Safe and secure software updates on high-performance mixed-criticality systems: the UP2DATE approach

Irune Agirre[a,*], Irune Yarza[a], Imanol Mugarza[a], Jacopo Binchi[a], Peio Onaindia[a], Tomasso Poggi[a], Francisco J. Cazorla[b], Leonidas Kosmidis[b], Kim Grüttner[c], Patrick Uven[c], Mohammed Abuteir[d], Jan Loewe[e], Juan M. Orbegozo[f], Stefania Botta[g]

[a] *Ikerlan Technology Research Centre, Basque Research and Technology Alliance (BRTA), Spain*
[b] *Barcelona Supercomputing Center, Spain*
[c] *OFFIS Institute for Information Technology, Germany*
[d] *TTTECH, Austria*
[e] *IAV GmbH, Germany*
[f] *CAF Signalling, Spain*
[g] *Marelli, Italy*

Abstract

Over-The-Air Software Updates (OTASU) are gaining popularity on the safety-critical domain. The motivation behind this trend is twofold. On the one hand, the ability of adding new functionality and services to the system without a complete redesign makes product makers more competitive and improves user experience. On the other hand, the increasing connectivity of emerging embedded devices makes OTASU a crucial cyber-security demand to keep the system up-to-date with latest security patches. However, the application of OTASU in the safety-critical domain is not straightforward, as they are not contemplated by current functional safety standards. The UP2DATE European H2020 project, seeks to provide solutions to cope with the challenging requirements of safety and security standards with respect to software updates. This paper gives an overview of UP2DATE, its foundations and the initial description of its safe and secure architecture that builds around composability and modularity on heterogeneous high-performance platforms.

Keywords: OTASU, mixed-criticality, safety, security, availability, heterogeneous computing

1. Introduction and Background

During the last decades, the safety-critical industry has been immersed in a process of continuous transformation. As a result, the number of critical functions realized by software is growing in conjunction with features that make systems more intelligent and autonomous [1, 2, 3]. However, this evolution calls for a dramatic paradigm change that replaces traditional simple, predictable and isolated proven-in-use safety-critical systems by complex and inter-connected solutions with higher computing power. This tendency brings several challenges for assuring the safety and security of the system. On the one hand, it gets increasingly harder to provide full evidence of the lack of residual faults in the system, due to uncertainties of the testing process and the unfeasibility to test all possible scenarios that may arise at system operation. On the other hand, the extended connectivity also brings potential threats that become cyber-security of utmost importance.

Current trends forecast that Over-The-Air Software Updates (OTASU) will play an important role for overcoming these challenges [4] by enabling the execution of regular remote updates for bug fixing,

*Corresponding author
Email addresses: iagirre@ikerlan.es (Irune Agirre), iyarza@ikerlan.es (Irune Yarza), imugarza@ikerlan.es (Imanol Mugarza), jbinchi@ikerlan.es (Jacopo Binchi), ponaindia@ikerlan.es (Peio Onaindia), tpoggi@ikerlan.es (Tomasso Poggi), francisco.cazorla@bsc.es (Francisco J. Cazorla), leonidas.kosmidis@bsc.es (Leonidas Kosmidis), kim.gruettner@offis.de (Kim Grüttner), patrick.uven@offis.de (Patrick Uven), mohammed.abuteir@tttech.com (Mohammed Abuteir), jan.loewe@iav.de (Jan Loewe), jmorbegozo@cafsignalling.com (Juan M. Orbegozo), stefania.botta@magnetimarelli.com (Stefania Botta)

adding new functionality and solving security vulnerabilities [5, 3, 6]. The automotive domain is a clear example of this trend, where over-the-air updates are being currently adopted for non-critical parts like the entertainment system [7, 8]. The next natural step is to extend OTASU to the whole vehicle, including safety-critical software, an area where for the moment most mainstream manufacturers stand aside. This is due to the potential severe consequences of their malfunction and the strict certification requirements of such systems [7, 6].

The UP2DATE project seeks to address the main dependability challenges brought by OTASU to the critical domain, with special focus on *safety*, *security*, *availability*, *maintainability*, and the increasing *platform complexity* of emerging heterogeneous Multiprocessor System on a Chip (MPSoC) devices.

This paper is an extension of a previous conference paper that summarizes the UP2DATE project (i.e., [9]) and adds details on its status and description of the architecture. The rest of this paper is organized as follows. Section 2 defines the project mission and its objectives. Section 3 describes the main innovations and concepts that have been extensively extended with the definition of an update cycle and its steps. Section 4 (completely new) details previous UP2DATE architecture definition and lists main safety and security features. Section 5 presents the industrial use-cases for the evaluation of the project and Section 6 (completely new) presents the project implementation details. Finally, the paper concludes with a summary of the project in Section 7.

2. Project mission and objectives

The mission of UP2DATE is bringing together the trend towards OTASU and heterogeneous computing platforms in Mixed Criticality Cyber-Physical Systems (MCCPS). To this end, UP2DATE has the technical objectives and the expected results explained below and represented in Figure 1. These objectives and results are related to three main components that comprise the UP2DATE architecture: a server, a mixed-criticality gateway and several end-devices.

O.1 Safety and Security (SASE) of complex platforms.

The first objective of the project is to lay the basis for the safe and secure integration of mixed-criticality applications on heterogeneous MPSoC

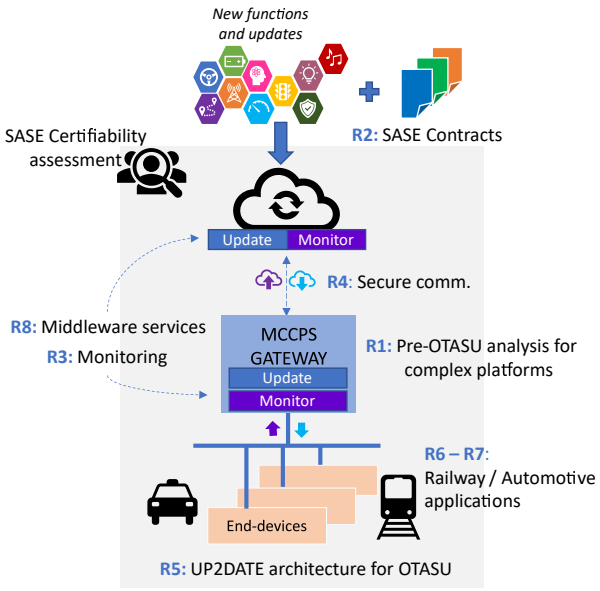


Figure 1: Main UP2DATE building blocks and results.

platforms. To this end, during the initial stages of the project, we analysed the safety and security design methods for such computing platforms. The activities related to this objective resulted in the definition of countermeasures according to safety and security standards and their evaluation on candidate project platforms like the Zynq UltraScale [10] or NVIDIA JETSON [11] ($R1$ in Figure 1). As outcome of these activities, the publication in [12] presents the evaluation and rationale behind platform selection and a second publication that is currently under review describes the safety and security analysis framework and results [13].

