Formal Design and Validation of an Automatic Train Operation Control System

Arturo Amendola[1], Lorenzo Barruffo[1], Marco Bozzano[2], Alessandro Cimatti[2], Salvatore De Simone[1], Eugenio Fedeli[1], Artem Gabbasov[2], Domenico Ernesto Garrubba[1], Massimiliano Girardi[2], Diana Serra[1], Roberto Tiella[2], and Gianni Zampedri[2]

[1] Rete Ferroviaria Italiana, Osmannoro, Italy
[2] Fondazione Bruno Kessler, Trento, Italy

Abstract. In this paper, we report on the design of a complex control system, namely the Automatic Train Operation (ATO), which aims at enhancing the Grade of Automation in train operations (passenger transportation, infrastructure monitoring) in high-speed lines. The development of ATO is being conducted as an industrial project, with contributions from different research teams. The design of the system is complex in terms of architecture, functionality, safety and reliability requirements to be fulfilled, and geographical distribution of the development teams. Formal methods and model-based design are used to master the complexity of the design and of the system integration. Our approach is based on formal tools for system specification and validation, which support automatic code generation, early design validation, testing and simulation, and runtime verification. Moreover, we structured the development process in different phases and configurations, corresponding to increasing functionality of the system and different deployment configurations. The project is at an advanced stage of execution. In this paper, we demonstrate the effectiveness of the proposed approach and methodology, we discuss our experience and the lessons learned.

1 Introduction

The steady progress of the Information and Communication Technology and the limited efficiency of manual drive, which is mainly based on training and human experience, lead to the need for an automated management and control for railway traffic, which can best perform and react to different operating conditions or sudden changes. According to the International Association of Public Transport, there are five Grade of Automation (GoA) [8] that go from 0, which means absence of automation, up to 4, which indicates a fully automated train control and management without any staff on board.

In this context, Automatic Train Operation (ATO) systems aim to transferring the responsibility of train management from the driver to an automated control system, optimizing the driving performances due to the characteristics and conditions of the track, the energy consumption and the passenger comfort and

quickly reacting to unsafe situations. An additional protection level is guaranteed by the constant supervision of a vital computer (EVC) which interfaces with the infrastructure monitoring system following the ERTMS/ETCS standard for high-speed railway lines [5]. ETCS, with its direct connection to the braking system, protects the vehicle from some critical situations such as violating speed limits or running through places where it is not allowed.

In this paper, we report on the development of an ATO control system, which is carried out as an industrial project, with contributions from different research teams throughout the Italian territory (RFI, FBK and the Universities of Naples, Salerno and Bari, for a total of 4-6 persons per team). The design of the system is complex in terms of architecture, functionality, safety and reliability requirements to be fulfilled. In order to master the complexity of the design and of the system integration, we based our approach on the use of formal methods and model based-design for system specification and validation, which support automatic code generation, early design validation, testing, simulation and runtime verification.

ATO is currently at the stage of a prototype, but it will eventually evolve into a product. The objective is to have a GoA4 ATO operating on a prototype light-vehicle, equipped with devices for the infrastructure monitoring, running on an ERTMS/ETCS Italian high-speed line. This eventually could be the first step on the way to meet the challenge of adapting the design and control techniques from this prototype domain to applications on the high-speed mainline railway. The ATO project is at an advanced stage of execution. In this paper, we demonstrate the effectiveness of our design approach and methodology and discuss our experience and lessons learned.

The rest of the paper is structured as follows. In Section 2 we describe the ATO system, its architecture and requirements. In Section 3 we discuss the design challenges. In Section 4 we present our formal approach to the development and verification of ATO. In Section 5 we discuss the lessons learned. Finally, in Section 6 we draw some conclusions and outline directions for future work.

Related Work. For metropolitan railway lines, several approaches have been proposed to optimize train operation and energy consumption with autonomous driving [9], by combining high-level and low-level control of the ATO [10] or dealing also with train load and delays of the line [6]. However, the challenge of autonomous driving is still open for high-speed lines. Detailed modeling of high-speed trains is one of the most demanding research issues together with the development of powerful simulation platforms. Formal methods have been extensively applied in various industrial domains, including transportation [11], see e.g. [7] for a recent survey on the application to railway systems.

