ESEVO
The Time-Triggered Architecture

Bernhard Frömel

based on slides by Hermann Kopetz, Christian El-Salloum, and Armin Wasicek.
Institute of Computer Engineering
Vienna University of Technology
- 182.722 Embedded Systems Engineering LU

November, 2014
Part I

The Time-Triggered Architecture [3][5]
Technological Paradise

“[In a] Technological Paradise no acts of God can be permitted and everything happens according to the blueprints.” [Hannes Alfven]¹.

We are not living in a technology paradise!

¹Nobel laureate
Systems, Subsystems, and Components [1]

- **System**: An entity that is capable of interacting with its environment and may be sensitive to the progression of time.

- **Environment of a System**: The entities and their actions in the Universe of Discourse (UoD) that are not part of a system but have the capability to interact with the system.

- **System Boundary**: A dividing line between two systems or between a system and its environment.

- **Subsystem**: A subordinate system that is part of an encompassing system.

- **Component**: A subsystem of a system, the internal structure of which is of no interest.

- **Cyber-Physical System (CPS)**: A system consisting of a computer system (the cyber system), a controlled object (a physical system) and possibly of interacting humans.

- **Real-time system**: A computer system which must produce value-correct results within time-constraints.
System Architecture: The blueprint of a design that establishes the overall structure, the major building blocks and the interfaces among these major building blocks and the environment.

Architectural Style: The set of explicit or implicit rules and conventions that determine the structure and representation of the internals of a system, its data and protocols.
Interface [1]

- **Interface**: A point of interaction of a system with another system or with the system environment.
- **Behavior**: The timed sequence of the effects of input and output actions that can be observed at an interface of a system.
- **Deterministic Behavior**: A system behaves deterministically if, given an initial state at a defined instant and a set of future timed inputs, the future states, the values and instants of all future outputs are entailed.
- **Service**: The intended behavior of a system.
Time [1]

- **Time**: A continuous measurable physical quantity in which events occur in a sequence proceeding from the past to the present to the future.
- **Instant**: A cut of the timeline.
- **Interval**: A section of the timeline between two instants.
- **Event**: A happening [i.e., change of state] on the timeline.
State: The state of a system at a given instant is the totality of the information from the past that can have an influence on the future behaviour of a system.

Information: A proposition about the state of or an action in the world.
Communication [1]

- **Message**: A data structure that is formed for the purpose of the timely exchange of information among computer systems.
- **Channel**: A logical or physical link that transports information among systems at their connected interfaces.
- **Protocol**: The set of rules that govern a communication action.
What is a Component?

“It is much easier to build a wall with bricks than with stones.”[Kopetz?]

- Component is a building block for the (ideally side-effect free) construction of larger systems.
- Self-contained hardware-software unit that has behavior and state (not software alone!).
- Has access to a global sparse time base.
- Communicates with its environment solely over its (external) interfaces by exchange of messages.

**Purpose:** Components process information.
Data versus Information

- Data (e.g., a bit pattern) represents information.
- Information can only be retrieved if an explanation is either explicitly or implicitly (context, shared ontology) available.
- Mismatch problem: systems adhering to different architectural styles interpret data differently (e.g., degrees Celsius vs. degrees Fahrenheit).
- Concept of *Itom* [4]: An Itom (Information Atom) is a tuple consisting of data and the associated explanation of the data.
- Open research: self-describing data, machine interpretable explanations that can be used for *gateway components*.
- By the way: what is information?
- **Information**: A proposition about the state of or an action in the world.
Sparse Global Time

- Problem: agreement on time and order of observations originating from different components.
- Establishment of consistent order in asynchronous setting difficult.
- Global time by local clock synchronization
  - Reasonableness condition: Granule $\mathcal{G}$ of global clock must be larger than precision $\Pi$ of clock synchronized components.
  - $\pi/\Delta$-precedence: Events only occur within interval $\pi$, but not in $\Delta$.
- $\pi/\Delta$ sparse global time
  - Enforced/agreed $\pi/\Delta$-precedence: interval of activity, interval of silence
  - Exam question (?): Is a 1/4 sparse time-base sufficient as a consistent\(^2\) global-time base?

