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

Irune Agirre*, Peio Onaindia*, Tomasso Poggi*, Irune Yarza*, Francisco J. Cazorla†, Leonidas Kosmidis†,
Kim Grüttner‡, Mohammed Abuteir§, Jan Loewe¶, Juan M. Orbegozo‖, and Stefania Botta**

*IKERLAN, Spain; †Barcelona Supercomputing Center, Spain; ‡OFFIS Institute for Information Technology, Germany;
§TTTECH, Austria; ¶IAV GmbH, Germany; ‖CAF Signalling, Spain; **Marelli, Italy

***Abstract*—Following the same trend of consumer electronics, safety-critical industries are starting to adopt Over-The-Air Software Updates (OTASU) on their embedded systems. The motivation behind this trend is twofold. On the one hand, OTASU offer several benefits to the product makers and users by improving or adding new functionality and services to the product without a complete redesign. On the other hand, the increasing connectivity trend makes OTASU a crucial cyber-security demand to download latest security patches. However, the application of OTASU in the safety-critical domain is not free of challenges, specially when considering the dramatic increase of software complexity and the resulting high computing performance demands. This is the mission of UP2DATE, a recently launched project funded within the European H2020 programme focused on new software update architectures for heterogeneous high-performance mixed-criticality systems. This paper gives an overview of UP2DATE and its foundations, which seeks to improve existing OTASU solutions by considering safety, security and availability from the ground up in an architecture that builds around composability and modularity.**

***Keywords*—OTASU, mixed-criticality, safety, security, availability, heterogeneous computing.**

I. Introduction and Background

In recent years, the safety-critical industry has witnessed a striking increase of critical functions realized by software [3], [15]. The automotive domain is a clear example of this trend, where a growing number of Advanced Driver-Assistance Services (ADAS) are already embedded in cars in conjunction with features that make vehicles more intelligent and autonomous [9]. This results in a dramatic increase of software complexity that inevitably requires hardware platforms with higher computing power. As a direct consequence, it is more and more difficult to guarantee at design time that all security flaws and system errors associated to the electronic system component are prevented or duly mitigated in such a way that the system will be sufficiently safe and secure at operation time. These challenges lead to the need of continuously improving the system based on the experience acquired over its service-life. In this regard, the advent of new connection technologies (e.g., 5G) bring a new range of capabilities such as exchanging real-time data about the system and the environment and downloading regular software updates [9], [20]. Unfortunately, the extended connectivity also brings potential cyber-security threats and it is essential to keep the system up-to-date with the latest security patches.

Over-The-Air Software Updates (OTASU), a common technology in consumer electronics like the mobile market, allow the execution of remote updates from user-level applications down to firmware and offer several benefits in terms of bug fixing, adding new functionality and solving security vulnerabilities [18]. These benefits make OTASU a key technology to stay competitive in many safety-critical markets. For instance, in the automotive domain it is not rare to find automakers that perform regular over-the-air updates of the entertainment system software [21]. 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 [21], [20].

The UP2DATE project seeks to address the main dependability challenges brought by OTASU to the critical domain, with special focus on *safety*, *security*, *availability*, and *platform complexity*:

- **Safety:** The current practice to guarantee functional safety is by strict development processes usually dictated by safety standards that aid in achieving system certification prior to operation. Once the system is certified and deployed, modifications are rather uncommon, as they might involve high re-certification efforts and costs. However, OTASU allows performing regular software adaptations and often demands system re-configuration (e.g., task schedule, resource allocation, Operating System (OS) upgrades). Current safety standards do not capture OTASU and there is not a clear procedure to guarantee that the system remains safe after the update.
- **Security:** While OTASU eases the application of security patches and it is a practice required by security standards [18], they shall be applied with extra care to prevent the downloading of fraudulent software into the system.

