Resilience of Automotive Embedded Systems

Jean-Charles Fabre

Micheal Lauer, Matthieu Roy, Miruna Stoicescu (TSF team, LAAS)
Matthieu Amy, Helene Martorell (LAAS/Renault)
Philippe Quéré, Regis Valentin (TCR Renault)

Introduction

Resilient computing
Development process
Automotive systems
Conclusion

..... The future is exiting!

Open embedded systems
Very fast evolution

Autonomous connected cars
  ➢ Car-to-car, car-to-Cloud, C2I
  ➢ Remote maintenance

IOT everywhere on Earth
Drone taxi service
Single Pilot Operation

National Air and Space Academy workshops:
Will Air Transport be Fully Automated by 2050?
Introduction

Resilient computing

Development process

Automotive systems

Conclusion

..... The future is exiting!

Evolution as fast as possible

Agile development

Upgradable

Generic platforms

Over The Air Updates

Transient radiation effects on modern electronics

Remote maintenance, no massive recall

Business-model based on remote updates

Software complexity: 150 million lines of code in the F150 FORD Raptor

Extended set of software-implemented control functions

TIME-TO-MARKET !!!!!!!!!!

Is dependability a real concern today .... ?

Customer-Oriented Services

• Evolution & extended customization

Remote loading of new software-implemented functions (options – new feature, ADAS)

Multiple sensors + Multiples actuators ➔ opportunity for new functions

Sensor data fusion + (multifunction) actuators

Upgrade (£)

Updates
Manufacturer motivations

How often does your X get OTA updates? (Extrapolate if you’ve had the X less than 1 year.)

- About twice per month: 3 vote(s) 17.6%
- About once per month: 8 vote(s) 47.1%
- About once every two months: 6 vote(s) 33.3%
- About once every three months: 0 vote(s) 0.0%
- About once every six months: 0 vote(s) 0.0%
- Less than every six months: 0 vote(s) 0.0%

66 OTA over a one year period (May 2017 / May 2018)

Presentation outline

1. Introduction
2. Resilient computing principles
3. Development process
4. Application to automotive systems
5. Conclusion
Any system today is open and subject to changes in its operational life!

Can we anticipate all changes?

RESILIENT COMPUTING: Persistence of dependability properties when facing changes (J.C. Laprie – LAAS, 2008)

Evolution in response to change during service life is a key challenge for a plethora of systems, including critical systems.

Changes can originate from?

- user needs ➔ application software updates / upgrades (versioning)
- the environment ➔ unforeseen physical perturbations due to natural phenomena, operational phases conditions, component aging
- system resources ➔ added, removed or lost.

Changes ➔ Impact on FTM assumptions

- FTM update
- FTM substitution
- FTM composition

A key concept for resilient computing: Separation of concerns
Motivation for Resilient Computing

Protection mechanisms (EDM, ERM, FTM) are defined by risk analysis, FME(C)A and/or FTA

Validation is a big part of safety critical systems development processes.

A big worry: Validation vs time to market?

Changes → Impact on FTM assumptions
- FTM update
- FTM substitution
- FTM composition

A key concept for resilient computing: Separation of concerns

Challenges… and Q & A

Once the system is deployed, it faces changes.

System designers cannot predict everything.

Systems should be designed with adaptation in mind.

Why is it important?
Because changes may affect assumptions of FTM, possibly making them inefficient!
<table>
<thead>
<tr>
<th>Introduction</th>
<th>Resilient computing</th>
<th>Development process</th>
<th>Automotive systems</th>
<th>Conclusion</th>
</tr>
</thead>
</table>

**Presentation outline**

1. Introduction
2. Resilient computing principles
3. Development process
4. Application to automotive systems
5. Conclusion

**Proposed approach: Adaptive Fault Tolerance (AFT)**

Agile fine-grained adaptation of FTMs leveraging Component-Based Software Engineering (CBSE) techniques

Development process

- Change model
- Design for adaptation of FTMs
- Component-based implementation
- Transitions between FTMs

Adaptive Fault Tolerance
Introduction Resilient computing Development process Automotive systems Conclusion

Change model

What kind of changes and impacts?
An adequate/relevant FTM is based on:
1. Fault tolerance requirements (FT)
2. Application characteristics (A)
3. Resources (R)

FTM must remain consistent with the current (FT,A,R).
A change in (FT,A,R) → transition to a new FTM? Maybe!

