MDE based IoT Service to enhance the safety of controllers at runtime

Miren Illarramendi Rezabal[1][0000−0003−3770−1495], Leire Etxeberria[1], Xabier Elkorobarrutia[1], Jose Maria Perez[1], Felix Larrinaga[1], and Goiuria Sagardui[1]

Mondragon Goi Eskola Politeknikoa, Loramendi 4, 20500 Mondragon, Spain
millarramendi@mondragon.edu

Abstract. One of the challenges for complex IoT software systems is to increase their safety. A Model Driven Development approach helps in the design and development phase of these systems while runtime checking techniques help to enhance safety. To supervise the status of different IoT services that are registered in a local cloud at runtime, the solution that is presented in this work uses the information that it receives from the different services registered in a local cloud in model terms. The runtime checker, the new Safety related service of the Arrowhead framework, has predefined contracts to ensure the correctness of the services at runtime. Based on these contracts and checking the information that it receives at runtime it is able to detect unsafe scenarios. Once an unsafe scenario is detected, it starts a safe process to protect the behaviour of the whole system adapting the wrong service or services to a degraded operation mode at runtime. All these services will be Arrowhead compliant.

Keywords: Models@runtime · IoT Services · Runtime Verification · Runtime Adaptation · Runtime Monitoring.

1 Introduction

In our live, we are surrounded by Cyber Physical Systems (CPSs) and System of CPSs (SoCPS)s due to an increasing number of intelligent systems that involve safety, life and business-critical requirements in domains such as transportation, healthcare or systems for managing aspects of our homes. These systems directly interfere with our physical world which makes their safe, dependable and resilient operation one of their primary requirements.

In recent years, software components have gained importance as controller part of the CPSs. This has led to the control software taking more responsibility and needing mechanisms to enhance correct and safe behaviour. Furthermore, every component of a CPS is a potential point of failure.

Monitoring information related to the internal status of the CPSs at runtime can anticipate the occurrence of failures. This makes it possible to take corrective actions earlier and prevent faulty scenarios. This idea is described as a safety bag in [5]. The goal is to prevent software systems' hazardous states by means of safety verification at runtime. Thus, we increase their robustness ensuring safety.

Advances in computing and communication are leading to the digitization of industry. The use of IoT platforms allows the access devices and machines in

a transparent way making possible the digitalization of manufacturing systems. There are many initiatives around Digital Manufacturing Platforms and IoT that have been developed in different Research and Innovation actions in the European Community (CREMA [6], FIWARE [8], Arrowhead [7],...).

In the European ECSEL project Productive 4.0 [1], using IoT platforms to enhance the robustness of the software controllers has been identified as a novel research topic. Different approaches and works have been started in this area and the work that is presented in this position paper, the Arrowhead compliant Safety service, is one of them.

Section 2 presents background concepts of the work, and in Section 3 the overall Safety Service Architecture is shown. Section 4 presents an academic example of the solution and finally, Section 5 closes the paper with conclusions and future lines.

2 Background

The proposed work in this position paper is a mix of two different approaches. On the one hand, REflective State-Machines based observable software COmponents (RESCO) software components, which have introspection and reflection ability at runtime are considered. For this end, the work presented in [9] has been the starting point. On the other hand, we have selected an IoT platform to develop the Safety service itself: the Arrowhead IoT framework.

2.1 RESCO software components

The overview of the approach considered in this study to automatically generate software components with introspection and reflection ability at runtime is highlighted in Figure 1. It represents the Model-driven workflow that safety and software engineers must consider when using the methodology we propose.

First, the behaviour of the components is modelled using Unified Modeling Language - State Machines (UML-SMs) by the Papyrus tool [11].

This first step is performed by the designer and in order to have the adaptation ability at runtime, the designer has to design two or more annotated UML-SM models: the normal-mode UML-SM and one or more safe-mode (degraded-mode) UML-SMs. Thus, when an error is detected, the software component will adapt from normal-mode UML-SM to the safe-mode (degraded-mode) UML-SM.