2 The ATO Control System

ATO consists of two cooperating systems: ATO Track Side (TS) and ATO On Board (OB). ATO-TS collects and forwards data on trains, tracks and timetables

concerning the train journey, while ATO-OB receives such data and uses them to control and drive the train. ATO can be operated by a remote driver, who is responsible for activating autonomous driving. The architecture of ATO is described in Fig. 1. In the rest of this section, we focus on ATO-OB. The Interface Manager allows ATO to interface with the different modules such as ATO-TS, ETCS, SMO (Speed Monitoring and Odometry), SCS (Supervision and Control System) and TIU (Train Interface Unit). The Controller implements the main finite state machine for the different ATO functional operating modes. Track Database Manager uses odometry data to localize the train on the line and validates the journey received from the trackside before the start of the mission. Autonomous Driving Functions receives the track and journey profile data from the Track Database Manager and uses them to generate an optimal speed profile, and the brake and traction commands to forward to TIU. The Energy Manager uses the battery and fuel data and the traction system status to monitor the energy level and evaluate the consumption needed to achieve the current mission.

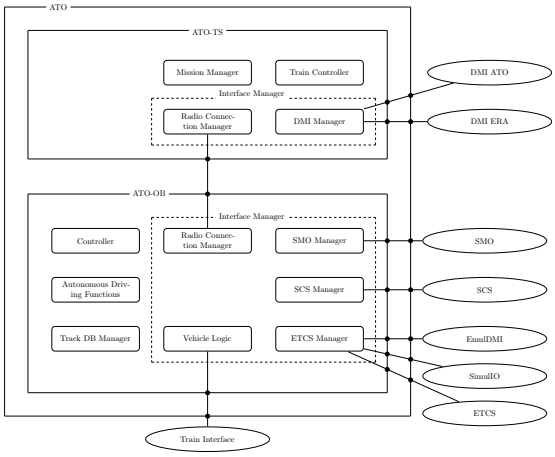

Fig. 1. ATO architecture

As an example, we consider the nominal scenario with a train stopped at a charging point. When the train is selected by an ATO-TS remote driver, ATO-OB verifies that its database version matches with the trackside one and performs some internal system tests. The remote driver plans the journey profile to be sent on board, and waits for an acknowledgment. When ETCS mode evolves to full supervision and other engagement conditions are fulfilled (e.g., ETCS and ATO are not applying full service or emergency brake, ATO-OB is localized on a specific Segment Profile sent by trackside, and train direction is forward), the remote driver can engage ATO-OB enabling autonomous driving. In such drive mode, the train reaches the final destination of the journey, respecting the related timetable of the assigned timing points and stopping points. An example of non-nominal scenario consists in using an autonomous vehicle equipped with ATO to rescue another vehicle that is blocking a high-speed line, due to a breakdown.

The design of ATO is subject to complex requirements. In order to meet specific project goals, we had to review and customize the functional and interface requirements defined by standard UNISIG subset, and add or discard some of

them. For example, we provided train localizing function data on board, moving it from trackside; we discarded all the requirements related to doors management since the prototype light-vehicle has none; we added camera requirements to mitigate the absence of driver on board and we customized the interface protocol in order to manage the data for the new functions we added.

3 Challenges

The design of ATO is very challenging, due to the complexity of the system and of the associated requirements. The ATO system is distributed. It consists of on-board controller (ATO-OB) and a trackside counterpart (ATO-TS) which, in turn, are composed of several modules realizing different functions, and connecting to external systems, such as ETCS. ATO is composed of heterogeneous components, specifically it includes components that interact directly with the underlying HW, e.g., those commanding braking or traction of the train. Such components rely on models of the HW which are inherently continuous (e.g., specified using differential equations). For this reason, ATO relies on heterogeneous design tools, based on different specification languages (e.g., Scade, Simulink, C).

The specification of ATO relies on a complex and evolving set of (functional, safety and performance) requirements, therefore the design process needs to be robust against changes and adaptations, and support system evolution. Moreover, the architecture of ATO, its control logic and modules must be designed to match the Safety Integrity Levels (SIL) requirements, according to the EN50128 standard. A Preliminary Hazard Analysis is in progress to assign SIL to the product. It is expected that different ATO modules will be assigned different SIL, with the highest levels assigned to the most critical components.

