Building Safe, Highly-Automated Driving Applications with a Deterministic Subset of the AUTOSAR Adaptive Platform ARA API
What is the AUTOSAR Adaptive Platform?

AUTOSAR in a POSIX environment

- Usage of Linux for the development and execution of AUTOSAR applications
- Usage of open source software in AUTOSAR applications
- Introduction of technologies that are already used in the POSIX community for many years but not yet in AUTOSAR
  - C++
  - Service-oriented architecture (SOA)
  - Software-Over-The-Air (SOTA)

Goal: More modern, fast and innovative by using a POSIX environment

TODO: Define the automotive collaboration framework for this environment
What is the AUTOSAR Adaptive Platform?

But also POSIX in an AUTOSAR environment

- Deployment of ASIL relevant and/or real-time applications on POSIX-based operating systems

- Example: Highly automated driving (HAD) applications will run on POSIX-based ECUs
  - New features like multiprocessing, C++ and SOA are welcome
  - Because of the high safety demands, a deterministic behavior of the system is needed

Goal: A deterministic platform for ASIL relevant SW based on POSIX
TODO: Define the APIs of this platform in a way to enable deterministic implementations

ISO 26262

BOSCH
Applications on a deterministic platform will always run computations (that depend on each other) in the same order, and will produce the same output when starting conditions and inputs are the same.
Highly Automated Driving (HAD) has safety goals up to ASIL D

High computing performance demands can only be met with commercial µPs

Currently no µP hardware available for higher ASIL levels

Additional system level safety mechanisms required to support ASIL C / D functions

SW lockstep is one state-of-the-art approach to reach higher ASIL on commercial µPs

For a SW lockstep a deterministic platform is a must-have
Determinism Use Case: SW Lockstep

\[ A = f(In_1, In_2) \]
\[ B = f(In_3) \]
\[ Out = f(A, B) \]
The AUTOSAR Adaptive Platform ARA API

- The AUTOSAR runtime for Adaptive applications (ARA) provides the Application Programming Interfaces (API) for the development of Adaptive Applications (AA)

![AUTOSAR Runtime for Adaptive Applications (ARA) diagram]

- Adaptive Platform Foundation
  - API: Execution Management
  - API: Communication Management
  - API: Identity Access Management
  - API: Cryptography

- Adaptive Platform Services
  - Service: Update & Configuration Management
  - Service: Diagnostics
  - Service: Signal-2-Service Mapping
The AUTOSAR Adaptive service model

In the AUTOSAR Adaptive Platform a service

- is identified by a global unique service identifier (i.e. “com.oem.door_control_service”)
- can be available multiple times as different instances (i.e. “front_left”, “rear_right”)
- consists of:
  - 0..x events (i.e. “lock_status”) -> publish/subscribe communication
  - 0..y methods (i.e. “lock_request()”, “unlock_request()”) -> request/response communication
  - 0..z fields (data element that you can get(), set() or subscribe for to get events on change)
- can be discovered (“who has a “door_control_service”?)
- Is represented on the provider side by a skeleton and on the consumer side by a proxy
The ara::com API

- The ara::com API is the communication API for the AUTOSAR Adaptive Platform and its service model.
- It is based on C++11 and not bound to a specific communication protocol.
- It supports event, method, and field communication.
- It allows polling-based and callback-based interaction on the proxy side.

**Proxy Communication Access**

<table>
<thead>
<tr>
<th>event</th>
<th>method</th>
<th>field</th>
</tr>
</thead>
<tbody>
<tr>
<td>polling</td>
<td>callback</td>
<td>polling</td>
</tr>
<tr>
<td>polling</td>
<td>callback</td>
<td>polling</td>
</tr>
<tr>
<td>callback</td>
<td></td>
<td>callback</td>
</tr>
</tbody>
</table>
The ara::exec API

- In the AUTOSAR Adaptive execution management, two different strategies for the application internal execution control can be distinguished

<table>
<thead>
<tr>
<th>In the responsibility of the application</th>
<th>Externally controlled</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ Fully self-contained applications, no ara::exec API calls used for execution control</td>
<td>➢ Controlled execution with features provided by the ara::exec API</td>
</tr>
<tr>
<td>➢ The execution management does only the startup and shutdown of applications. Flexible runtime scheduling by using the PSE51 or C++ thread API</td>
<td>➢ Besides startup and shutdown, recurring (e.g. cyclic) activation based on external triggering (wait_for_next_activation() API)</td>
</tr>
<tr>
<td>➢ Free running threads with the possibility to have own triggers or timers</td>
<td>➢ Only one main thread and multithreading via a worker pool</td>
</tr>
</tbody>
</table>
SW Lockstep with full ARA solution space

Creation of loops with polling-based processing of event data

```cpp
while(true)
{
    // get samples
    // do computation
    std::this_thread::sleep_for(...100);
}
```

Usage of C++/POSIX multi-threading API and free running threads

```cpp
std::thread first(Thread1);
std::thread second(Thread2);
std::thread third(Thread3);
```

Data passing/sharing and triggering between threads

```cpp
std::condition_variable condVar;
condVar.notify_one();
```

Synchronous/asynchronous method calls

```cpp
Future<Call::Output> callFuture = proxy.Call();
```

Callback-based processing of event data

```cpp
proxy.event.SetReceiveHandler(MyCallback);
```
SW Lockstep with full ARA solution space

μP / Core 0

μP / Core 1

Different scheduling of free running threads
Different input data in the computations
Different internal data states
Different timing behavior of event callbacks
Different interaction of threads (time, data)
Different response times for method calls

No Way!
Definition of the deterministic ARA subset

- A deterministic behavior can be reached with a subset of the ARA API:
  - Polling-based event communication only (no methods and callbacks)
  - Activation management that sends a wake-up trigger based on the availability of input data (wait_for_next_activation() API)
  - One main thread and a flexible multithreading based on a worker pool. No access to shared data or communication between the worker threads

- Additional measures:
  - Restricted access to deterministic PSE51 API calls and deterministic memory allocation
  - Ensuring the sufficient availability of resources (memory, runtime)
SW Lockstep with deterministic ARA subset

- µP / Core 0
  - Wait for next activation
  - Data
  - Wait for next activation
  - Out 0

- µP / Core 1
  - Wait for next activation
  - Data
  - Wait for next activation
  - Out 1

0. Non-busy waiting for next activation
1. Wake-up on activation trigger
2. Polling-based access to the event data
3. Optionally, parallel computations in worker threads
4. Sending of event data
0. Non-busy waiting for next activation

Activation trigger

In 1n
In 2n
In 3n
Out 1n
Out 0n
Summary

- HAD systems have high safety demands and will use POSIX-based operating systems
- Safety critical applications must have a deterministic behavior (e.g. for SW lockstep)
- The ARA API provides a large solution space for communication and execution
- A deterministic behavior for Adaptive Applications is possible with an ARA API subset
- The deterministic subset of the ARA API includes polling-based event communication, externally controlled process activation and worker thread pools

AUTOSAR Adaptive can be THE deterministic platform for autonomous driving
Thank you for your attention!

Michael Pöhnl

Robert Bosch GmbH
CC-AD/ESW Engineering Software & Infrastructure

michael.poehnl@de.bosch.com
Tel: +491727653309