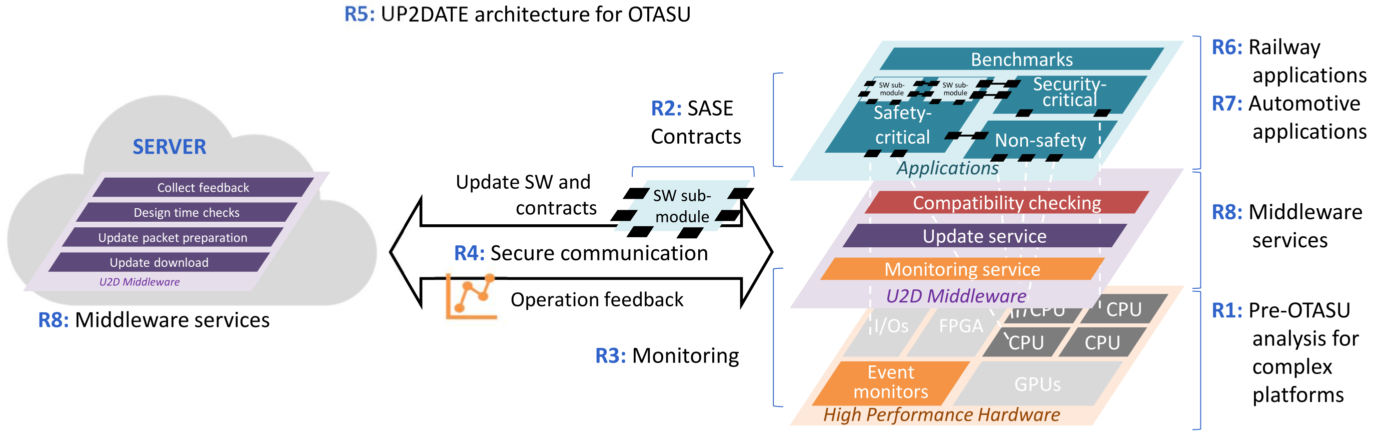


Fig. 1. Main UP2DATE project building blocks and results.

This may not only affect the security of the system but also its safety. Accordingly, it is essential to address these issues with a safety and security co-engineering approach.

- **Availability:** During updates the system is commonly not available. While this is just inconvenient for mainstream devices, it may not be acceptable for fail-operational systems that must remain active during operation (e.g., a pacemaker). Therefore, OTASU shall consider *dynamic software updates*, to safe and securely update software at runtime, without stopping or restarting the system.
- **Platform complexity:** The increasing demands of the software can only be reasonably satisfied by using more complex hardware platforms based on multicore processors and accelerators (e.g., Graphics Processing Unit (GPU)). The higher computing performance provided by these platforms allow integrating several control units, often of different safety and security implications (a.k.a. *mixed-criticality*), in one single platform. In such complex Mixed Criticality Cyber-Physical Systems (MCCPS), applications are subject to intricate dependencies in their functional and non-functional behaviour. Both have a negative impact in system safety and security.

This paper summarizes the UP2DATE project for simultaneously addressing these challenges in a novel software architecture to support OTASU. The rest of this paper is organized as follows. Section II defines the project mission and its objectives. Section III describes the main innovations and concepts around which the UP2DATE architecture of Section IV will be built. Section V presents the industrial use-cases for the evaluation of the project. Finally, the paper concludes with a summary of the project in Section VI.

II. Project Mission and Objectives

The mission of UP2DATE is bringing together the trend towards OTASU and heterogeneous computing platforms in MCCPS. Accordingly, the overall goal is to develop a new software paradigm that guarantees the safety and security of a mixed-criticality system able to execute regular software updates. To this end, UP2DATE has the following technical objectives and the expected results represented in Figure 1.

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

Since the design of applications running on heterogeneous computing platforms is not well covered by current safety and security methodologies, the first objective of the project is to lay the basis for the safe and secure integration of mixed-criticality applications on such platforms. This objective will focus on the identification of *safety risks* and *security threats* of mixed-criticality systems and on the definition of countermeasures according to safety and security standards. This objective will result on a evaluation of the feasibility of implementing these countermeasures on specific heterogeneous platforms like the Zynq UltraScale [24] or NVIDIA JETSON [19] and permit, in this way, a safe and secure mixed-criticality integration ($R1$ in Figure 1).

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

The second objective of UP2DATE is to design an update concept based on modularity and composability. The availability of a modular and composable update architecture enables doing partial software updates, while guaranteeing independence and a shift in the concept of integration testing: from design time to online integration testing or even self-aware integration testing. To this end, UP2DATE will use the concept of *design-by-contracts* [2] ($R2$ in Figure 1), further explained in Section III-A, to ensure that software modules can be updated (i) without incurring unaffordable validation and verification costs; and (ii) ensuring the SASE properties of the already integrated mixed-criticality software.

O.3 Observability, controllability and feedback strategies.

During and after the deployment of a software update, the project has the objective to monitor the safety and security properties, adapt hardware and software configuration to the new update and to collect operation time behaviour data to optimize the design ($R3$ in Figure 1). To this end, UP2DATE will rely on event monitors present in most modern architectures [16] to check at runtime if the SASE contracts of the update are satisfied. This information will be additionally analysed to constantly improve the design in the new updates based on operation time data.

O.4 UP2DATE software architecture.

