptions
  - Properties, requirements, ...
  - Machine processable; e.g. in XML
  - **Generation of source code**
- What checks are possible on system compositions?
  - Completeness of specification
  - Correctness

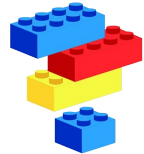


```
<Description>emergency switch</Description>
  <Attributes>
    <Attribute> <Name>Period</Name>    <Value>50</Value>
  </Attribute>
    <Attribute> <Name>Omission</Name>  <Value>2</Value>
  </Attribute>
</Attributes>
```

# Compile-Time Checking

Why is the second stage necessary?

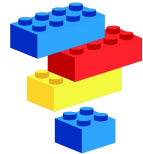
- Refactoring without design tool support
- Writing applications from scratch



# Compile-Time Checking

## Why is the second stage necessary?

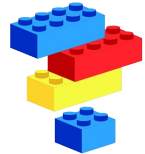
- Refactoring without design tool support
- Writing applications from scratch



## Challenge of compile-time checks

- Testing of attribute constraints
- Generation of compile-time errors ...
  - ... despite of correct code
  - ... with expressive messages

# Compile-Time Checking



## Why is the second stage necessary?

- Refactoring without design tool support
- Writing applications from scratch

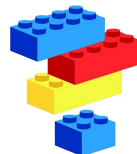
## Challenge of compile-time checks

- Testing of attribute constraints
- Generation of compile-time errors ...
  - ... despite of correct code
  - ... with expressive messages

## The solution

- Template Meta-Programming (TMP)

# Compile-Time Checking



## Why is the second stage necessary?

- Refactoring without design tool support
- Writing applications from scratch

## Challenge of compile-time checks

- Testing of attribute constraints
- Generation of compile-time errors ...
  - ... despite of correct code
  - ... with expressive messages

## The solution

- Template Meta-Programming (TMP)
- Generation of machine-readable composability information

# Compile-Time Checking: Implementation

How is it performed?

```
struct EmergencySwitchSpec {  
    typedef AttributeList<  
        Period<50>,  
        Omission<2>  
    >::type attributes; };
```

# Compile-Time Checking: Implementation

How is it performed?

```
struct EmergencySwitchSpec {  
    typedef AttributeList<  
        Period<50>,  
        Omission<2>  
    >::type attributes; };
```

```
typedef CheckAttributeAgainstAttributeList<  
    EmergencySwitchSpec::attributes,  
    Omission<5>  
>::type test;
```

# Compile-Time Checking: Implementation

How is it performed?

```
struct EmergencySwitchSpec {
    typedef AttributeList<
        Period<50>,
        Omission<2>
    >::type attributes; };

```

```
typedef CheckAttributeAgainstAttributeList<
    EmergencySwitchSpec::attributes,
    Omission<5>
>::type test;

```

Compiler output:

```
CheckAttributeAgainstAttributeList.h:60: error: no matching
function for call to assertion_failed(*** (
TestAttributes<Omission<5u>, deref<l_iter<list1<
Omission<2u> > > >::ATTRIBUTES_MISMATCH::***) (
Omission<5u>, Omission<2u>))

```



# Run-Time Checking

## Typical Run-Time Checks

- Service discovery (UPnP, Jini, ...)
- Matching of functional attributes (CORBA)
- Requirement of Composability Checks?



# Run-Time Checking



## Typical Run-Time Checks

- Service discovery (UPnP, Jini, ...)
- Matching of functional attributes (CORBA)
- Requirement of Composability Checks?
  - Crossing network borders (tests on gateways)
  - Dynamic network topologies (WMN)
  - Overload on addition of new components

# Run-Time Checking



## Typical Run-Time Checks

- Service discovery (UPnP, Jini, ...)
- Matching of functional attributes (CORBA)
- Requirement of Composability Checks?
  - Crossing network borders (tests on gateways)
  - Dynamic network topologies (WMN)
  - Overload on addition of new components