O.2 Design-by-contracts for modularity and composability.

The second objective rests in designing an update concept based on modularity and composability. This will allow doing partial software updates, while guaranteeing independence among multiple software components. To this end, UP2DATE will use the concept of *design-by-contracts* [14] ($R2$ in Figure 1), further explained in Section 3.1.1. Contracts will be part of the project update cycle (Section 3) and will serve to ensure that software components can be updated (i) without incurring unaffordable validation and verification costs; and (ii) ensuring that the SASE properties of the already integrated mixed-criticality software are preserved.

O.3 Observability, controllability and feedback strategies.

The project has the objective to develop observability and controllability solutions ($R3$ in Figure 1) that provide support for updates while ensuring safety and security for mixed-criticality tasks. On the one hand, observability will be implemented through event monitors present in most modern architectures [15], which will be compared against the specification and safe design thresholds. This information will be additionally examined to continuously improve the design. On the other hand, controlability seeks to adapt hardware and software configuration to the needs of the new update. On the initial project phase, we have started with the definition of the overall monitoring strategy, later presented in Section 3.3.

O.4 UP2DATE software architecture.

This objective seeks to implement previous concepts in the UP2DATE architecture ($R5$ in Figure 1). This architecture shall implement the required services to support the entire update cycle ($R8$ in Figure 1) together with best safety and security practices and requirements from standards, such as secure communications ($R4$ in Figure 1). The current project progress includes a high-level definition of this architecture, which is later presented in Section 4.

O.5 Industrial use-cases.

The fifth objective of UP2DATE is to evaluate the architecture and contributions in two real-world case-studies: railway and automotive ($R6$ and $R7$ respectively in Figure 1). A description of these use-case applications is given in Section 5.

O.6 Future safety and security certifiability.

The final technical objective of UP2DATE is to work with certification authorities towards the future certifiability of main UP2DATE concepts, architecture and solutions. Where needed, the project will propose updates to current standards or certification processes.

3. Concept and approach

This section, defines the overall UP2DATE concept and approach by defining an update cycle that has been conceived with the purpose of achieving previous objectives. This update cycle, depicted in Figure 2(a), is comprised of 10 steps classified into three main phases: (i) Design and release of updates, (ii) Update deployment phase and (iii) Runtime phase. In addition, Figure 2 shows how this update cycle is aligned with the IEC 62443-2-3 [16] technical report, entitled *Patch management in the IACS environment*, which defines the patch lifecycle model of Figure 2(b) [5, 17].

Next subsections will explain each of these phases, with special focus on the following key technological innovations that UP2DATE seeks to introduce to current OTASU approaches: (i) Safety and security contracts concept on the design phase (Section 3.1), (ii) Software update continuum for the deployment (Section 3.2), and (iii) Monitoring on the runtime phase (Section 3.3).

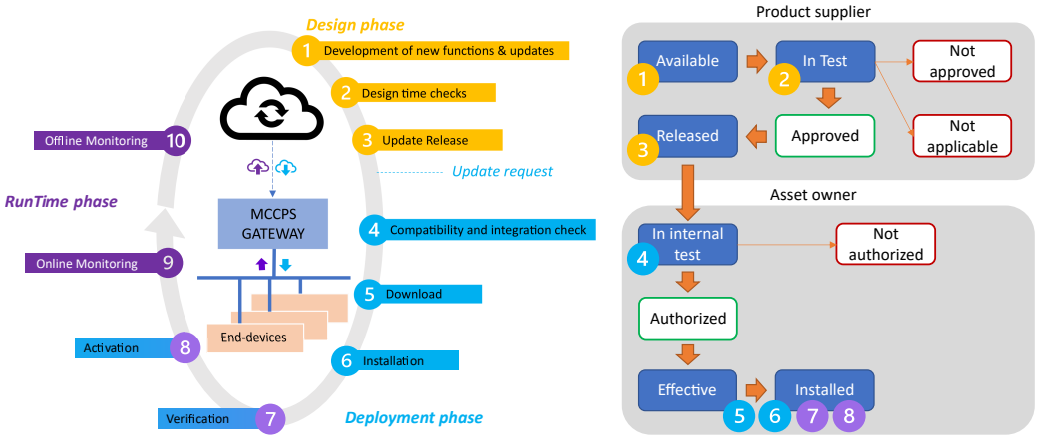
3.1. Design and release of updates

The UP2DATE project will advocate for modularity as the design method to facilitate software modifications in line with the recommendations of functional safety standards such as IEC 61508-3, clause 7.4.2.4 [18]. To this end, UP2DATE will adopt the design-by-contracts concept [14], where a set of SASE properties will be used to specify the dependencies of software components.

3.1.1. Step 1: Development of new functions and updates

The development of new functions and updates shall include the definition of Safety and Security properties associated to the different software components. These properties will be then implemented by contracts and compatibility checks that play an important role on the OTASU strategy, specially during the compatibility and integration checking step (Section 3.2.1) and runtime monitoring (Section 3.3). Until now, contracts remained in the realm of software, between two entities only, and static (i.e., used at design time and never changed during system life time). In contrast to common design-by-contracts approaches, in UP2DATE, contracts will be complemented with hardware resource requirements and safety and security constraints and their definition will evolve during system lifetime with new software updates. UP2DATE will integrate the definition of two types of SASE properties (see Figure 3):

- **Horizontal** cover the dependencies between connected software components (e.g., timing



(a) Update cycle for MCCPS.

(b) IEC 62443 Patch lifecycle model [5, 17].

Figure 2: UP2DATE cycle relation with IEC 62443 patch model.

and causality properties in a chain of software components).

- **Vertical** define the interfaces with the underlying system software and hardware (i.e., resource assignment). This is critical to ensure that there exists at least one acceptable configuration and environment to accommodate the new update, without causing unexpected interferences on the rest of the system.

3.1.2. Step 2: Design-time checks

As illustrated in the IEC 62443 process of Figure 2(b), all available software updates shall be subject to test before they are approved for release. Following the modularity approach, the project will assume that each software component, before being available for update, has been subject to strict verification and validation according to the lifecycle requirements defined by safety and security standards. The update cycle will then focus on evaluating the dependencies of the new software component with the rest of the system, both in terms of software and hardware dependencies as previously explained. It shall also be ensured that the update does not introduce regressions in the system [16]. In this step, an additional check, where the applicability to the update for the specific target platform and its configuration is evaluated, shall be performed. If passed, it will result in update approval or, otherwise, in an update rejection.