This objective seeks to implement previous concepts in the UP2DATE architecture ($R5$ in Figure 1) comprising:

- *Secure communication library* ($R4$ in Figure 1): security mechanisms that allow secure over-the-air transmission of the update and operation time data.
- *U2D middleware* ($R8$ in Figure 1): services to support the entire update cycle starting with the update packet preparation and over-the-air download at server side and i) compatibility and integration checking, ii) update deployment and iii) monitoring, controllability and feedback services at target side.

O.5 Industrial use-cases.

The fifth objective of UP2DATE is to evaluate the UP2DATE architecture in a laboratory-controlled environment applied to two real-world case-studies: railway and automotive ($R6$ and $R7$ respectively in Figure 1).

O.6 Future safety and security certifiability.

The final technical objective of UP2DATE is to carry out an *assessment* of the future certifiability of main UP2DATE concepts and architecture. In other words, this objective seeks to evaluate with certification authorities that the designed dynamic software update technology does not compromise the safety and security properties of the MCCPS. Where needed, UP2DATE will propose updates to current safety standards to consider the proposed technology.

III. Key UP2DATE Innovations

UP2DATE will reach previous objectives by introducing the following key technological innovations to current OTASU approaches.

A. Safety and Security (SASE) contracts concept

UP2DATE will adopt the design-by-contracts concept [2] to allow the execution of modular and composable updates without compromising overall system SASE properties. Until now, contracts remained in the realm of software, between two entities only, and static (i.e., used at design time only, specified once and never touch during system life time). UP2DATE will work towards an innovative use of contracts supporting the safe and secure update and monitoring of software components and its associated contracts during the operation phase of a system. In contrast to common design-by-contracts approaches, in UP2DATE, hardware resource allocation requirements and safety and security constraints become an integral element of the contracts. To this end, UP2DATE defines the concepts of horizontal and vertical contracts (see Figure 2):

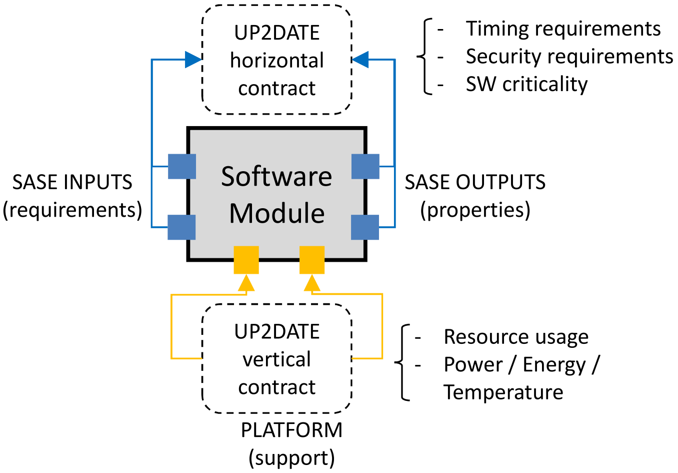

UP2DATE horizontal contract
- Timing requirements
- Security requirements
- SW criticality
SASE INPUTS (requirements)
Software Module
SASE OUTPUTS (properties)
UP2DATE vertical contract
- Resource usage
- Power / Energy / Temperature
PLATFORM (support)

Fig. 2. Horizontal and vertical SASE contracts.

- **Horizontal contracts** cover the dependencies between connected software components (e.g., timing and causality properties in a chain of software components).
- **Vertical contracts** define the interfaces with the underlying system software and hardware to ensure that software modules get access to the resource they need to fulfil its SASE requirements.

Contracts will be used as part of the UP2DATE OTASU strategy, to compare the SASE requirements (e.g., timing, performance, energy consumption and criticality) of any new application update against the current status of the MCCPS load before applying the update. The innovative combination of horizontal and vertical contracts permits the execution of *modular updates* able to replace individual functions and not necessarily the entire firmware or control unit. In addition, it is done in a *composable* manner, that is, guaranteeing independence and avoiding the need to test the whole system but just the updated module.

B. Software update continuum

UP2DATE defines the software update continuum of Figure 3 to cover a grey scale from a simple to more complex OTASU. The complexity of performing an update depends on its location on the update continuum, that spans a design space defined over the three different axes of Figure 3:

- **Criticality** refers to whether the software element to update does have functional safety implications and security relevance. In the case of non-critical software update, the main challenge rests on guaranteeing that the update does not affect the critical software running on the system. To this end, UP2DATE will rely on the SASE properties defined in the horizontal and vertical contracts. For critical software, in addition to this, the project will need to focus on safety-security co-engineering to define the procedures to test, validate and update safety software.
- **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 in a safe

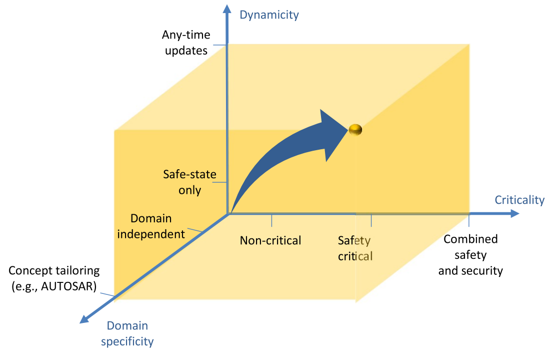


Fig. 3. Software Update Continuum.