\(^2\)Every component that has access to the sparse global time-base arrives at the same temporal order of time-stamped observations, regardless which component has observed and time-stamped a observation.
Interfaces of a Component

- One interface per purpose
- External interfaces
  - Linking Interface (LIF): Offers component’s service to other components
  - Utility Interfaces: Configuration/Planning Interface, Diagnosis Interface, Local I/O Interface
- Linking Interface (LIF) as a boundary:
  - Requires memory to store and check message for validity before it passes interface
  - Event messages: queue, State message: shadowed memory (memory where two messages fit such that one message is always consistent)
## Message Classification

<table>
<thead>
<tr>
<th>Property</th>
<th>Explanation</th>
<th>Antonym</th>
</tr>
</thead>
<tbody>
<tr>
<td>valid</td>
<td>A message is valid if its checksum and contents are in agreement.</td>
<td>invalid</td>
</tr>
<tr>
<td>checked</td>
<td>A message is checked at source (or, in short, checked) if it passes the output assertion.</td>
<td>not checked</td>
</tr>
<tr>
<td>permitted</td>
<td>A message is permitted with respect to a receiver if it passes the input assertion of that receiver.</td>
<td>not permitted</td>
</tr>
<tr>
<td>timely</td>
<td>A message is timely if it is in agreement with the temporal specification.</td>
<td>untimely</td>
</tr>
<tr>
<td>value-correct</td>
<td>A message is value-correct if it is in agreement with the value specification.</td>
<td>not value-correct</td>
</tr>
<tr>
<td>correct</td>
<td>A message is correct if it is both timely and value-correct.</td>
<td>incorrect</td>
</tr>
<tr>
<td>insidious</td>
<td>A message is insidious if it is permitted but incorrect.</td>
<td>not insidious</td>
</tr>
</tbody>
</table>
Interface Ports

- Ports are communication channel endpoints for messages: input and output ports.
  - Port properties (e.g., message name, message length, direction, temporal constraints, ... ) determine channel properties (e.g., max. jitter, latencies, ...).
  - In interfaces with memory a port can be modeled as an inner port and a matching outer port:
    - **Inner Port**: access point of interface memory for component.
    - **Outer Port**: access point of interface memory for communication system.
    - Inner and outer ports may adhere to different protocols.
  - Interface may have many ports where communication actions may occur concurrently.
  - However: no concurrency at a single port.
  - An output message can only depend on input messages that have been read at an earlier instant.
Information Push versus Information Pull

**Information Push**
- Producer pushes information to consumer
- Examples: interrupt
- For reads: disruptive, because control delegated to environment
- For writes: natural, because control remains within component

**Information Pull**
- Consumer requests information
- Examples: email
- For reads: natural, because control remains within component
- For writes: disruptive, because control delegated to environment

What would you prefer for real-time systems?

It depends on whether you are producing or consuming information!

- Input ports ⇒ Information pull
- Output ports ⇒ Information push
Consider unidirectional data flow from sender to receiver component.

**Elementary Interface**
- Unidirectional control flow.
- Sender remains independent of receiver.
- Example: Satellite-based TV

**Composite Interface**
- Bidirectional control flow.
- Sender dependent from receiver.
- Example: TCP/IP
Temporal Firewall

Desired control flow semantics of interfaces for real-time components:

- Information consuming component can pull information.
- Information producing component can push information.
- No control flow dependencies between sender and receiver.

An interface is a **temporal firewall** if it prohibits external control on the component.

If an interface is not a temporal firewall, then back-propagation of faults possible!
LIF Specification

- Precise description of messages w.r.t. value and time domain.
- Agnostic of concrete component implementation technology.
- Component can be used solely by its LIF specification without knowledge about its internals.
- Allow for phase-alignment of computation and communication actions so that a RT transaction can complete in the same cycle (cf. modeling of time).
Principles of the TTA Architectural Style

- **Abstraction**: Component as basic structural, computational and design unit determined by interface specification. What about faults?
- **Separation of Concerns**: Disentangled functions: e.g., computational and communication activities separated s.t. computation subsystems and communication subsystems can be developed independently.
- **Causality**: Deterministic behavior of core services to establish causal chain between cause and effects.
- **Segmentation**: Temporal separation of complex behavior – wherever possible. Sequential behavior more simple than concurrent.
- **Independence**: Interdependence of architectural elements as minimal as possible.
- **Observability**: Side-effect free external observation of components.
- **Consistent Time**: Establish system-wide consistent temporal relations and temporal distances among events.
Complexity management, recursive component concept, and coherent communication