In a second step, a RESCO metamodel conformant model (instrumented model) is generated automatically by ATL [3] language performing Model to Model (M2M) transformations. In this step, the original UML-SM model is enriched by information needed to decide which states have to be observed at runtime or not and as result we have the RESCO-SMs.

In the last step, a model conforming to the RESCO metamodel is transformed to code by means of a Model to Text (M2T) transformation by Acceleo [2].

Having these software components with introspection ability, we can design a solution that enables the developers to generate software systems able to check their software components in model terms at runtime.

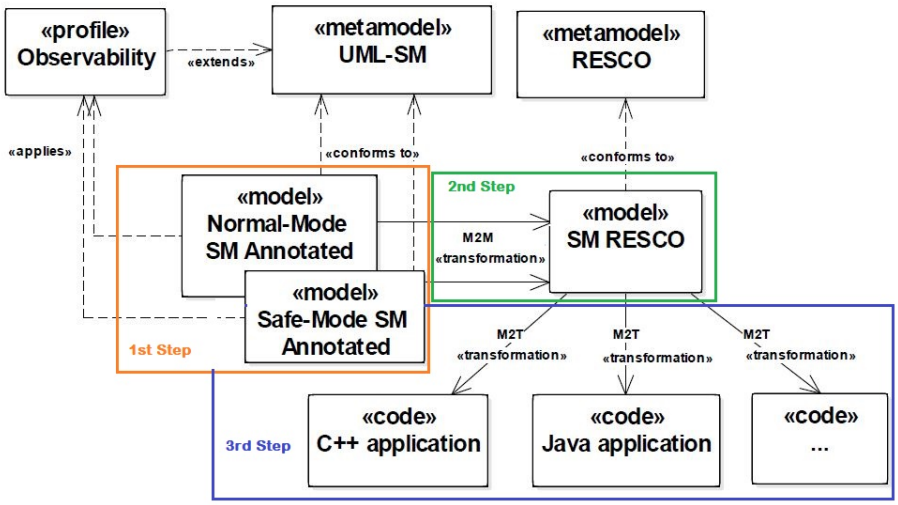

Fig. 1. Model-driven Workflow

2.2 Arrowhead framework

The Arrowhead project addresses efficiency and flexibility on a global scale through collaborative automation. Thus, Arrowhead's greatest challenges are to enable: 1) interoperability of services provided by almost any device. 2) integrity of the services provided by any device. In response to these challenges, the Arrowhead framework [7] was created.

The framework aims to normalize the interaction between IoT industrial applications through service-oriented architectures (SOA). Services are exposed and consumed by systems, which run on devices. Figure 2 shows the different types of services within the Arrowhead framework.

2.3 State of the Art and Opportunity of the Solution

In the scope of Models@run.time, different solutions to trace the UML-SMs in order to obtain information about the monitored software components at runtime have been analyzed. Most of the approaches focus on instrumenting the code and not the model (e.g., [10]).

There is a similar approach, [4], but the implementation and the model-to-model transformation rules are different. In their solution they add new objects to debug/trace the execution in all the transition chains. In our case, the same object is reused in all the transitions. This way, the number of objects in the model is independent of the size and number of transitions of the state machine. In addition, they use these traces to debug the model at runtime. Our aim is to use these traces to detect system inconsistencies and errors as soon as possible and once something wrong is detected to start an adaptation process at runtime.