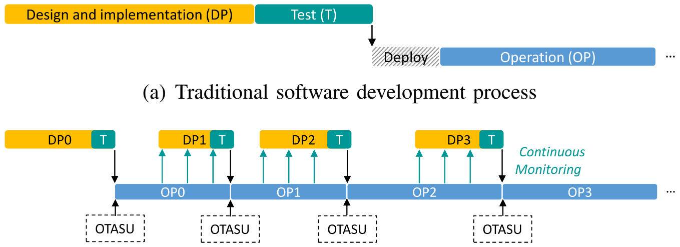
state (e.g., a vehicle is stationed). At the other end is the situation of dynamic updates that take place while the system is under operation without the need of a halt and restart. The timing of the update has a direct impact on system availability. UP2DATE will work on solutions to keep critical tasks available during the update process.

- **Domain specificity** has to do with the abstraction of the update architecture. While the project will first focus on domain-independent approaches, in final phases of the project the concepts will be tailored to the automotive and railway domains.

The safe and secure OTASU paradigm promoted by UP2DATE will incrementally combine these three properties together with next generation heterogeneous platforms. While there are extensive works addressing OTASU with any of these elements separately [17], [18], [22], UP2DATE will rely on the contract concept to build an update framework capable of handling all these aspects together. Additionally, UP2DATE will study how to exploit the higher computing performance properties of heterogeneous platforms to enable dynamic updates and their implications on safety and security.

C. Development and Operations (DevOps) practice

Traditionally, the safety-critical development process has followed a sequential approach as that illustrated in Figure 4(a): it starts with requirements specification, followed by design and implementation of the complete solution and finishing with testing prior to its release and deployment. In recent years, however, it has become evident that the complexity of software functions and hardware used in MCCPS is creating significant issues for creating an efficient design and guaranteeing a comprehensive testing of all possible conditions that can arise at operation time. Software timing analysis is a clear example of the limitations of the current development process: guaranteeing the worst-case execution time of a function without being overly pessimistic has become an arduous task where measurement-based approaches seem to be the only viable solution [23]. Still, in complex platforms, timing behaviour cannot be verified to a sufficient extent with end-to-end measurements due to complex interactions among hardware resources [16].



(a) Traditional software development process

(b) DevOps software development process

Fig. 4. Traditional vs. DevOps development practices.

For this reason, it is becoming increasingly common to combine operation time data with the design phase to improve the design and the testing of the software. In UP2DATE, this single design phase approach illustrated in Figure 4(a) will be replaced by the interactive process depicted in Figure 4(b) that relies on event monitors for enhancing the testing with runtime checks and uses the collected operation time data to optimize the non-functional properties of the applications (e.g., timing and power consumption). This process matches well with the recently proposed DevOps software development practice that combines software development (Dev) and information technology operations (Ops) to shorten the system development cycle [10].

In this interactive process, every OTASU is preceded by a Design Phase (DPi) in which the required assessments are performed on the validity of the update. In case it is accepted, in the next Operation Phase (OPi) the system will deploy the software update. Note the concurrent nature of the phases: the design phase $DPi + 1$ is carried out during the execution of OPi, without halting the system and with the continuous feedback from the application and platform states. To that end, the information collected during OPi will be sent to the backside to assess whether tasks are using their timing or Energy, Power and Temperature (EPT) budget. This information will be obtained by event monitors that capture the exact execution conditions in which the task is executed, as opposed to the pessimistic ones created during system tests at design time. The usage of event monitors in the software design is still in its infancy, with some techniques using few of them to regulate for instance traffic to memory [16]. However, current hardware platforms have in the order of hundreds of event counters that can be used to gain more confidence on derived bounds [4]. This provides very valuable and reliable feedback on whether derived bounds are over-estimated. In such scenario, the UP2DATE architecture will safely decrease those bounds allowing to optimize the contracts and augment the load that can be safely put on the system.

IV. UP2DATE Architecture

Previous innovative concepts will be consolidated in the form of a software architecture supporting high-performance hardware platforms. Figure 5 presents the UP2DATE architecture comprised of the following six main building blocks.