- Fault-tolerant *sparse global time-base*
- Real-time transactions spanning across multiple components have guaranteed end-to-end temporal properties.
- Components can be integrated to form hierarchical structures
- Single mechanism of component interaction which is independent of component location
Development Process

1. System design by interface specification which determines how components can interact.
2. Node design according to interface specification
MARS

- **1980s**: Few skilled engineers developed systems where minor changes required expensive readjustments/testing (engineering by intuition)
- Risk of failing to deliver required performance during *rare event scenarios*
- Project MAintainable Real-time System (MARS) started in 1979, TU Berlin
  - Objective: Strong conceptual basis, constructive methods for systematic design and maintenance of RT systems
- First prototype in 1982 showed that more fundamental research was required
- In 1983 MARS project moved to TU Vienna where a second prototype was developed: The Rolling Ball on MARS: [http://pan.vmars.tuwien.ac.at/mars/](http://pan.vmars.tuwien.ac.at/mars/)
- Major achievement: Clock Synchronization Unit (CSU), VLSI, fault tolerant
Further Research

- Academic research success lead to follow up projects
- Dependable Embedded COmponents and Systems (DECOS), 2004–2007
- Time-Triggered Communication Architecture for Robotic systems (TTCAR), 2005–2008
- Time-Triggered System-on-Chip (TT-SoC) 2007–2009
- GENeric Embedded System Platform (GENESYS) 2008–2009
- INDustrial EXploitation of the genesYS cross-domain architecture (INDEXYS), 2009–2012
- ARTEMIS CROSS-Domain architecture (ACROSS), 2010–2013
Industrial Exploitation

- Industrial TTA prototype of a fault-tolerant brake-by-wire system
- In 1998 the TU Vienna spin-off TTTech\(^3\) has been founded
  - Airbus A380 (TTP/C based cabin pressure control system)
  - Boeing 787 “Dreamliner” (TTP/C)
  - AUDI A8 premium car (FlexRay-based data communication)
  - NASA Orion program (TTEthernet)
  - ...

\(^3\)http://tttech.com
GENESYS, a Generic Cross-Domain Architecture

- Many embedded systems challenges are identical in different application domains
  - Composability: Reuse of components
  - Robustness: Reduce system fragility
  - Dependability (incl. Security)
  - Energy efficiency
  - Predictability (temporal!)
- Integration of systems of different domains
  - Access car/airplane/medical device/... with smart phone
- Economies of scale w.r.t. semiconductor industry
- Unified development methodology
ACROSS, an Embedded MPSoC Implementation of GENESYS Reference Architecture

- Multi-core scalable w.r.t. computational power.
- Multi-core running at lower clock frequency more energy efficient than single-core running at high clock frequency.
- Heterogeneous cores can be tailored to specific functionalities of an embedded system (e.g., security cores, video encoders/decoders, ...).
- Integration of multiple cores in a single chip reduces number of wires and connectors. What’s the benefit?
- Potential performance gain in embedded applications
  - What about Amdahl’s law?

\[ S(n) = \frac{1}{B + \frac{1}{n} (1 - B)}, \text{ B .. strictly serial fraction.} \]
However, typical embedded applications consist of many concurrently and independently operating subsystems.
Why not use existing MPSoCs?

- Classified as "highly complex microcontrollers" (EASA, FAA) ⇒ High impact on certification efforts!
- No temporal determinism, because optimized for fast average execution and too complex to analyze.
  - Instruction reordering, data caches, …
- Insufficient temporal and spatial isolation of concurrently executing independent functions.
  - Shared resources (e.g., caches, I/O, power management, …).
  - Especially a problem, if independent functions have different criticality (Safety Integrity Level). Why?
    In case of insufficiently isolated, multiple safety functions with different SLI requirements, highest must be used for all.
Objectives of ACROSS

- Provide heterogeneous MPSoC architecture that enables certification for highest criticality classes (1 FIT).
- Construction of systems that are temporally predictable.
- Complexity management by allowing for independent development of subsystems (prevent non-intended interference).
- Enable mixed-criticality integration (prevent error propagation between subsystems).