Finally, the design process of ATO must take into account the distribution of the development teams. This makes system integration a particularly challenging task, which calls for suitable verification and validation and testing strategies.

4 Formal Design of ATO

The software specification and design of ATO is based on formal methods. The V-Model, as specified in the CENELEC EN-50128 [2] standard guides the software development process from the definition of the system requirements to testing, integration and validation phases. Model-based design takes advantage of co-design strategies and interdisciplinary effort, favoring cooperation between teams with skills in different disciplinary sectors.

Given the high assurance requirements of ATO, we based the design on tools such as ANSYS SCADE Suite[3] and Architect[4], which offer qualifiable/certified code generation capabilities and interoperability with other development tools and platforms. The use of SCADE meets the production standards for SIL

[3] https://www.ansys.com/products/embedded-software/ansys-scade-suite
[4] https://www.ansys.com/products/embedded-software/ansys-scade-architect

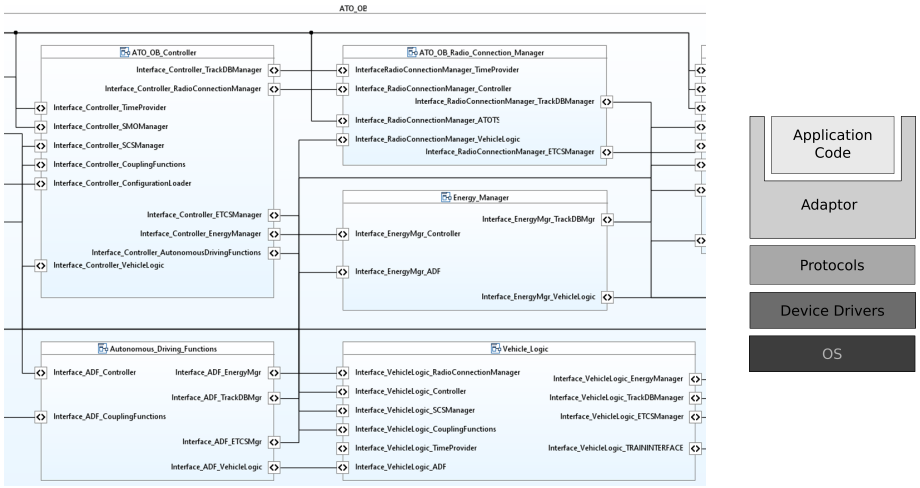

Fig. 2. An excerpt of an IBD for ATO-OB (left) and ATO process layers (right)

3/4 SW, since it is compliant with the required metrics and constraints. Our approach integrates the capabilities offered by SCADE with other functionality for verification and validation implemented in our proprietary tool chain based on the nuXmv model checker [1]. In this section, we discuss how these solutions address the challenges outlined in Section 3.

4.1 Requirements and Architecture

The development of ATO is guided by an evolving set of requirements, and by a set of operational scenarios specifying some real case missions the system must fulfill. We grouped requirements by functionality and we identified the corresponding modules responsible for taking into account the set of assigned features. We further split requirements into those allocated to ATO-OB and those allocated to ATO-TS. Requirements have been analyzed to extract a hierarchical representation of ATO and the corresponding logical architecture. The architecture has been modeled using SCADE Architect, resulting in several Block Decision Diagrams (BDD) and Internal Block Diagrams (IBD). As an example, in Fig. 2 (left) we present (part of) an IBD focusing on a subset of ATO-OB. Since the design is distributed, it is of utmost importance to design a robust and stable architecture, in which interfaces between different modules are well defined and shared with all the involved teams. In this respect, scenarios have been formalized into sequence diagrams, which strictly refer to the architectural decomposition, and have then been used to guide the implementation of the components and to derive test suites to perform unit and integration testing.