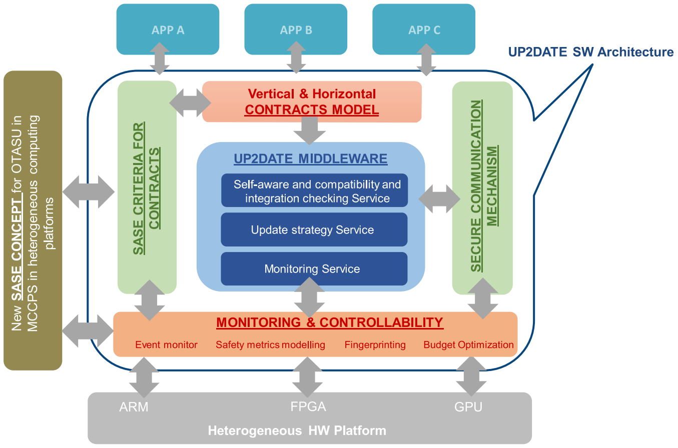

APP A
APP B
APP C
UP2DATE SW Architecture
New SASE CONCEPT for OTASU in MCCPS in heterogeneous computing platforms
SASE CRITERIA FOR CONTRACTS
Vertical & Horizontal CONTRACTS MODEL
UP2DATE MIDDLEWARE
Self-aware and compatibility and integration checking Service
Update strategy Service
Monitoring Service
SECURE COMMUNICATION MECHANISM
MONITORING & CONTROLLABILITY
Event monitor
Safety metrics modelling
Fingerprinting
Budget Optimization
ARM
FPGA
GPU
Heterogeneous HW Platform

Fig. 5. UP2DATE software architecture components.

A. New SASE concept for OTASU in mixed-criticality heterogeneous computing platforms

This first building block relates to the safety and security certifiability of the UP2DATE approach and takes a pivotal role in the project, as the safety and security concerns and solutions worked throughout this module drive the activities across all other components and shape the UP2DATE architecture. This activity is motivated by the rapid evolution of hardware and software in MCCPS, which has surpassed the capabilities of current safety and security oriented design methodologies. Generally, functional safety standards reflect the state of practice in industry rather than the state of art. As a result, they do not evolve as fast as technology and they do not provide explicit guidance for next generation architectures yet [1]. UP2DATE will pave the way towards mixed-criticality integration and certification in next generation heterogeneous computing platforms and will reconcile the conservative safety-related requirements coming from standards with the disruptive nature of OTASU.

To this end, UP2DATE will follow an incremental strategy to define safe and secure design concepts and the required argumentation and evidences to support the future certification of the UP2DATE architecture. The safety-security concepts will define a safe and secure architecture based on arguments for mixed-criticality integration, dynamic updates, contracts, platform monitoring and diagnosis. These concepts will undergo a review process by an external certification authority with respect to selected industrial functional safety (e.g., IEC 61508 [11], ISO 26262 [13], EN 5012x [5], [6], [7]) and security (e.g., IEC 62443 [12]) standards.

B. Vertical and horizontal contracts model

Contracts are associated to an executable of a software module and they contain metadata that specifies its functional (e.g., allowed value ranges) and non-functional (e.g., causality, timing, resource usage or power consumption) requirements on the environment in which the module shall be integrated. UP2DATE builds upon the following modularity properties of the software:

- Logical architecture: the connection (dependencies and interaction) among functions and their decomposition (i.e., a function contains sub-functions).
- Technical architecture: the mapping of functions to software components and the relation (and configuration) of software components to a run-time environment (e.g., AUTOSAR). Furthermore, it specifies the allocation of software components to hardware resources.

In UP2DATE these properties are managed in two axes: the horizontal and vertical contracts respectively. In the execution of a software update, *horizontal contracts* describe the integration with the remaining (not updated) system. It consists of assumptions on the activation of input data from neighbouring components and provides guarantees about the activation of its output data to other neighbouring components. *Vertical contracts* describe “constraints” under which the horizontal contracts shall be fulfilled. They include information about required communication, configuration, computation and memory resources on the target hardware platform.

As a key property, SASE contracts are composable, allowing their application in a hierarchical manner to different sub-modules (or sub-applications) and from there derive the properties for the umbrella module (application). This heavily reduces the computation needs for the validation and provides scalability of the UP2DATE approach because integration tests only have to be conducted for the updated module and not for the entire system.