Making assumptions (hypothesis) explicit is mandatory to determine the consistency of an FTM attached to an application A → effectiveness!

Adaptive Fault Tolerance

Classification of FTMs – FTM Design patterns

Classification regarding accidental hardware faults only

Classification regarding application characteristics

Resource requirements:
cost function based on # CPU, bandwidth, memory, energy consumption, etc....
→ order relation between FTMs → highly dependent on system configuration

Non exhaustive set of FTM
Extension regarding software faults, specific application faults, intentional faults / attacks
Fault tolerance design patterns

Development

application server with state management services

elementary fault tolerance mechanisms and fault detection mechanisms

composition of fault tolerance mechanisms

PBR=Primary-Backup Replication
LFR=Leader-Follower Replication
TR=Time Redundancy
PBR_A=Assertion&PBR
LFR_A=Assertion&LFR

Source Lines of Code

PBR=Primary-Backup Replication
LFR=Leader-Follower Replication
TR=Time Redundancy
PBR_A=Assertion&PBR
LFR_A=Assertion&LFR
**Introduction**

**Resilient computing**

**Development process**

**Automotive systems**

**Conclusion**

---

**Generic protocol execution scheme**

- **Caller component**
- **FTM**
- **Requested service**

**Runtime**

*Separation of concerns*

*Inspired by reflective computing & Aspect Oriented Programming*

---

**Two replication techniques examples (duplex inter-replica protocols)**

<table>
<thead>
<tr>
<th>PBR</th>
<th>Before</th>
<th>Proceed</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>State access assumption</td>
<td>Primary</td>
<td>Forward Rqt</td>
<td>Compute State checkpointing &amp; reply</td>
</tr>
<tr>
<td></td>
<td>Backup</td>
<td>Handle forwarded request</td>
<td>Nothing Process Checkpoint</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LFR</th>
<th>Before</th>
<th>Proceed</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Determinism assumption</td>
<td>Leader</td>
<td>Forward Rqt</td>
<td>Compute Notify &amp; Reply</td>
</tr>
<tr>
<td></td>
<td>Follower</td>
<td>Handle forwarded request</td>
<td>Compute Handle notify</td>
</tr>
</tbody>
</table>

---

**Transitions between FTMs / Runtime support requirements**

**Minimal API for on-line fine-grained adaptation**

- Graph of components at design time and at runtime
  - Control over component life cycle (add, remove, start, stop)
  - Control over interactions between components (connect, disconnect)
  - Components are stopped in a *quiescent* state
  - Incoming requests on stopped components are *buffered*

**The approach is reproducible on any support with these features.**
Adaptation process

- Systematic modifications of the component-based architecture
- New FTM variants are developed off-line and integrated on-line
- Transitions consist of 3 steps: deployment, script execution, removal

Presentation outline

1. Introduction
2. Resilient computing principles
3. Development process
4. Application to automotive systems
5. Conclusion
Application composed of SWC, a set of code fragments called runnables
The RTE is an Ad-Hoc communication middleware: set_lamp(), get_speed()...
The basic software, a runtime support including OS + services + drivers
Example of Upgrade

Runnable 1 → Runnable 2 - V.2 → Runnable 3

Example of Upgrade

Runnable 1 → Runnable 2 - V.3 → Runnable 3

AUTOSAR (AUTomotive Open System Architecture)
Implementation of the characteristics compatible, in particular from the point of view of the mode of activation and type. The temporal budget allocated to the container must therefore be added to the total execution time of the task. This approach is therefore quite pessimistic and if a large number of containers is added to the application, the CPU load remaining free in the initial application can be quite low. An example of a container placed in a task is given by Figure 3.2. On this figure, arrows represent communications. Given that each update has specific needs in terms of communication, the container does not have an arrow. The details of the practical implementation of containers are given later.