The layered architecture of an ATO process is depicted in Figure 2 (right): (a) *Application Code Layer* is a pure C function which computes abstract[5]

[5] We use the term *abstract* for protocol-independent data

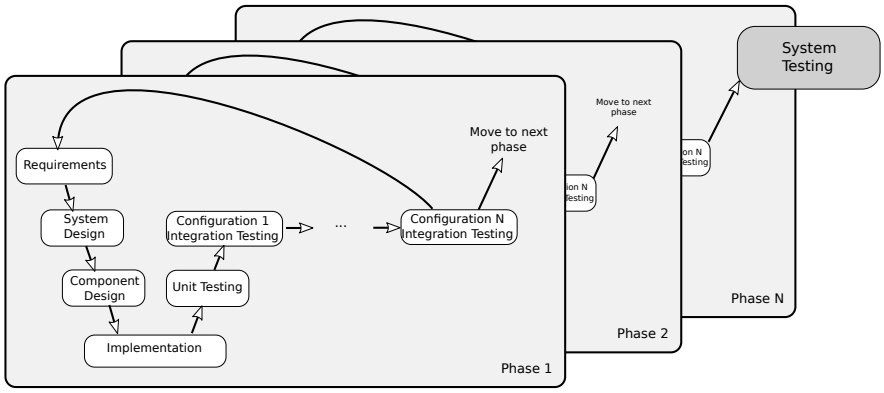

Fig. 3. Phased V-Model

outputs from abstract inputs and current internal state. The layer is run at each execution cycle, (b) *Adaptor Layer* is in charge to periodically invoke the Application Code and to route abstract data from/to the layer below, (c) *Protocols Layer* handles incoming/outgoing data frames performing encoding/decoding operations mapping protocol data from/to abstact data, (d) *Device Drivers Layer* is in charge to cope with connected devices hiding communication details to the upper layers, finally (e) the underlying *OS Layer* provides required services such as scheduling and access to disks, network and other peripherals.

4.2 Development Process

Development life-cycle. The system is developed following a process that can be described as a *phased* V-Model (Figure 3). To face the challenges implied by the novelty of the project, such as instability of requirements and variability of components interfaces, the process extends the classical V-Model with aspects borrowed from the Agile philosophy.

In details, the classical V-model is extended along two dimensions: *phases* and *configurations*. Phases concern functionalities and target a subset of system features. In a phase, the V-model is iterated refining requirement analysis, system architecture, implementation, unit and integration test until stability is reached. Once the last phase is terminated, the system will undergo a final system testing activity. Three phases were identified: remote operations, autonomous driving, and the full ATO system. Orthogonally, a *configuration* specifies which layers of the system are involved in the integration testing activities and, consequently, which running environment tests are run on. We identify three main configurations: (a) *Configuration 1* involves the Application Code Layer only. The code is tested in the simulation/testing environment of SCADE Suite (see next section for details), (b) *Configuration 2* extends Configuration 1 with the Adaptor and Protocols layers. The code is run on host, introducing asynchronous execution of subsystems and the interaction with services, e.g. logging service. Communication

with devices is simulated, (c) *Configuration 3* adds the Device Layer and the Target OS Layer. Each subsystem is run on the proper target using real devices. Configuration 3 actually is subdivided in a set of subconfigurations (3a, 3b, etc.) that more and more integrate larger parts of the final physical system.

System architecture, component design and implementation. The development process follows the model-based approach, where a machine-readable formal representation of a system is built as the main project's artifact. Such a representation (model) is the input for all the downstream development activities, most notably allowing for formal verification of the properties of the system, i.e. model checking [4], certified source code and documentation generation. In details, the system architecture is specified in SCADE Architect using the SysML language. A SysML architectural model typically comprises hierarchical decomposition of the system, connections, interfaces and data types. SCADE Architect provides validation tools for early identification of flaws (see Section 4.3). Using the model generator provided by SCADE Architect, for each subcomponent we generate a skeletal behavioral model written in SCADE Suite language. The majority of subsystems/components are implemented using the SCADE Suite language. One subsystem is implemented using MathWorks Symulink and one data-intensive component is manually written in C. In total, the SW for ATO-OB contains about 75K lines of code. SCADE Suite comprises a code generator for translating models into C code which is certified under EN 50128:2011 at T3/SIL 3/4. The ratio of the C code we automatically generated for ATO-OB using Scade is about 75% of the whole code (including manually written code, and code generated by other means). Certification implies that unit/component testing activities can be performed on models instead of on generated code, reducing certification times and costs (the reduction is estimated in the order of 50%).