C. SASE criteria for contracts

In addition to the logical and technical architecture presented in previous subsection, UP2DATE will also integrate safety and security constraints into the contracts. This aims to guarantee that neither safety nor security are jeopardized upon the execution of an update. Adopting these non-functional properties in the contracts allows supervising these properties at runtime (before, during and after an update) by associated monitoring services (see Section IV-F). In particular, this module covers the definition of:

- *Safety properties* such as timing, energy, power, temperature, time and space independence needs and resource usage. The formal definition of these properties will be guided by i) their relevance for the assessment of non-functional safety according to standards; and ii) the possibility to measure them with monitoring services.
- End-to-end *security properties* for OTASU, starting from the download of the update up to its deployment and sending back the monitoring data. In addition, the security contracts will be used to define fingerprints that capture the nominal timing and energy profile of applications. Then this profile will be constantly checked against the behaviour of the application at operation time to detect possible intrusions.

D. UP2DATE middleware

The UP2DATE middleware comprises services to support the entire update cycle starting with the over-the-air download of the update from a server, the target platform specific update compatibility and integration testing, the deployment of the update on the target platform, and finally the update monitoring. This process entails the following minimum services:

- **Compatibility and integration checking service.** Before deploying an update on the target platform, an update acceptance check must be executed. This service defines a self-aware contract-based integration checking in the update continuum. To this end, it will test the compatibility of the downloaded update with the specific target system and its configuration.
- **Update service.** Once the update is accepted, UP2DATE will support two alternative deployment strategies:
    1) *Online deployment:* the update is deployed on the target platform when the system is in a safe-state (i.e., currently not used but online). It replaces an existing software component or functions with a new function (e.g., by replacing the program or flashing the entire data and instruction memory). Store and restore of state information across function updates will be handled by a dedicated state restoration service. Deployment information (e.g., in which partition(s) or on which processor(s) the updated functions shall be executed) is provided through vertical contracts within the update metadata.
    2) *Run-time deployment:* software updates are triggered while the system is in use, i.e., without interrupting or stopping the execution while the update is in progress. In order to guarantee system safety, for a predefined period of time, this run-time deployment service is executed and verified in parallel to the normal operation of the system (i.e., in a *quarantine mode*). Once the quarantine period ends, this service will include a glitch switch between the old and the new function. For the realization of this run-time update deployment well-known techniques for software live-patching (e.g., the Linux kernel live patching [14]) will be applied.
- **Monitoring service.** This middleware service is the responsible of monitoring the correct deployment of updated software modules. This monitoring will check the ports and interfaces of each module (to other neighbouring software modules and to the hardware platform). The update monitoring will be realized in two flavours:
    1) *Dynamic online checks with rollback:* The updated function is stimulated with test data to check predefined corner cases on the target platform, similar to built-in-self-tests. This check will be performed for the online deployment strategy.
    2) *Parallel and monitored operation in quarantine mode*: The “old” and the updated software modules are executed in parallel for a predefined amount of time. The updated software module is executed in so-called quarantine mode until it has been approved to be working correctly according to a predefined confidence level. This check will be performed for the run-time deployment strategy.

    For an efficient collection and traceability with the system requirements (which have been expressed through contracts), this middleware service will acquire information from horizontal and vertical observers: the horizontal contract observers check for possible violations of logical integration conditions of a function with its environment. The vertical contract observers will check if the underlying operating system and middleware provides the assumed computation, communication and memory resources.

E. Secure communication

This module will define and implement the security communication mechanisms and library at the base of the software updates. It will allow the secure transmission of both the update (binaries, configuration files and UP2DATE contracts) from the server side to the target platform and the continuous feedback information collected by the monitoring module from the target to the server. In the process of sending the new software and the contracts from the server to the target, at least the following security requirements shall be considered:

- Authentication: ensure that the information is sent from a trusted server.
- Confidentiality: guarantee that the information sent is kept confidential by using encryption mechanisms.
- Integrity: detect any corruption in the data in the transmission process by for instance hashing algorithms.

The project will try to implement these mechanisms with the support of hardware security accelerators such as cryptographic modules and key management features.

F. Monitoring and controllability

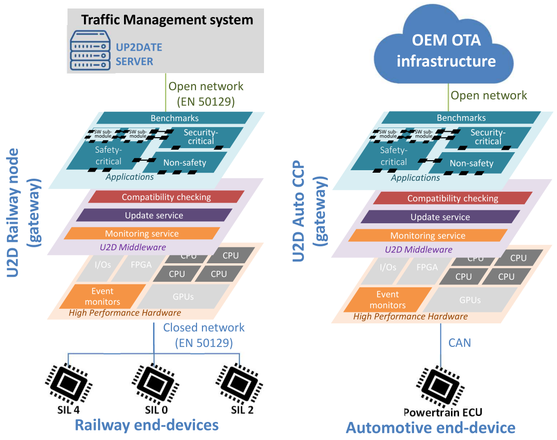
This module gathers the information required to execute the monitoring service at the middleware and configures the hardware platform to host new updates. It works on three separate lines:

- **Observability.** It implements a model-based monitoring system that relates safety and security properties required by contracts to low level event monitors. Those monitors have to be operated to create the proper abstractions such as ‘resource partition’. For instance, contracts will work with the concept of ‘resource capacity’ so that tasks can be given a capacity of each resource. In providing this abstraction, it is required to define it for each resource like buses, GPUs or caches. Likewise, the observability module will integrate the monitoring of the required events to create and detect variations in security fingerprints.
- **Controllability.** The module permits to configure the hardware by applying predefined setups. The different

setups will be used during the validation of a new software integration. For each update a new hardware set-up might be needed to accommodate new functionalities.
- **Feedback.** The module supports the analysis of the applications' resource usage during operation. The collected information is sent back to the server and used to determine whether in the new software update the application can be given fewer (optimized) resources without affecting its SASE properties.

V. Use-case applications

The project outcomes will be evaluated in two industrial use-cases for intelligent, autonomous and high-performance applications from the railway and automotive domains. For the different reasons stated in the subsections below, both use-cases are based on a centralized mixed-criticality node that apart form executing critical control tasks, acts as a gateway that manages OTASUs in several end-devices (see Figure 6).



(a) Railway use-case (b) Automotive use-case

Fig. 6. High level UP2DATE use-case architectures.

A. Railway use-case

The railway use-case is based on the railway signalling product range. It will integrate the UP2DATE architecture in a gateway device that centralizes the simultaneous downloading of updates on all the railway devices in a facility. The UP2DATE solution can bring great savings in the installation phase and through the product service life in the maintenance activities or when evaluation new test versions:

- In the *installation phase*, it is frequently necessary to adjust the configuration parameters and some software functionalities. Typically, these changes affect more than one device of the system, with different criticality levels.
- In the *maintenance phase*, uploads are less frequent, but the impact is greater. In this phase, security updates to keep the security level through the long product service life are especially important. Currently devices are updated in situ by specialized technicians, generating extra travel costs. In addition, when updating safety related devices, the current approach require the physical presence of a member from the railway administration infrastructure manager. The update procedure includes upgrading devices, performing conformance tests and managing the documentation.
- In a scenario where the system is *in service*, uploading a test version involves coordinating several teams performing simultaneous downloads on all the system. The tests are carried out at night, when the train service is suspended. After completing the tests, for re-establishing normal operation the previous approved versions have to be restored again. It is estimated that the cost of loading one version is equivalent to 15 specialized technicians working during the night.

The architecture proposed for the railway use-case (Figure 6(a)) is based on the introduction of an UP2DATE (U2D) railway node with three main purposes: (i) the simultaneous and coherent update of the multiple railway end-devices, (ii) to allow the secure transmission of data from an open network (server - U2D node) to a proprietary network (U2D node - end-devices) in compliance to the EN 50159 standard [8], and (iii) to integrate multiple applications with different criticalities and reduce in this way the number of control units distributed across the line. The UP2DATE server could be located together with the current Traffic Management System. Here the software and the configuration files for all the railway signalling equipment are generated from a common source for a given version and this coherency shall be maintained throughout the complete update process.

B. Automotive use-case

The automotive use-case will demonstrate the effectiveness of the UP2DATE approach on updating the software controlling a power train unit satisfying SASE requirements.

The new functionalities proposed in the context of the project are very interesting topics in the automotive market for the great benefits that they can bring to the management of software product evolutions during the post production and service phases. In the post production and service phase it is frequently necessary to adjust the configuration parameters and some software functionalities: currently devices are updated in situ (customer production or service sites) by specialized technicians, generating significant extra costs. In addition, a service/recall campaign involves public and users' communications that cause a brand image damage. The application of UP2DATE will allow to perform these activities in a lighter and more flexible way producing great savings through the product life-cycle maintenance activities.

The automotive industrial demonstration will take place in the context of the next generation of connected automotive in-car computing architectures. Today's cars contain more than a hundred computerized Electronic Control Unitss (ECUs). This number is increasing with every new car generation since a new function requires a new ECU, e.g., engine control, infotainment, driving assistance. An architecture containing so many, low-performance, separate control units is not able to address the future requirements on high-performance, yet safe and secure, computing for autonomous and connected driving.