ACROSS has been implemented on top of the TTSoc Architecture (see later).
AMADEOS, 2013 – 2016

Architecture for Multi-criticality Agile Dependable Evolutionary Open System-of-Systems

- Time-Aware SoSs, Availability of a Sparse Global Time
- Emergent phenomena
- Dynamicity
- Evolution
- Smartgrid Usecase
AMADEOS, 2013 – 2016

Challenges:

- Conceptual model
- Information transfer
- Interface design
Part II

An Application Development Approach for the Time-Triggered System-on-Chip Architecture [6][7][2]
Motivation

- Time-Triggered Architecture (TTA) for building dependable distributed component-based Systems-on-Chips.
- Developing applications for the Time-Triggered System-on-Chip (TTSoC) Architecture challenging.
- Existing work “solves” this problem only in theory.
- Strong application development approach key to make the TTSoC Architecture and other TTA-based architectures accessible for research, education and industry.
The TTSoC Architecture

- Resource Management Authority (RMA)
- Diagnostic Unit (DU)
- User component 1
- User component 2
- TTE gateway
- User component 3
- User component 4
- User component 5

Time-Triggered Network-on-Chip (TTNoC) Interconnect

Distributed application A

Distributed application B
TTNoC interconnect (own clock domain)
The TTSoC Architecture

- Components attached to TTNoC Interconnect.
- TTNoC Interconnect provides Encapsulated Communication Channels (ECCs) adhering to the Pulsed Data Stream (PDS) paradigm.
- Architectural elements Resource Management Authority (RMA) and TTNoC Interconnect form trusted subsystem
  - provide and controls shared NoC resources.
  - RMA responsible for dynamic reconfiguration.
- Diagnostic Unit (DU) monitors and validates behavior of distributed applications.
- Time-Triggered Ethernet (TTE) for off-chip communication.
- User components form distributed applications.
Model-based Development Process

- Platform Independent Model
- Abstract Application Model
- Fully-Specified Interface Model
  - Macro FIM
  - Uniform FIM
- Physical Allocation Model
- Platform Specific Model
Recursive Component Concept

Applied a recursive component concept to TTSoC Architecture:

- Components can contain other components.
- Gateways can establish links between components.

⇒ Recursive component concept avoids special cases in model transformations.
Recursive Component Concept Examples

(a) Virtual Components

TTNoC Interconnect

host component

TT-OS with VCRE

TTNoC Interconnect gateway

Virtual Interconnect

virtual component (task)

(b) NoC Bandwidth and Power Scaling

Low Speed TTNoC Interconnect

gateway

cmponent

cmponent

High Speed TTNoC Interconnect

cmponent

cmponent

cmponent

slow clock

fast clock
Virtual Components

- Implemented Time-Triggered Operating System for Virtual Components (TTOSVC) for
  - Posix-compliant library (debugging),
  - Leon 3 (Sparcv8 softcore CPU), and
  - Nios II (Altera softcore CPU).

- Implemented an execution environment for *virtual components*:
  - For TTOSVC and Linux kernel based operating systems.
  - Consists of set of libraries and callbacks for virtual components.
  - Establishes Virtual Interconnect to enable component interaction.
Topography Invariant Scheduling of PDSes in the TTNoC Interconnect

- Use of Dijkstra algorithm and its weight function to find valid routes in NoC.
- Implemented in C and deployed for dynamic resource management on RMA.
Service-Oriented Application Development Approach

- System under development represented as models at different abstraction levels.
- Platform executable model derived by model transformations.
- Service is container for behavior and non-functional properties.
- Service decomposition gives service dependency tree that captures service interaction and guides definition of message-based Linking Interface (LIF).
- Mapping of services to components defines LIF of component.
Design Flow

Platform Independent Model
- derive services from requirements
- decompose services to service dependency tree

Abstract Application Model (AAM)

Platform Specific Model
- refine services OR use OEM services
- map services to components OR use OEM components

Macro Fully-Specified Interface Model (MFIM)
- elaborate macros (e.g., resolve communication channels, TMR, ...)

Uniform Fully-Specified Interface Model (UFIM)

Platform Model (PM)
- allocate platform components conforming to UFIM

UFIM-PM allocation

Platform Allocation Model (PAM)

Tools
- instantiate physical communication channels (scheduling)

Skill

Time-Triggered Architecture Agnostic

Time-Triggered Architecture Specific
Towards MFIM

R₁ → S₁
R₂ → S₂
... → ...
Rₙ → Sₘ

decomposition (e.g., S₁)

S₁
S₁,₁
S₁,₂
S₁,₃
S₁,₄
S₁,₅
... S₁,k
S₁,k+₁

refinement (e.g., S₁,₃)

outgoing message ports

I/O interface

formal behavior description

assertions, multi-client

incoming message ports

P₁
P₂
... Pᵢ
Pᵢ₊₁
Pᵢ₊₂
... Pᵢ₊ᵢ

Case Study

Application Development Approach

Evolution

TTSoC Architecture

Frömel

ESEVO The Time-Triggered Architecture

Conclusion

45/59
Towards PAM

- Defined meta-models (Eclipse Modeling Framework) for MFIM, PM and UFIM-PM allocation.