Regarding the IoT platforms, as mentioned before, there are many initiatives around Digital Manufacturing Platforms and IoT that have been developed in

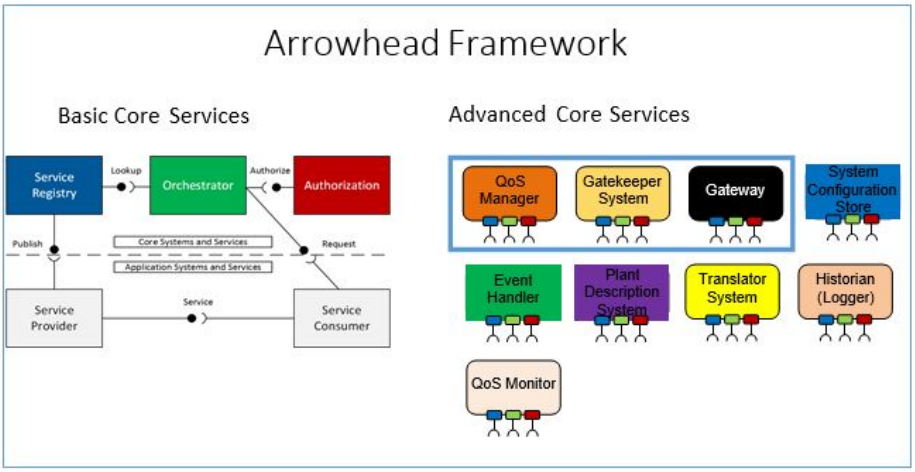

Fig. 2. Basic Services and Advance Services of the Arrowhead Framework. Source: Productive 4.0 project [1].

different Research and Innovation actions in the European Community. However, these platforms have not led to a successful and efficient digitisation of all aspects and resources of manufacturing industry. There are also commercial IoT platforms being used in industry such as Amazon's AWS IoT, Microsoft's Azure IoT Suite, IBM's Watson IoT or MathWorks' ThingSpeak. These solutions are closed solutions based on specific technologies making interoperability between different solutions difficult. In response to these challenges, the Arrowhead framework [8] was created.

This work, aims to fill the gap identified in the above defined two research lines. On the one hand, RESCO based software are able to provide software components information in model element terms at runtime and, on the other hand, if we are able to serve this information as services, we can use them as Arrowhead compliant services with the aim to detect hazardous scenarios in IoT based solutions (e.g. industrial plants). This is the main objective of this work: proof of concept of the Arrowhead Safety Manager based on RESCO software components.

3 Safety Service Architecture

The main objective of the Safety service is to supervise the status of different Arrowhead compliant services that are registered in a local cloud. Based on defined contracts, it will ensure the reliable and safe operation of these services.

Its role can be defined as unsafe scenarios detector. When it detects an unsafe scenario, it starts a safe process to protect the behaviour of the system.

The approach or concept developed has the following characteristics:

- Arrowhead Local cloud has a Safety related service.
- Different services are monitored by the Safety Monitor.

– These services are based on the RESCO software components. they have to be offered as Arrowhead compliant services and they provide information about the status of the controlled system in model terms at runtime.
– The monitored information is checked by the Safety Manager. For doing that, safety rules or properties common language is defined and the safety rules will be defined using this common language in each use case.
– When an unsafe scenario is detected, a safe process starts and the services involved in the use case are updated to a graceful mode.

4 Safety Service: Toy Example

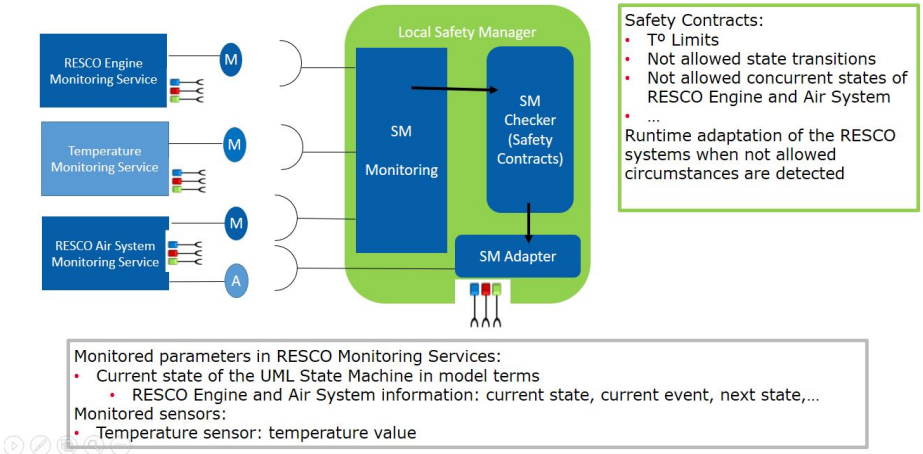

Fig. 3. Safety Manager Toy Example.