**Figure 3.2 – Example of task with a container**

**3.3 Development of a software embedded application: standard process**

When developing an embedded automotive application, a precise process must be followed. This process is based on a certain number of tools necessary to create an automotive application [106].

The figure 3.3 shows the different steps necessary to create an automotive application defined by Renault. It is a top-down approach, i.e. the start is the functional specifications to arrive at the code of the processors. This process respects the AUTOSAR methodology [15], but brings a certain number of additional contributions.

First, an additional step is added: it consists of performing a complete modeling from the point of view of the functional software architecture. This step is not necessarily specific to an AUTOSAR design. It is a global and high-level architecture of the software. Then, specific criteria for grouping runnables into software components are explicitly stated. On the other hand, criteria for allocating SWCs on ECUs and tasks are detailed. These elements do not appear explicitly in the AUTOSAR standard.

The first step of the development process is to identify the functional needs, i.e. which specifications must be fulfilled by our system and what characteristics it must have. When the needs have been identified, the development process can begin. This section describes the different steps of this process. First, how the global software model is created (step 2 of the process)

**AUTOSAR (AUTomotive Open System Architecture)**

**Off-line: Preparing application & warnings**

- Adaptation ➔ extra level of indirection: using smart pointers for adaptability within containers

- Mandatory careful safety analysis for determining all the implication of the update safety-wise, and consequently add or modify safety mechanisms.

- Last but not least, additional runnables imply modification of tasks WCET. The question is thus: « Is the new set of tasks schedulable? »
The AUTOSAR Architecture is too static

Limited space for adaptation (memory/time)

Adaptation spaces have to be defined in advance

What’s next?

– New evolution of the official standard ➔ Adaptive AUTOSAR

– Looking for another candidate ➔ our proposal based on ROS

Is ROS a good candidate for resilient computing and AFT in automotive embedded systems?

Several of projects (ranging from emergency stop assistance and autonomous highway driving to fully automated valet parking and 360° collision avoidance) were presented at the 2015 Consumer Electronics Show, and as it turns out, the cars were running ROS for both environment detection and planning.”

Michael Aeberhard (BMW): Automated Driving with ROS at BMW, May 31, 2016
At launch time, it is possible to reconfigure the name of any.

This end, a relevant ROS feature is its remapping capability. A means to insert functionality, such as safety or monitoring into an existing ROS application in the most transparent manner, is the ROS Master. It provides a means to register there services and topics to the ROS master. It is the interaction manager: 

- Topics / names
- Services
- Publishers
- Subscribers

When a node issues a service call, it queries the master for the address of the node providing the service and then it sends its request to this address.

Adaptive Fault-Tolerance: from a Component-Based approach to Resilient Automotive Computing on top of ROS.

**A. Introduction to ROS**

ROS can be viewed as a middleware running on top of a computation graph. When a node issues a service call, it queries the master for the address of the node providing the service and then it sends its request to this address.

Asynchronous interaction on topics

Synchronous interactions with services

Type of topics:
- local, denoted A2B for instance
- distributed, denoted /A2B for instance

**B. Component model and reconfiguration**

Two communication models are available in ROS: a publisher/subscriber model defines one-way, many-to-many, asynchronous communications through the concept of producer/consumer model and a client/server one. The publisher/subscriber model is used through a request/reply exchange between a node and its application in order to tolerate a crash fault of the node. One or more replicas can replace the faulty node. The FTM nodes, topics, and services are generic for every application in the context of ROS.

We present the behavior of each nodes, the topics/services in specializing the FTM discussed in section II. Implementing a FTM consists in rerouting requests and replies of the application in order to tolerate a crash fault of the node. The FTM computation graph is represented. Elements of the slave are suffixed with "S" before and "A" after protocol.

We identified a generic pattern for the computation graph of a FTM. Figure 1 shows the generic computation graph for FTM

**I. AFT on ROS Master**

- The CLIENT which is also hosting the ROS master, the MASTER hosting the primary replica and the SLAVE hosting the backup replica. For the sake of clarity, the symmetric roles will be denoted Server/Client and PROXY/FTM. The FTM nodes, topics, and services are generic for every application in the context of ROS.