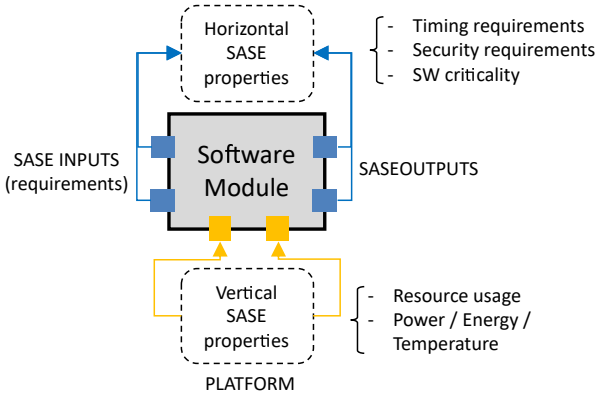


Figure 3: Horizontal and vertical SASE properties.

3.1.3. Step 3: Update release

Once the update is tested and approved, it shall be released for download to the target devices. For the software update release and distribution, the project considers two alternative approaches:

1. Polling: the gateway regularly asks to the server if new software is available (e.g., once a day).
2. On request: the server sends a notification to the gateway informing that a new update has been created and is ready for release. The gateway downloads and installs such update.

Commonly, the first approach is adopted as its implementation is usually simpler. Nevertheless, this approach presents scalability issues, including communication overheads and bandwidth waste. In line with this, a much more detailed and precise software update distribution control is possible fol-

lowing the second approach. In this case, the distribution and installation of the update can be accurately managed. To this end, a Software Updates Management System (SUMS) is commonly used, which may decide (also upon user request) which device will be updated with which specific software version and at which time instance. This method makes possible, for example, organizing updates according to the geographical location or market needs as well as simultaneous update downloads and upgrades of dependent systems.

3.2. Update deployment

In contrast to best security practices, regular software updates are not considered by current safety standards. Given that, the project will follow an incremental strategy for their deployment, covering a grey scale from simple to more complex software updates. This incremental strategy spans a design space defined over the three different axes of Figure 4:

- **Criticality** refers to whether the software element to update does have functional safety implications and security relevance. As a starting point, the project will consider non-critical software updates. In this case, the main challenge rests on guaranteeing that the update does not adversely impact to the critical functionality of the system. Then, the approach will be extended to critical software, considering additional mechanisms in the update cycle to demonstrate that the whole procedure is safe and secure.

- **Dynamicity** relates to the fact that updates can take place at different system states. In the simplest scenario, the update can be deployed when the system is not in operation (i.e., in a safe state). At the other end is the case for dynamic updates that take place while the system is under operation without the need of a halt and restart [19], improving overall system availability. Dynamicity has a direct impact on the update verification and activation steps of the update cycle of Figure 2(a), which for this reason, are represented in a different colour. In the first case, the verification and activation steps correspond to the deployment phase of the cycle. However, for dynamic updates, these two steps are effected at the runtime phase. At the end of this section, Table 1 defines the main differences of the update deployment phase depending on this dynamicity.

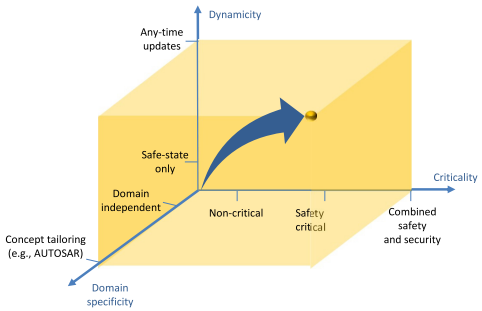


Figure 4: Software Update Continuum.

- **Domain specificity** has to do with the abstraction of the update architecture. On its initial phases, the project will focus on domain-independent approaches and the technical implementations will be tested through benchmarks and example applications. In final phases of the project the concepts will be then tailored to the automotive and railway domains through the case-studies of Section 5.

3.2.1. Step 4: Compatibility and integration check

According to IEC 62443-2-3, the update release shall be followed by internal tests performed by the asset owner before authorizing the update for installation (Figure 2(b)). In this step, it is required to confirm the compatibility of the new software with the running system and its configuration [20]. For this, the gateway will implement a self-aware integration checking service. This service will compare the SASE properties (e.g., timing, performance, resource assignment, energy consumption and criticality) of any new update against the current status of the MCCPS load before applying it. This check will guarantee system independence and composability, avoiding the need to test the whole system but just the updated component and its interfaces and dependencies with the rest of the system.

3.2.2. Step 5: Download

The download step relates to all required activities for the transfer of required software, data and files from the server to the gateway and from the gateway to the end-devices. In this process, security is of paramount relevance. Initially, the server and

gateway shall stablish a secure connection channel. To this end, a secure communication protocol, based on Transport Layer Security (TLS), shall be used, such as FTPS or HTTPS. In the handshake process, both the gateway and the server shall mutually be authenticated, for example, by means of digital certificates. Once the secure channel is established, the software update is downloaded into the gateway.

3.2.3. Step 6: Installation

After secure download, the installation process will store the received files in the reprogrammable program memory. This shall be complemented with a controllability service responsible of setting up the system to accommodate the new software (e.g., assigning new resources to the software component). The installation can be done either by placing the software image in a temporary memory location and then copying it to the corresponding address space at the later activation step or by creating a new location from where the new software will run after activation. Depending on the update severity and confidentiality, the information contained in the update package shall also be protected within the gateway, by means of encryption.

3.2.4. Step 7: Verification

At this point, the gateway checks the correctness of the new software installation and configuration before its activation. The verification will be realized in two flavours:

1. *Dynamic online checks with rollback:* The updated function is stimulated with test data to check pre-defined corner cases on the target platform, similar to built-in-self-tests. This check belongs to the deployment phase of the update cycle of Figure 2(a).
2. *Parallel and monitored operation in quarantine mode*: The "old" and the updated software components are executed in parallel for a predefined amount of time (i.e., in a *quarantine mode*) until it has been approved to be working correctly according to a predefined confidence level. This check will be performed at run-time.

3.2.5. Step 8: Activation

The activation is the last step to start running the newly installed software update. This step involves invalidating and storing a backup of previous software version and activating the new one. The activation can be done either at deployment phase or at runtime phase:

- *Offline activation:* the new image is copied to the target location replacing previous software image and it is activated through a reboot. This approach is usually quite slow due to the overhead required for copying the image and as it requires a reboot, this approach is only valid for offline updates where the system shall be in a safe-state.
- *Run-time activation:* in this case, software updates are dynamically triggered while the system is in use, without interrupting or stopping the program execution while the update is in progress. To this end, the new binary is stored and instantiated in a new memory location during the installation phase. For this purpose, state information transferring procedures are executed. The update manager just swaps the old image by the new one, once the quarantine period of the verification ends [21, 22].

The following Table 1 provides a summary of the main differences between the static and dynamic update deployment strategies for the affected steps.