Moreover, to develop the Protocol Layer, we used ASN.1 as the interface description language, due to its flexibility, widespread use, and extensible format. In particular, we used ASN.1 to generate an intermediate formal specification to support component interaction. This approach can accommodate different communication protocols, including ad-hoc protocols described in textual or tabular form. Given that the protocols are constantly evolving and that the manual implementation of ASN specifications is time-consuming and error-prone, we generate them from tabular description for one protocol and from SCADE components based on textual description for others.

System Integration. When managing distributed teams and heterogeneous components, system integration becomes a highly important task. ATO contains some modules implemented using the SCADE language natively, while others are designed in different formats, e.g. Simulink and C. Source code generated from such models is linked to the rest of the code by means of a SCADE Suite language feature called 'external operators', i.e., operators whose interface is mapped to the interface of the corresponding external module via some glue code. We require a test suite associated to each module which must invoke all its nominal behaviors. Each test is then replicated in the SCADE framework so

that we can mimic the same actions and prove that we obtain the same results even with the integrated system. Operational scenarios are then used to derive some integration test cases that are intended to simulate interactions between components and to prove that ATO behaves as expected. In order to avoid non regression failures, we followed a continuous-integration approach by designing a custom framework based on the python package 'pytest'. Moreover, we rely on the 'git' versioning tool to share the SW development effort among different teams and to freeze system implementation at specific milestones.

4.3 Verification and Validation

We used multiple and complementary ways to formally verify and validate the design and implementation of ATO. First, we used SCADE to perform early model validation. SCADE offers some checkers that can be used to validate the hierarchical composition of the architecture, in particular the compatibility of the component interfaces, and to check that sequence diagrams refer to valid data. This allows us to guarantee that the formalization of the scenarios is compliant with the architecture, before moving to the implementation. Then, we used SCADE Suite to design, simulate and test the system by means of scenario validation, i.e. by specifying values on input ports and checking that the outputs are as expected. We performed this task starting from component level up to system level. In case of unit testing we also made use of the model coverage feature, which allows to highlight which (if any) paths of the model are not stimulated by tests; in this case, we enriched the test suite in order to reach the highest possible coverage. When scenarios need to be modified or we want to verify other sets of requirements (for instance when moving to a subsequent development phase), the same process is replicated, starting from the architecture up to system integration testing.

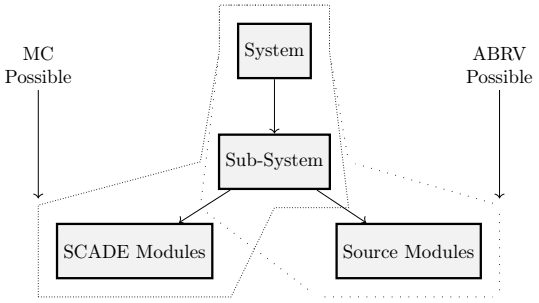

Fig. 4. Our approach to property-based formal verification

The ATO outcomes can be visualized and verified after executing a scenario by means of logging. A separate component (called ATOLOG) records all the relevant diagnostic information. It consists of: a server that receives and saves log packets from the log clients (written in C and Python) of other ATO components; a log client that provides an API for sending log packets; and various tools for decoding, analyzing, and visualizing raw binary data collected by the server. The collected diagnostic information is represented by the messages exchanged among

components, and it can be processed with various tools to verify that the scenario execution results correspond to the expected behavior.

Finally, once system integration is consolidated and all the individual sub-modules have been validated, we used model checking (MC) techniques to perform property-based (runtime) verification. Namely, we used a custom tool-chain based on the nuXmv model checker [1] and NuRV, an extension of NuXmv for runtime verification [3], and we implemented a (in-house) translation from SCADE to nuXmv. Our approach is described in Fig. 4. The system to be verified is split into one part which is formally modeled in SCADE, hence amenable to formal verification, and one which is not. In the first case, we used model checking to automatically verify system-level properties. In the second case, we used techniques based on Assumption Based Runtime Verification (ABRV) to automatically generate monitors that can be used as test oracles, see [3]. Interestingly, this process is completely automatic and requires just a small effort to connect the generated monitors to the rest of the system, before conducting the tests. In this way, many verifiers can be generated from the properties, while the effort required for refactoring, when the module interface evolves, is negligible.