- Tool produces deployable platform configuration data for RMA, DU, and gateway (e.g., schedules for TTNoC Interconnect, monitoring and mapping information, ...).
Results/Benefits

- Separated development of distributed, time triggered applications.
- Validation of distributed applications (traceability).
- Suppliers and legacy components (top-down vs. bottom-up).
- Automated model transformations for:
  - resolving service dependency relationships to UFIM-channels.
  - resolving non-functional properties to architectural means.
  - instantiating GWs.
Case-Study: Mixed Criticality

- Mixed criticality demonstrator:
  - automotive applications: high criticality.
  - multimedia application: low criticality, multiple modes of degradation.

- Subset of (toy) requirements:
  R1 The vehicle shall remain steerable during (emergency) braking.
  R2 The driver shall be able to steer the vehicle according to a configurable steering translation.
  R3 During high speeds the driver shall be able to apply at most 1/3 of the possible car steering angle.
  R4 Car acceleration shall respond to an electronic gas pedal.
  R5 Requirements R1, R2, R3 and R4 shall be certified up to safety-criticality levels.
  R6 ...
Service Dependency Tree/Forest

- FL wheel speed
- FR wheel speed
- RR wheel speed
- RL wheel speed
- Transmission speed
- Gas pedal
- Steering wheel
- Car velocity
- Brake pedal
- Cube video source
- Gas-by-wire
- Steer-by-wire
- ABS
- Multimedia
- Engine control
- Car steering
- Car braking
- Display A
- Display B
- FL wheel brake
- FR wheel brake
- RR wheel brake
- RL wheel brake
Macro Fully-Specified Interface Model

- platform://resource/org.froemel.ttsoca.models/demo_mc/System.xmi
  - MFIM System
    - Component Control Loop DAS
    - Component simulation
    - Service speed
      - Port front left
      - Port front right
      - Port rear left
      - Port rear right
      - Port transmission
    - Service actuator steer
    - Service actuator brake
    - Service actuator gas
    - Component user io
      - Component ABS controller
      - Component steer controller
      - Mode normal
    - Component Multimedia DAS
    - Component cube generator
      - Service spinning cube
        - Port frame data, normal
        - Port frame data, negative
        - Port frame data, degradation level 1
        - Port frame data, degradation level 2
Off-chip communication

Heterogeneous components

Legacy component wrapping
Runtime Impressions
Conclusion

▸ Contributions
  ▸ Applied recursive component concept to TTSoC Architecture.
  ▸ Execution environment for virtual components.
  ▸ Scheduling of PDSes for TTNoC Interconnect.
  ▸ Viable application development approach:
    ▸ Uses models and model transformations.
    ▸ Based on concept of services and recursive component concept.
    ▸ Manageable integration of distributed applications.

▸ Outlook
  ▸ (Semi)automate optimal UFIM-PM allocation process.
  ▸ Validation of services’ behavior during refinement (e.g., by formal methods).
  ▸ Design of cooperating architectural elements s.t. they support recursive component concept.
Part III

End – Thank You!
Summary

- Basic concepts employed in the TTA
- Components, Information, Time, Interfaces
- Principles of the TTA
- Research projects overview
- Application development approach for the TTSoC Architecture, one of the latest TTA realizations
Credits

- Images:
  - http://en.wikipedia.org/wiki/Amdahl%27s_law
  - URL
References I

[1] **AMADEOS Consortium.**  
*AMADEOS D2.1 – Basic SoS concepts, glossary and preliminary conceptual model, 2014.*  

[2] **Bernhard Frömel.**  
An application development approach for the time-triggered system-on-chip architecture, 2013.

[3] **H. Kopetz.**  
*Real-Time Systems: Design Principles for Distributed Embedded Applications.*  

[4] **Hermann Kopetz.**  
A conceptual model for the information transfer in systems-of-systems.  

[5] **Hermann Kopetz and GÃ¼nther Bauer.**  
The time-triggered architecture.  