Table 1: Differences on static vs dynamic update deployment

Step	**Static**	**Dynamic**
Installation	Temporary memory	Directly in Program memory (executing state transformations)
Verification	Dynamic online checks with rollback (at deployment)	Parallel and monitored operation in quarantine-mode (at runtime)
Activation	Offline: 1. Store old software 2. Copy new software 3. Reboot	Runtime: Swap to new program

3.3. Runtime monitoring

Monitoring is an useful approach that relies on observing the runtime execution behaviour of a target system in its final environment with different potential applications. In UP2DATE, runtime

monitoring will be used with the following 4 purposes:

1. Update validation: to get evidence that the updating procedure was successful and that it does not introduce regressions or affect on the operation or safety of the overall system.
2. Continuous safety monitoring: for preventing the system from entering an unsafe state, i.e., guaranteeing that it is operating within safe bounds (e.g., checking that the resource usage of a given software component is below a predefined safe upper-bound).
3. Continuous security monitoring: for identifying unexpected and unusual system patterns (i.e., anomaly detection) that could be caused by security attacks. In addition, for preventing from these attacks, these monitoring will include periodic vulnerability assessments through automated vulnerability checking tools.
4. Design optimization: to improve the design based on runtime information that serves to gain evidence on the real system behaviour and resource usage.

The overall strategy to accomplish previous monitoring goals is shown in Figure 5 and rests on observing internal system events and interactions for each software component at runtime and deriving the required SASE metrics out of them, which could be then evaluated either in real-time during the execution (i.e., online monitoring) or later on through offline monitoring depending on which of the aforementioned purpose is being accomplished.

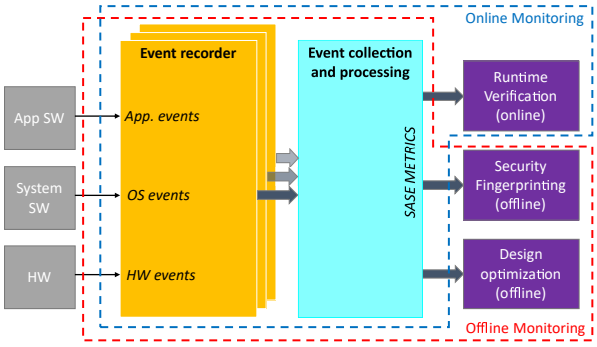


Figure 5: Monitoring strategy.

3.3.1. Step 9: Online Monitoring

The first two objectives of the monitoring, that is, update validation and continuous safety monitoring, require the runtime detection of faults in real-time, to perform dangerous fault detection and reaction before causing a hazardous event. Therefore, the runtime verification of Figure 5 will implement online monitoring to verify, based on runtime information, that the system meets its specification and that previously obtained SASE metrics are within their safe and secure range before, during and after an update.

3.3.2. Step 10: Offline monitoring

For security monitoring and design optimization instead, the runtime information will be sent to a remote server for offline automatic or manual inspection. For the former, the server will integrate existing monitoring tools from the security domain. For the latter, design optimization, the data will be made available to the system developer for an optimized system redesign.

4. UP2DATE Architecture

This section presents a high-level reference architecture that implements the previously described update cycle and its concepts. The overall architecture is comprised of the three main subsystems of Figure 6: a server, a MCCPS gateway and one or several end-devices that are logically separated into different network zones, in line with well-known security practices and standards. The numbers shown on Figure 6 refer to the update cycle steps of previous Figure 2(a). The definition of security zones is further explained in next Subsection 4.1, followed by a description of the three aforementioned subsystems.

4.1. Establishing zones and conduits

The overall system architecture is based on a defence-in-depth strategy, which is based on defining different security layers and protection measures. These protection barriers provide multiple, usually concatenated, security protections, with the aim of preventing and delaying any cyber-attack. To this end, assets are allocated into different security zones (depending on the required risk level and protection) and conduits are used to connect such zones. In addition, a target security level is assigned to each zone and conduit. The assignment of zones is the result of an initial cyber-security risk analysis for the reference architecture, which should be later tailored and detailed for specific use-cases. Those

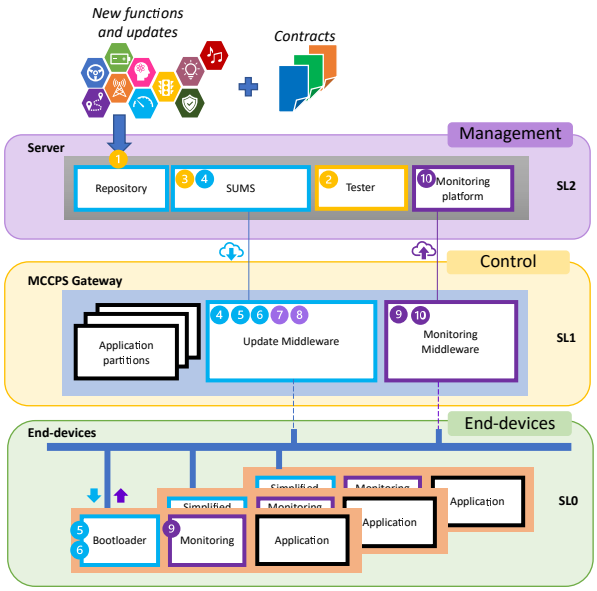


Figure 6: High-level update architecture.

assets within a zone share common security requirements and enable the implementation of common security countermeasures to mitigate the identified risks. In the scope of UP2DATE, the following reference zones, shown in Figure 6, have been defined based on IEC 62443 requirements:

- Management: industrial plant management services are included. In the scope of this project, it will include software update specific services such as the update and monitoring servers.
- Control: high-level industrial control and supervisory systems are included in this logical segment (i.e., the MCCPS gateway). The subsystems in this zone may execute both safety-related and non-safety-related functions and services.
- End-devices: safety-related I/O and low-level control devices (such as sensors, actuators, and programmable controllers) are included.

It shall be noted that for accesses to an external untrusted network (such as the internet) an additional zone that provides an isolated network layer between the Internet and the industrial network (i.e., the management zone) shall be set. This zone is often known as De-Militarized Zone (DMZ) and it is left out of project scope.

All zone boundaries are supervised and managed through firewalls, in which a security policy is enforced. In these security policies (in each firewall) all network traffic shall be denied by default, and only legitimate and required communications allowed. Within each zone, the corresponding communication protocols should be authorized.

4.2. Server

The UP2DATE architecture will integrate an update server and a monitoring server, which can be both integrated in the same device with a virtualized environment. The update server will be mainly controlled by the SUMS, which will integrate the necessary services to check available software updates on a repository, and execute the required design time checks before releasing them. The monitoring server instead, will collect continuous feedback information from the different gateways and make this data available for automatic or manual examination.

An important aspect of the server is its secure communication with the gateways or other networks. As defined in previous subsystem, the server and gateway are assigned to different security zones and their boundaries supervised. A secure communication should be established among such elements, which shall ensure at least the following security properties:

- Authenticity: ensure that the identity of both the gateway and the server are proved.
- Confidentiality: guarantee that the information sent is kept confidential by using encryption mechanisms.
- Integrity: detect any data manipulation during its transmission.