As a solution, the automotive use-case presented in Figure 6(b) will integrate the UP2DATE (U2D) automotive Centralized Computing Platform (CCP) to integrated several ECUs in a single platform with high-computational power connected to a reliable high-speed network. In addition, this automotive CCP will act as a gateway that communicates with the specific back-end infrastructure (private cloud) and distributes the information through the in-vehicle network to the corresponding end-device.

VI. Conclusions

This paper provided an overview of the recently started UP2DATE European project which has the main goal of implementing a new architecture to allow the execution of over-the-air updates of mixed-criticality software on complex platforms without jeopardizing system safety and security. Given the early stage of the project, the paper does not include project results yet. Instead, we explained the main current limitations for achieving the goal of adopting OTASU in the critical domain and we defined the objectives and key technological innovations that will guide the project towards them. In summary, the main challenges in which the project will focus are related to safety, security, availability and the increasing platform complexity. UP2DATE proposes a new software architecture based on a novel adoption of the design-by-contracts approach to foster modularity and composability of updates together with runtime monitoring strategies. This architecture allows obtaining safety and security guarantees before, during and after an update. In addition, the proposed architecture follows an interactive development process where the software design is optimized with runtime data in each software update.

UP2DATE will bring the OTASU technology closer to safety-critical industries 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 by a certification authority to pave the way towards the future certification of the UP2DATE architecture.

Acknowledgments

The research presented throughout this paper has received funding from the European Community's Horizon 2020 programme under the UP2DATE project (grant agreement 871465) and the European Research Council (ERC) under the European Unions Horizon 2020 Research and Innovation programme (grant agreement No. 772773).

References

[1] I. Agirre, J. Abella, M. Azkarate-Askasua, and F. J. Cazorla. On the tailoring of CAST-32A certification guidance to real COTS multicore architectures. In *2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–8, June 2017.

[2] Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Tom Henzinger, and Kim Guldstrand Larsen. Contracts for Systems Design: Methodology and Application cases. Research Report RR-8760, INRIA, July 2015.

[3] Ankita Bhutani and Shubhangi Yadav. Global Market Insights. Automotive Electronics Market, report ID: GMI183. https://www.gminsights.com/industry-analysis/automotive-electronics-market, 2018.

[4] Francisco J. Cazorla, Jaume Abella, Javier Jalle, Mikel Fernandez, Leonidas Kosmidis, Luca Fossati, and Marco Zulianello. PMCs for real-time multicore systems: analysis of the state of the art and initial proposal. *European space Agency software system division and data division final presentation days*, 2014.

[5] EN50126. EN50126:2012 - railway applications: The specification and demonstration of dependability, reliability, availability, maintainability and safety (RAMS), 2012.

[6] EN50128. EN50128:2011 - railway applications: Software for railway control and protection systems, 2011.

[7] EN50129. EN50129:2005 - railway applications. communication, signalling and processing systems. safety related electronic systems for signalling, 2003.

[8] EN50159. EN50159:2011 - railway applications. communication, signalling and processing systems. safety-related communication in transmission systems, 2011.

[9] Kersten Heineke, Alexandre Ménard, Freddie Sdergren, and Martin 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.

[10] Michael Httermann. *DevOps for Developers*. Apress, Berkeley, CA, 1 edition, 2012.

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

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

[13] International Organization for Standardization. ISO/DIS 26262 Road Vehicles – Functional Safety, 2009.

[14] Paul Jacobs. History of Linux Kernel Live Patching. https://www.howtoforge.com/history-of-linux-kernel-live-patching/, 2019.

[15] S. Liu, J. Mu, D. Zhou, and 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)*, pages 779–783, Aug 2017.

[16] E. Mezzetti, L. Kosmidis, J. Abella, and F. J. Cazorla. High-Integrity Performance Monitoring Units in Automotive Chips for Reliable Timing V&V. *IEEE Micro*, 38(1):56–65, January 2018.

[17] I. Mugarza, J. Parra, and 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)*, pages 1–8, June 2018.

[18] Imanol Mugarza, Jorge Parra, and Eduardo Jacob. Analysis of existing dynamic software updating techniques for safe and secure industrial control systems. *International Journal of Safety and Security Engineering*, 8:121–131, 01 2018.

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

[20] John 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.

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

[22] M. Steger, C. A. Boano, T. Niedermayr, M. Karner, J. Hillebrand, K. Roemer, and W. Rom. An Efficient and Secure Automotive Wireless Software Update Framework. *IEEE Transactions on Industrial Informatics*, 14(5):2181–2193, May 2018.

[23] Reinhard Wilhelm. Mixed Feelings About Mixed Criticality (Invited Paper). In Florian Brandner, editor, *18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018)*, volume 63 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:9, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[24] XILINX. Zynq UltraScale+ MPSoC. https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html.