## Run-Time Composability Checking

- Automatic evaluation of component attributes
  - Matching of application vs. network layer(s)
  - Consideration of routing information
- ⇒ No service > unreliable service

# Middleware Integration

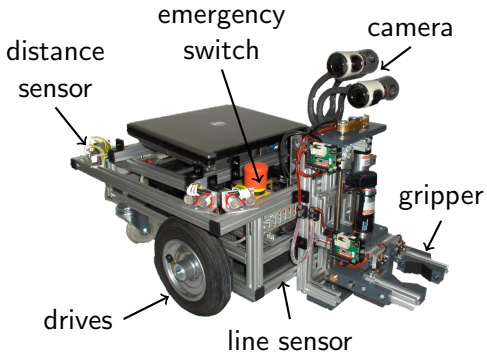
## MLCCA integration in communication middleware

- Design-Time, Compile-Time, Run-Time checking
- Transparency for applications
- Portability (100% standard C++98)
- Scalability (8-bit  $\mu$ C ... 64-bit server)

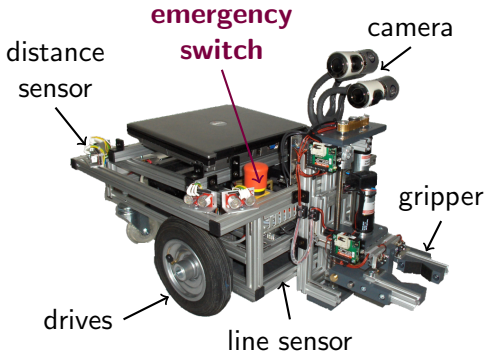
## Implementation in FAMOUSO

- event-based pub/sub communication middleware
- Support for C++, Python, Matlab, ...
- Integration of CAN, WMN, UDP/MC, ...
- <http://famouso.sourceforge.net/>

# Application Example: Emergency Switch



# Application Example: Emergency Switch



## Emergency Switch (ES)

- periodic "okay" messages
- absence  $\Rightarrow$  emergency

## Component Requirements:

- period: 50ms
- max. two omissions

# Emergency Switch: System Configuration CAN

- Emergency switch on local CAN bus



# Emergency Switch: System Configuration CAN

- Emergency switch on local CAN bus



MLCCA:



Design Time



Compile Time

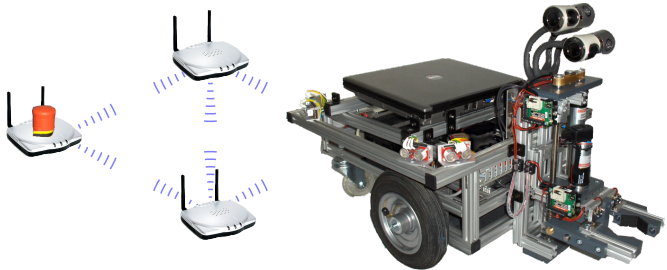


Run Time



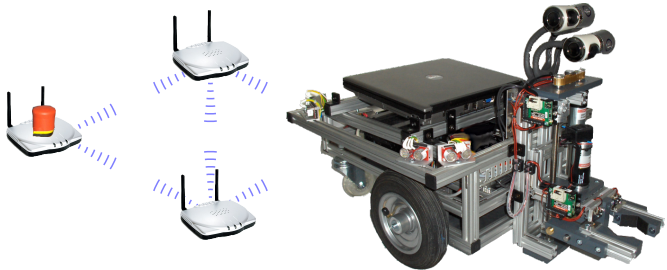
# Emergency Switch: System Configuration WMN

- Emergency switch over WMN



# Emergency Switch: System Configuration WMN

- Emergency switch over WMN

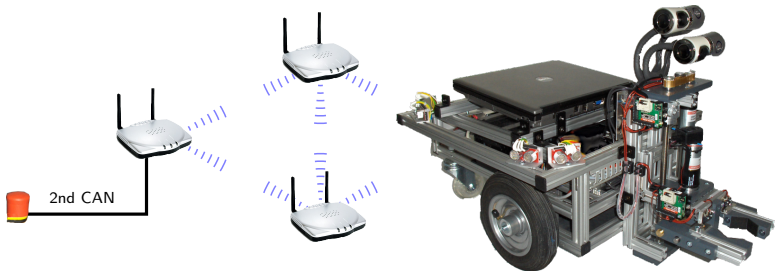


MLCCA:

× Design Time | × Compile Time | ? Run Time

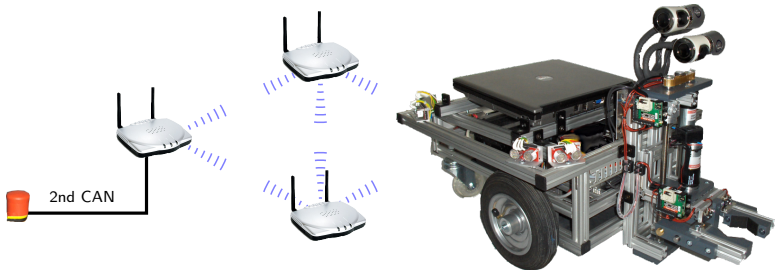
# Emergency Switch: System Configuration WMN/CAN

- Emergency switch on CAN, via WMN gateways



# Emergency Switch: System Configuration WMN/CAN

- Emergency switch on CAN, via WMN gateways



MLCCA:

× Design Time | ✓ Compile Time | × Run Time

# Emergency Switch: Summary

- No automatic way to make wireless networks 100% reliable
- Prevention of unreliable setups ...
  - ... as early as possible
  - ... reliably
  - ... over network/protocol borders

# Conclusion

## Initial Question

How to ensure end-to-end application communication requirements?

# Conclusion

## Initial Question

How to ensure end-to-end application communication requirements?

1. Formal specification of non-functional attributes
2. Extension of Design Tool checks
3. Novel Compiler-based Checks
4. Automatic Run-Time checks in middleware

# Conclusion

## Initial Question

How to ensure end-to-end application communication requirements?

1. Formal specification of non-functional attributes
2. Extension of Design Tool checks
3. Novel Compiler-based Checks
4. Automatic Run-Time checks in middleware
  - Implementation in FAMOUSO communication middleware
  - Applicability for Industrial Automation scenarios





# CODES Creator

CODES Creator [C:\IR\ir.xml] 0.11 \*

Node UID: 0x544930364134832 Generate

Device Name: IRTiny

Device Type: 8-fold infrared distance sensor

Manufacturer: University of Ulm

Processor: MC68HC908AZ60A

Connections: CAN 2.0b + -

Hardware Version: 2.0

System Software Version: 1.0

Description: This is board provides 8 infrared distance channels.

Full Information: <http://www.informatik.uni-ulm.de/ts/mitarbeits>

Supported Event Channel Types:  
☒ Hard Real-Time ☐ Soft Real-Time ☒ Non Real-Time

Events:

UID	Subject	Descri...	Fieldcount	Payload size	Attri
0x505f4952504d5347	distance	Period...	8	64	Exp
0x505f4952414c524d	distance alarm	Is diss...	8	64	Exp
0x505f49525f434647	Configuration	Set al...	2	16	Exp
0x505f49525f544852	Threshold data		8	64	Exp
0x505f49525f485953	Hysteresis data		8	64	Exp

New Event Remove Event Edit Event

Event Channels:

Subject	Type	Direction	Attributes
distance	HRT	producing	period=100...
distance alarm	HRT	producing	
Configuration	HRT	consuming	
Threshold data	NRT	producing	
Hysteresis data	HRT	producing	

Replace Sanity Check Edit Channel

# Middleware Overview

## Attribute-Based Composability Checks

	Design-Time	Compile-Time	Run-Time
WSDL*	✗	—	✓
UPNP*	✗	—	✓
Jini*	✗	—	✓
CANopen*	✓	—	—
IEEE 1451*	✓	—	—
CORBA	✗	—	✓
MLCCA	✓	✓	✓

\*) no support for QoS attribute descriptions

# FAMOUSO Middleware