4.3. MCCPS Gateway

The MCCPS Gateway is the central element of the UP2DATE architecture for two main reasons:

1. It implements the update and monitoring middlewares, which are the main responsible to manage and run update specific services on multiple system devices. The gateway handles the connection with the server and connects to the system bus where multiple control devices are connected.

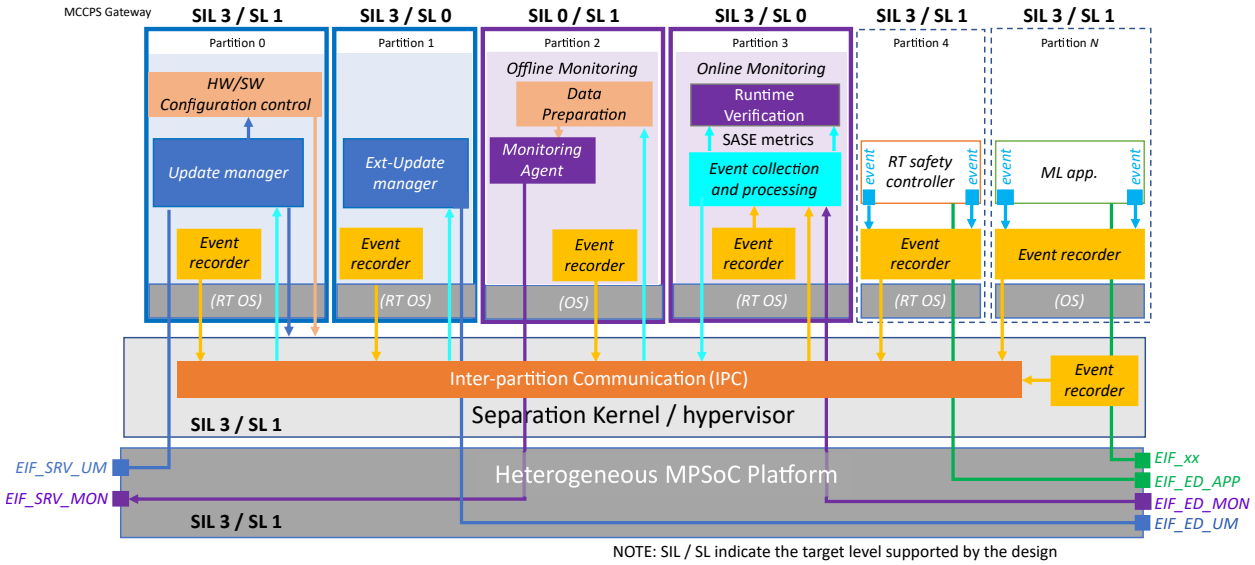


Figure 7: Partitioned MCCPS gateway architecture.

2. It integrates multiple mixed-criticality applications that in the past have been deployed into separate control units. This reduces the overall number of control units present in the system and brings several benefits such as improved integrity, scalability and flexibility [23].

In this section we present the high-level design practices and associated safety and security mechanisms adopted to combine mixed-criticality requirements together with the architectural needs of the update cycle.

4.3.1. Reference Gateway Architecture

Integrated mixed-criticality architectures usually have some common features, such as the integration of different system applications on top of a virtualization layer. Based on this approach, the gateway architecture depicted in Figure 7 has three main building blocks:

1. At the bottom of the architecture, the MCCPS gateway is based on emerging high-performance heterogeneous platforms that combine multicore CPUs with accelerators such as GPUs, FPGA or dedicated accelerators providing orders of magnitude higher performance than conventional control units.
2. In the middle, a separation kernel with virtualization features (i.e., hypervisor) will provide functional separation among mixed-criticality applications through virtual partitions. In addition, it allows the execution of multiple different operating systems on the same platform.
3. The different services that compose the conceptual update and monitoring middlewares of Figure 6 are allocated into different partitions (see Subsection 4.3.2). These partitions will command the hypervisor to provide the update and monitoring services to user applications.

The criticality level specified in Figure 7 refers to the target level (SIL-T /SL-T) of the UP2DATE gateway architecture, i.e., the maximum level supported by the system design. While the SIL and SL of each partition will depend on the final usage of the system and are application dependent, hardware layer and the hypervisor are shared among all partitions and the independence and incremental certification guarantees rely on them. Thus, the hypervisor and the platform inherit the maximum safety integrity and security level of the partitions present on the gateway.

4.3.2. Partitions

Partitioning is a key aspect for enabling the independent safety and security assessment and evaluation of each software component according to its associated safety-integrity and security level. This significantly reduces the impact of partial software updates on the system. From a safety point of view, partitioning confines the effects of design faults to

the partition where it originates and prevents their propagation to other partitioned applications by guaranteeing separation and design fault containment [24]. For security, partitioning is a common mechanism used to implement a Multiple Independent Levels of Security (MILS) architecture, where, if a partition gets compromised, its effects are restricted to the partition boundaries [21]. The purpose of each of the partitions of Figure 7 is the following:

- *P0 - Update Manager*: This partition is the responsible of managing the communication with the server and deploying software updates in the gateway itself according to the update-cycle steps, including system reconfiguration to adapt to the update. Therefore, this partition shall have higher privilege rights than other partitions. For this reason, this partition has the highest target integrity level of the system (SIL 3) and SL 1 due to its connection interface with the server (see Section 4.3.4).
- *P1 - Ext-Update Manager*: This partition manages and controls software updates on the end-devices. The inclusion of this partition between the update manager and the end-devices provides additional security ensuring that no partition has access to both server and end-device connection networks. As in the previous case, the target SIL of this partition is SIL 3 and regarding security SL 0 because it is separated from server side connections.
- *P2 - Offline Monitoring*: The main purpose of this partition is to transmit monitoring information from the gateway to the server. To this end, a monitoring client will be installed on the partition. The transmission of this data does not have a direct impact on system safety and therefore this partition is non-safety related (SIL 0). Regarding security, as it is connected to the server, its target level is SL 1.
- *P3 - Online Monitoring*: This partition will perform real-time fault detection by comparing the monitoring information with respect to the specification. For this reason, this partition is of highest target integrity level on the system (SIL 3) and SL 0 as its external interfaces are restricted to the connection with end-devices.
- *P4..N - Application Partitions*: These are optional user applications. For security, these partitions shall not be connected to the server network and their integrity level will be application dependant, considering SIL 3 and SL 1 as the maximum supported level.

For implementing the monitoring strategy of previous Figure 5, each partition and the virtualization layer integrate an event recorder. These event recorders will observe and transmit hardware, operating system and application events to the online monitoring partition (*P3*). With these data, *P3* will compute the SASE metrics for their evaluation through the online and offline monitoring approaches.