5 Lessons Learned

The design of ATO has raised many challenges, due to its inherent complexity, and the distribution of the development effort. The main problem we had to address was how to effectively split the work among different teams. Continuous integration, supported by custom strategies for testing and by versioning tools, was the natural choice to address the complexity of system integration, along with ad-hoc strategies to deal with system evolution, e.g. to deal with updates to the interfaces of the subsystems allocated to external development teams. In this respect, we were forced to agree with partners not only about the definition of the high-level interfaces, but also about the precise semantics of individual fields.

The phased V-model allowed us to progressively design and implement the functionality of the system in two different respects. First, it enabled us to streamline the support for different operational scenarios, concentrating on one scenario at a time. Second, it enabled us to test the implementation of the integrated system on different deployment configurations (using a simulator; on one or more hosts; on the final OS with the target HW in the loop), making it possible to progressively release the deployed system on different targets, as soon as the latter become available in the course of the project.

Finally, we have carried out verification and validation using a mix of strategies and tools, integrating the support given by tools such as Ansys Scade Suite and Architect, simulators, and our proprietary tool chain for formal verification, based on model checking. Particularly effective was our choice to use both *design-time* model verification, and custom techniques for *runtime* monitoring, in combination with testing. The latter enabled us to cover – via testing – the verification of system-level properties that were out of reach for model checking, due to the complexity of the models and the state explosion problem.

Based on our experience, the formal approach proved to be effective and gave numerous benefits. Indeed, most of the flaws we encountered during system integration were located in components that had been outsourced, and were designed and tested using traditional methodologies, without using formal methods.

6 Conclusions and Future Work

In this paper we discussed the design of a complex control system, the Automatic Train Operation, and we presented a formal methods approach, which guides the ATO development throughout all the development phases.

Currently, ATO is at the stage of a prototype. We estimate to execute first chassis dynamometer tests by March 2022 and then field tests by June 2022 on the Bologna San Donato railway test circuit, the first fully equipped laboratory in the field throughout Europe. ATO is designed on a single-unit unmanned prototype light-vehicle which does not require the presence of on board driver, cabin staff or passengers, with all the implications that such specific design brings. So far, possible future developments concern the design of an ATO which is able to control and drive a multiple-unit high-speed train, with passengers on board.

References

1. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv Symbolic Model Checker. In: CAV. pp. 334–342 (2014)
2. CENELEC: EN 50128, Railway applications – Communications, signaling and processing systems – Software for railway control and protection systems (2011)
3. Cimatti, A., Tian, C., Tonetta, S.: Nurv: A nuxmv extension for runtime verification. In: Proc. Runtime Verification. LNCS, vol. 11757, pp. 382–392. Springer (2019)
4. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (2000)
5. European Union Agency for railways: ERTMS – Making the railway system work better for society (2016)
6. Fernández-Rodríguez, A., Fernández-Cardador, A., Cucala, A., Domínguez, M., Gonsalves, T.: Design of Robust and Energy-Efficient ATO Speed Profiles of Metropolitan Lines Considering Train Load Variations and Delays. IEEE Transactions on Intelligent Transportation Systems 16(4), 2061–2071 (2015)
7. Ferrari, A., ter Beek, M.H.: Formal methods in railways: a systematic mapping study (2021)
8. International Association of Public Transport: A global bid for automation: UITP Observatory of Automated Metros confirms sustained growth rates for the coming years, Belgium
9. Licheng, T., Tao, T., Jing, X., Shuai, S., Tong., L.: Optimization of train speed curve based on ATO tracking control strategy. In: Chinese Automation Congress (2017)
10. Su, S., Tang, T., Chen, L., Liu, B.: Energy-efficient train control in urban rail transit systems. Journal of Rail and Rapid Transit 229(4), 446–454 (2015)
11. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.S.: Formal methods: Practice and experience. ACM Comput. Surv. 41, 19:1–19:36 (2009)