**Generic computational graph for FTM on ROS**

(Boxes represent nodes)

**TOPICS:**
- pxy2pro
- pxy2bfr, bfr2prdsrv, prd2aft
- aft2pro
- pro2pxy
- aft2bfr

**SERVICES:** clt2pxy (client to proxy) and prd2srv (proceed to server)

**NODES:**
- proxy
- protocol
- before
- proceed
- After

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Resilient computing</th>
<th>Development process</th>
<th>Automotive systems</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROS basic concepts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROS Master</td>
<td>Interactor manager:</td>
<td>Topics / names</td>
<td>Services</td>
<td>Publishers</td>
</tr>
<tr>
<td>Topic=(name, sending port, receiving port, data type, …)</td>
<td>Asynchronous interaction on topics</td>
<td>Synchronous interactions with services</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type of topics:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- local, denoted A2B for instance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- distributed, denoted /A2B for instance</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Services:** clt2pxy (client to proxy) and prd2srv (proceed to server)
Combining FTM (PBR + TR) on ROS

Adding some security features

Authentication and key distribution
Crypto-based / Signature protocols
Introduction

Resilient computing principles

Development process

Application to automotive systems

Conclusion

Presentation outline

1. Introduction
2. Resilient computing principles
3. Development process
4. Application to automotive systems
5. Conclusion

Lessons learnt

Many motivations for Resilient computing

Evolution as fast as possible

One technique to this aim

Adaptive Fault Tolerance ➔ FTM differential updates

Over The Air Updates

Leveraging CBSE & dynamic runtime executive

Promoting

Design principles for adaptation

Separation of concern

FTM design patterns

Flexible runtime supports (ROS, Adaptive Autosar)

Analysis, selection and update strategies

Measures and tools to help designers
Conclusions at this stage

- **Safety vs fast evolution**
  - Users demand is growing....
  - Over-The-Air updates performing changes
  - Impact of evolution on dependability mechanisms

- **From Classic Autosar to Adaptive Autosar**
  - Separation of concerns
  - Component-based approach
  - Dynamic Binding

- **Measures and tools**
  - Similar to dependability measures
  - RC, RE, MTTI, MTBI, MTRI (I=Inconsistency)

- **Monitoring and proactivity**
  - Runtime monitoring, models,
  - Data collection & fusion, AI, Big Data

Now and beyond?


Efficient and Adaptive load balancing with spatio-temporal guarantees for connected vehicles
(e-Horizon project, Jean Ibarz, 2018-2021)

EE Dependable Real-time Architecture for Autonomous and Connected Vehicles
(CIFRE, Daniel Loche, 2017-2020)

Intrusion-Tolerant Authentication Mechanisms for Connected Vehicles
(e-Horizon project, Rémi Adelin, 2017-2020)
Now and beyond?

- In collaboration with other teams at LAAS
  - Alexandre Monti (2017), Sécurité des couches de communication bas niveau de véhicules connectés: analyse de vulnérabilités et protection, Continental, Projet e-Horizon, Dir. Thèse : D. Dragomirescu (MINC), E. Alata, M. Kaâniche, V. Nicomette
  - Bilel Cherif (2017), Développement de systèmes coopératifs mobiles fiables : cas d’étude pour la détection de collision automobile, Continental, Projet e-Horizon, Dir. Thèse: P. Berthou (SARA), Y. Labit (SARA), N. Rivière

- Starting soon
  - Test de logiciel pour des véhicules connectés, Continental, Projet e-Horizon, Dir. Thèse : H. Waeselynck, N. Rivière
  - Sécurité des systèmes embarqués de véhicules connectés, Continental, Projet e-Horizon, Dir. Thèse E. Alata, M. Kaâniche, V. Nicomette
  - Protection de la vie privée dans le contexte du véhicule connecté, Continental, Projet e-Horizon, Dir. Thèse M-O. Killijian, G. Tredan