4.3.3. Resource allocation

Resource assignment and their usage are crucial to achieve the separation required by standards. This resource allocation is defined at design time and implemented through the configuration file of the hypervisor. Then, each software update shall be accompanied by the vertical SASE properties that specify the resource requirements of each component and by a new pre-defined configuration file. In order to detect failures in the resource allocation or usage, UP2DATE will implement multiple compatibility checks:

- Pre-update: for each resource used by a partition, it shall be ensured that the configured *resource budget* is sufficient to meet with the component's *resource requirements*. Moreover, it will be confirmed that the sum of every partitions' *resource budgets* does not exceed system's *resource capacity* for each resource.
- During and Post-update: At runtime, the *resource usage* of all partitions (including update and monitoring related partitions) will be monitored to guarantee that no partition exceeds its *resource budget* and that it fulfils its *resource requirements*.

Through this partitioning and resource allocation, the system provides resilience against resource exhaustion failures and attacks (e.g., (distributed) Denial of Service attacks) and is able to maintain essential services (e.g., safety functions).

4.3.4. Interfaces

The interfaces of the mixed-criticality gateway, depicted in Figure 7, can be classified into External Interfaces (EIF) and internal interfaces (referred as inter-partition communication (IPC)).

- **External interfaces** will connect the gateway with the server (EIF_SRV_xx), with the end-devices (EIF_ED_xx) and optionally with additional user-specific networks (EIF_xx). As illustrated in Figure 7, update and monitoring related partitions of the gateway already require a number of external interfaces to transmit the update packages and monitoring data.

- **Internal interfaces** (inter-partition communication (IPC)) channels will allow data transmission between different partitions. These channels shall be defined by the system integrator and managed by the hypervisor. The hypervisor shall deny all communication attempts on not configured IPC channels and control the access rights of each partition (e.g., read or write). The update and monitoring related partitions of the gateway already require a number of internal interfaces to share the update packages and monitoring observations. These internal interfaces, their purpose, associated partition (part.) and communication direction (dir.) are listed in Table 2.

Table 2: Partition connectivity table

Interface	Purpose	Part.	Dir.
IPC_UM	Update package	P0	B
	to end-devices	P1	
IPC_MON1	Monitoring information	P3	S
	to update manager	P0	D
IPC_MON2	Monitoring information	P3	S
	to Ext.-update manager	P1	D
IPC_MON3	Monitoring information	P3	S
	to the server	P2	D
IPC_MONP0	P0 events	P0	S
	for monitoring	P3	D
IPC_MONP1	P1 events	P1	S
	for monitoring	P3	D
IPC_MONP3	P2 events	P2	S
	for monitoring	P3	D
IPC_MONP4	P4..N events	P4..N	S
IPC_MONP*N*	for monitoring	P3	D

B = Bidirectional; S = Source; D = Destination;

4.3.5. Hardware access and privilege operations

The hypervisor is usually executed with higher privilege rights and manages the execution of privileged operations requested by the partitions. Therefore, in the same way as inter-partition communication interfaces, the hypervisor controls all accesses to the system hardware resources and prevents partitions from manipulating restricted registers. The hypervisor shall integrate a user control scheme to grant privileges only to users or partitions necessary to perform the intended operations. In the case of UP2DATE, the 'P0 – Update manager' partition needs higher privilege rights than other partitions to have the ability to control other partitions (e.g., stop, start, restart). Some commercial hypervisors already differentiate between "System Partition" and "user partition" with these different control rights.

4.4. End-devices

End-devices are a set of distributed embedded control subsystems or Electronic Control Units (ECUs) connected to the gateway. The gateway acts as the central distribution point for end-device updating. The execution of software updates on the end-devices brings a number of particularities that were not considered in the gateway update service, like the simultaneous update of several devices, the need to guarantee coherence among the different update files and versions and restricted amount of resources and stringent energy consumption and temperature requirements.

Nowadays, most ECUs already have some reprogramming capability, which is done in-situ by an operator commonly with dedicated tools to access the flash bootloader. UP2DATE will follow same strategy, with end-devices integrating a bootloader that will be addressed via gateway communications. In addition to the bootloader, the end-devices will also integrate monitoring services to send to the gateway the required information for update validation.

5. Use-case applications

The project outcomes will be evaluated in two industrial use-cases from the railway and automotive domains.

5.1. Railway use-case

The railway use-case sets its focus on the railway signalling product range. The end-device is a Safety Detector Manager (SDM), that gets information from several sensors along the railway line (lateral wind detectors and falling object detectors for instance), processes this information and sends it as consolidated data to the railway interlocking. The railway interlocking uses that information to protect the train routes in case any of the detectors raises an alarm.

5.1.1. Current update solution

In the *installation phase* of a Signalling System, it is frequently necessary to modify operating parameters or even some software functionalities. Such changes usually affect multiple devices (with different levels of criticality) along the trackside signalling and communication system. The update of such devices usually requires a team of specialized technicians to physically access them, which results in a costly and time-consuming procedure. When updating safety related devices, the physical presence of a member from the railway administration infrastructure manager is required. Moreover, the update procedure of safety related devices includes, besides upgrading devices, performing conformance tests and managing the corresponding documentation.

During the *maintenance phase*, uploads are less frequent, but given that the system is under operation, the impact is even greater. The update must be performed when the train service is suspended, late at night or during scheduled track maintenance cycles. When the update is performed at night, it might require several successive nights, where multiple coordinated teams simultaneously download test versions on every system device, carry out the tests and restore the previous approved version to re-establish normal operation. The estimated cost of loading one version is equivalent to 15 specialized technicians working during a whole night.

5.1.2. UP2DATE objectives

The UP2DATE architecture instantiation proposed for the SDM can bring great savings in the installation and maintenance phases of railway signalling devices. In fact, the main objective is to achieve a 50% long term cost reduction during the whole product life cycle. All in all, the integration of the UP2DATE platform in the railway signalling system will:

- Automatize this both costly and time-consuming updating procedure, especially in safety critical systems where the update procedure itself gets even more complicated.
- Reduce the team of technicians that usually work simultaneously in order to reduce the time needed for the update (thus maximizing testing time each night).
- Provide means to collect maintenance and performance information of the signalling devices to be able to monitor the operation of the system after an update.

5.1.3. UP2DATE railway architecture

The UP2DATE architecture instantiation proposed for the railway use case (see Figure 8) is based on the introduction of a gateway device that centralizes the update of all the individual devices of the signalling system without the need to physically access them. The gateway will communicate with a central server, located together with the current Traffic Management System (Control Centre), where operators schedule updates in a centralized manner.