In this section we will present a proof of concept example developed in the Productive 4.0 [1] project. In that example, we consider a system with a RESCO Engine Monitoring Service, RESCO Air System Monitoring Service and a Temperature Monitoring Service.

Both RESCO based services are providing their internal status at runtime and in model terms (using the UML-SM formalism). Every time that the transition is going to be performed, they send the related information to the Safety service of the Arrowhead framework. In addition, the Temperature service provides the value of the temperature sensor that is installed in the engine.

In addition, different contracts that define the not allowed unsafe scenarios are defined. Thus, at runtime, when the Safety service receives new updates, it checks if these contracts are fulfilled or not. In case they are not, it starts a process to adapt the RESCO services to a safe state.

Figure 3 shows the example that has been developed to check the concept we are presenting in this work. We have done the first evaluation of the demonstrator presented in this work obtaining successful results. We inserted faulty

temperature values to force non-safe scenarios and the Safety Manager service detected the situations and started the adaptation process in the involved services sending the system to a safe scenario.

5 Conclusion

In this paper we have presented the first concept of a Safety Service that could be added to the Arrowhead framework. Its main aim is to check the correct and safe operation of the different controls by this IoT platform. For doing this, we suggest to develop RESCO based Arrowhead compliant software services. These software components have the ability to offer their internal status in models terms at runtime. Thus, we may reuse the models used in design and development phases also at runtime for correctness verification purposes.

In order to check that the concept is achievable, we have developed an simple toy example. We have mixed both approaches, RESCO and Arrowhead framework, and the result has been that it is possible to check the correctness of software controllers in model terms at runtime. In the future, we would like to expand the evaluation using more realistic industrial cases and environments.

Acknowledgment

The project has been developed by the Embedded System Group of MGEP and supported by the Department of Education, Universities and Research of the Basque Government under the projects Ikerketa Taldeak (Grupo de Sistemas Embebidos) and TEKINTZE (Elkartek 2018) and the European H2020 research and innovation programme, ECSEL Joint Undertaking, and National Funding Authorities from 19 involved countries under the project Productive 4.0 with grant agreement no. GAP-737459 - 999978918.

References

1. 4.0, P.: (2019), https://productive40.eu/
2. A: Acceleo. Tech. rep., https://www.eclipse.org/acceleo/ (2016)
3. ATL: Atl transformation language (2018), http://www.eclipse.org/atl/
4. Bagherzadeh, M., Hili, N., Dingel, J.: Model-level, platform-independent debugging in the context of the model-driven development of real-time systems. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. pp. 419–430. ACM (2017)
5. Brini, M., Crubill, P., Lussier, B., Schn, W.: Complementary methods for designing safety necessities for a safety-bag component in experimental autonomous vehicles. In: Proceedings 12th National Conference on Software and Hardware Architectures for Robots Control (2017)
6. CREMA: https://www.crema-project.eu/
7. Delsing, J.: Iot automation: Arrowhead framework. CRC Press (2017)
8. FIWARE: https://www.fiware.org/
9. Illarramendi, M., Etxeberria, L., Elkorobarrutia, X., Sagardui, G.: Runtime observable and adaptable uml state machines: Models@run.time approach. In: 34th ACM/SIGAPP Symposium On Applied Computing (SAC) (2019)
10. Mazak, A., Wimmer, M., Patsuk-Bösch, P.: Execution-based model profiling. In: International Symposium on Data-Driven Process Discovery and Analysis. pp. 37–52. Springer (2016)
11. Papyrus: Papyrus (2019), https://eclipse.org/papyrus/