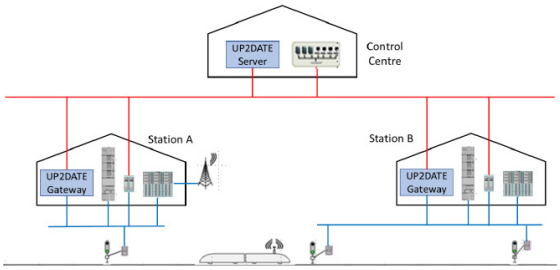


Figure 8: UP2DATE railway use-case architecture.

The UP2DATE railway architecture will integrate a simplified subset of the SDM. On the one hand, the gateway will integrate a simplified version of the safety detector manager together with the required update and monitoring services, and the end-device will implement a second instance of it. The simplified SDM includes safety critical and not safety critical software, which is integrated in the gateway.

5.2. Automotive use-case

The automotive industrial demonstrator takes place in the context of the next generation of connected automotive in-car computing architectures. The number of computerized ECUs, which is over a hundred in nowadays cars, is increasing with every new car generation, since every new function requires a new ECU, e.g., engine control, infotainment, driving assistance. The UP2DATE gateway seeks to reduce this amount of ECUs by centralizing them on a more powerful platform. Regarding the end-device, it is based on a Vehicle Domain Control Module (VDCM), an integrated platform for Powertrain and Vehicle dynamic control. The VDCM system is compliant with the ISO 26262 standard for Road Vehicle Functional Safety requirements.

5.2.1. Current update solution

As the quantity of code governing car functions increases, so does the need and frequency to update such code. In the post production and service phases a software update implies asking the user to go to a service point where a specialized technician updates the device in situ. This is clearly a costly and lengthy procedure which has a huge impact on the post-production and maintenance cost. In the case of safety related software, the effort and associated costs are even greater, since artefacts showing evidence of correct update must be produced and stored. Moreover, often recalls concerning safety features have a negative impact on car brand's image and can impact the market share of an auto-maker and even other auto-makers.

5.2.2. UP2DATE objectives

UP2DATE aims at reducing post production and maintenance costs by allowing remote updates. The long-term objective is to reduce the number of recalls and service campaigns up to 90% by enabling OTASU. To this end, the integration of UP2DATE technology in the automotive use-case will:

- Enable automatic updating operations in safe conditions without driving to a service point.
- Guarantee the fulfilment of safety and security requirements during remote software updates, even when updating safety related functions.

5.2.3. UP2DATE automotive architecture

As a solution, in the automotive use-case architecture presented in Figure 9, each vehicle is equipped with an UP2DATE gateway connected to a remote UP2DATE server. This automotive Centralized Computing Platform (CCP), manages updates and integrates several features of the vehicle (e.g., safety and/or non-safety critical functions classically placed in separated ECUs) in a centralized platform with high-computational power. Whenever an update is issued, the gateway distributes the information in a safe and secure way through the in-vehicle network to the corresponding end-devices.

6. Implementation

The work structure and responsibilities of UP2DATE are distributed into Work Packages (WPs) and a multidisciplinary consortium will ensure the achievement of the UP2DATE goals.

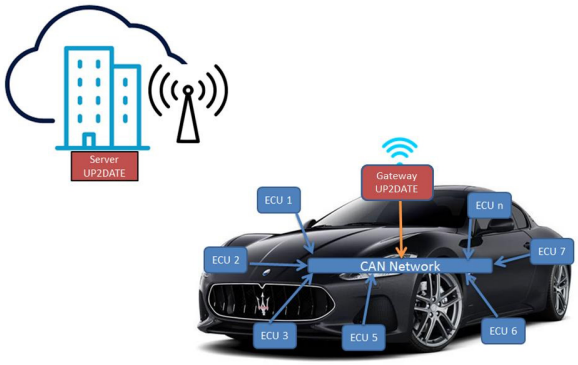


Figure 9: UP2DATE automotive use-case architecture

6.1. Work plan – Work packages

UP2DATE comprises 5 scientific and technological WPs and 2 other WPs devoted to management, dissemination and exploitation. The technological WPs cover from the UP2DATE concept definition up to its validation:

- *WP1* ensures an efficient and smooth project development and implementation according to the European Commission rules.
- *WP2* covers a detailed definition of the project requirements as well as the selection of the baseline platform on which the project will be implemented and evaluated, supported by an initial safety and security analysis of MCCPS on high-performance heterogeneous platforms.
- *WP3* develops the safety and security infrastructures, in accordance with the requirements coming from WP2, and covers the SASE assessment of the software update concepts by an external certification authority.
- *WP4* defines and implements the overall UPDATE OTASU middleware that includes

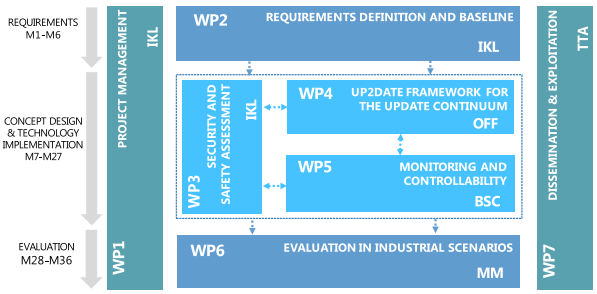


Figure 10: WP Structure.

contract-based services and defines the compatibility checks and integration testing to be carried out in each OTASU.

- *WP5* provides support to WP3 and WP4 in terms of observability, monitoring, controllability of the hardware configurations, security fingerprinting and budget optimization.

- *WP6* evaluates UP2DATE ecosystem through both, railway and automotive, industrial use-cases.

- *WP7* defines and implements an impact-driven, multi-stakeholder and multi-channel Communication and Dissemination strategy and develops an overall roadmap to the market to assess the exploitation plans of all partners.

6.2. Consortium as a whole

UP2DATE consortium consists of a multidisciplinary team of 7 partners from 4 EU countries (i.e., Spain, Germany, Austria and Italy), providing a multi-stakeholder approach to the project and covering the whole value chain through a combination of 3 Research and Technology Centres and 4 Companies.

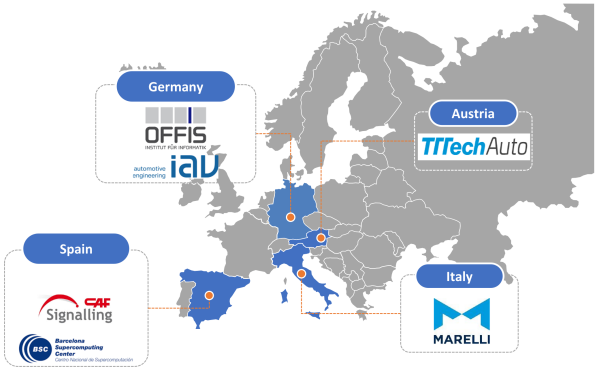


Figure 11: UP2DATE partners geographical.

In particular, the consortium counts with:

- Information and Communication Technology experts in the different project technological areas. Knowledge generators (i.e., IKERLAN, BSC, OFFIS) will address the research questions in terms of safety and security, design-by-contracts and monitoring of the software updates.

- Technology integrators (i.e., TTTechAuto, IAV) are developers of products and services to improve safety and reliability of electronic systems in the industrial and transportation sectors.

- End users (i.e., Marelli and CAF) will contribute to the validation of the technology in a real controlled environment proposing real world use-cases from the automotive and railway domain.

7. Conclusions

This paper has provided an overview of the UP2DATE H2020 project, which has the main goal of designing and implementing a reference architecture for the safe and secure execution of over-the-air updates on mixed-criticality MPSoC. At the time of writing, the project is in its concept definition phase, which has been the main focus of this paper. Based on current safety and security standards, the paper has described the update cycle and the architecture that will be implemented as a solution for enabling the safe and secure execution of software updates. As a summary, this architecture will rely on the design-by-contracts approach for checking the validity of an update before, during and after its execution. To this end, it will be complemented with runtime monitoring strategies that, apart from validating the update, will also serve to optimize system design after its deployment following an iterative development process.

UP2DATE will bring the OTASU technology closer to critical domains in two ways: first by evaluating the project outcomes in representative industrial scenarios from the railway and automotive domains and second by conducting a review of the main project concepts and design solutions with certification authorities.

Acknowledgments

The research of this paper has received funding from the European Union's Horizon 2020 research and innovation programme (grant agreement No 871465 (UP2DATE)). This work has also been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under grants PID2019-107255GB and FJCI-2017-34095 and the European Research Council (ERC), European Union's Horizon 2020 research and innovation programme (grant agreement No. 772773).

References

[1] A. Bhutani, S. Yadav, Global Market Insights. Automotive Electronics Market, report ID: GMI183, https://www.gminsights.com/industry-analysis/automotive-electronics-market (2018).

[2] S. Liu, J. Mu, D. Zhou, A. G. Hessami, Model safety standard of railway signaling system for assessing the conformity of safety critical software based weighted factors analysis, in: 2017 4th International Conference on Transportation Information and Safety (ICTIS), 2017, pp. 779–783. doi:10.1109/ICTIS.2017.8047856.

[3] K. Heineke, A. Ménard, F. Södergren, M. Wrulich, Development in the mobility technology ecosystem — how can 5G help?, https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/development-in-the-mobility-technology-ecosystem-how-can-5g-help (2019).

[4] S. Halder, A. Ghosal, M. Conti, Secure over-the-air software updates in connected vehicles: A survey, Computer Networks 178 (2020) 107343. doi:https://doi.org/10.1016/j.comnet.2020.107343.
URL http://www.sciencedirect.com/science/article/pii/S1389128619314963

[5] I. Mugarza, J. Parra, E. Jacob, Software updates in safety and security co-engineering, in: International Conference on Computer Safety, Reliability, and Security, Springer, 2017, pp. 199–210.

[6] J. R. Quain, With benefits — and risks — software updates are coming to the car, https://www.digitaltrends.com/cars/over-the-air-software-updates-cars-pros-cons/ (2018).

[7] M. Smith, Over-the-Air Updates - Is OTA the Future of Cars?, https://www.autocreditexpress.com/blog/ota-updates-the-future-of-cars/ (2018).

[8] S. Halder, A. Ghosal, M. Conti, Secure over-the-air software updates in connected vehicles: A survey, Computer Networks 178 (2020) 107343.

[9] I. Agirre, P. Onaindia, T. Poggi, I. Yarza, F. J. Cazorla, L. Kosmidis, K. Grüttner, M. Abuteir, J. Loewe, J. M. Orbegozo, S. Botta, UP2DATE: Safe and secure over-the-air software updates on high-performance mixed-criticality systems, in: 2020 23rd Euromicro Conference on Digital System Design (DSD), 2020, pp. 344–351. doi:10.1109/DSD51259.2020.00063.

[10] XILINX, Zynq UltraScale+ MPSoC, https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html (2020).

[11] NVIDIA, Jetson TX2 Developer Kit and Modules, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/ (2020).

[12] A. Jover-Alvarez, A. J. Calderon, I. Rodriguez, L. Kosmidis, K. Asifuzzaman, P. Uven, K. Grüttner, T. Poggi, I. Agirre, The UP2DATE Baseline Research Platforms, in: Proceedings of the Design, Automation & Test in Europe (DATE), 2021.

[13] I. Yarza, I. Agirre, I. Mugarza, J. P. Cerrolaza, Safety and Security Collaborative Analysis Framework for High-performance Embedded Computing Devices, in: Under review on Microprocessors and Microsystems, 2021.

[14] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, K. G. Larsen, Contracts for Systems Design: Methodology and Application cases, Research Report RR-8760, Inria Rennes Bretagne Atlantique ; INRIA (Jul. 2015).
URL https://hal.inria.fr/hal-01178469

[15] E. Mezzetti, L. Kosmidis, J. Abella, F. J. Cazorla, High-Integrity Performance Monitoring Units in Automotive Chips for Reliable Timing V&V, IEEE Micro 38 (1) (2018) 56–65. doi:10.1109/MM.2018.112130235.

[16] IEC, ISA/IEC 62443 series of standards on industrial automation and control systems security (2010).

[17] I. Mugarza, J. Flores, J. Montero, Security Issues and Software Updates Management in the Industrial Internet of Things (IIoT) Era., in: Sensors 2020, Vol. 20, 2020, p. 7160.

[18] IEC, IEC 61508 Functional safety of Electrical/Electronic/Programmable Electronic safety-related systems (Second edition) (April 2010).

[19] I. Mugarza, J. Parra, E. Jacob, Analysis of existing dynamic software updating techniques for safe and secure industrial control systems, International Journal of Safety and Security Engineering 8 (2018) 121–131. doi:10.2495/SAFE-V8-N1-121-131.

[20] UNECE, Draft Recommendation on Software Updates of the Task Force on Cyber Security and Over-the-air issues of UNECE WP.29 GRVA, Report, UNECE WP.29 (2018).

[21] I. Mugarza, J. Parra, E. Jacob, Cetratus: Towards a live patching supported runtime for mixed-criticality safe and secure systems, in: 2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES), 2018, pp. 1–8. doi:10.1109/SIES.2018.8442088.

[22] I. Mugarza, J. Parra, E. Jacob, Cetratus: A framework for zero downtime secure software updates in safety-critical systems, Software: Practice and Experience 50 (8) (2020) 1399–1424.

[23] S. Trujillo, A. Crespo, A. Alonso, J. Pérez, MultiPARTES: Multi-core partitioning and virtualization for easing the certification of mixed-criticality systems, Microprocessors and Microsystems 38 (8, Part B) (2014) 921 – 932. doi:https://doi.org/10.1016/j.micpro.2014.09.004.
URL http://www.sciencedirect.com/science/article/pii/S0141933114001380

[24] R. John, Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance, Tech. rep. (1999).