SmartCLIDE[*]
Runtime Monitoring and Verification (RMV)

Rance DeLong

September 12, 2022

Abstract

SmartCLIDE offers services to accelerate the creation and deployment of Cloud solutions by providing the ability for non-programmers to construct applications and new services using smart automation. One of the backend services provided by SmartCLIDE is runtime monitoring and verification (RMV) which in conjunction with automated testing is applied to assure the quality of the created services. In this paper we describe the objectives of RMV, and provide an overview of the approach and the benefits.

SmartCLIDE Quality Assurance

SmartCLIDE constructs new services according to the user's specifications from pre-existing and bespoke components. Supplementing the construction of new services, SmartCLIDE's strategy for quality assurance (QA) of user-constructed services includes both development-time and *run-time quality assurance* for functional and non-functional properties. In addition to the expected functional behavior of a service, key characteristics such as security, safety, privacy, resilience and reliability are general categories of runtime quality attributes that may be required of the service. Runtime QA is applied along with design-time QA, development testing, verification and qualification testing to assure that the needed quality attributes have been achieved.

Assurance of the correctness of a service may be addressed largely by the manner of its construction, giving rise to the term *correct by construction.* To achieve it a rigorous methodology is required, typically supported by automation[1] and tools. By automating the construction process certain sources of potential flaws may be systematically eliminated. In SmartCLIDE automation extends to AI-powered assistance in the selection and composition of components. SmartCLIDE can already make some claims to correctness by construction because the automation is systematic. However, the details of the construction methods may not be rigorous enough to extend correctness by construction to every functional or non-functional claim that could be made for a SmartCLIDE-constructed service.

Whilst, runtime QA is also a concern for entirely human-fashioned software artifacts, it may be even more beneficial for software that is constructed without human involvement and scrutiny

[*]This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871177

[1]Here "automation" or "automated" are used for processes that are fully or partially automated.

of every design and implementation decision. By reducing the development effort through automation some of the detailed expert human scrutiny that the service development would otherwise receive will likely not occur. Subtle semantic anomalies and "corner cases" may go undetected when automation uses service specifications to construct service implementations from diversely sourced and specified components, and only surface when run-time execution behaviors are observed.

Complementary Quality Assurance Methods

Among the methods that that could have been utilized for quality assurance of SmartCLIDE-created services are: correctness by construction, formal verification (FV), automated testing, and runtime verification RV. All of these methods, except for formal verification, are employed in SmartCLIDE. FV provides construction-time assurance that increases confidence that run-time behavior will not include unwanted effects by exploring all possible executions *a priori*. The application of FV, even the "fully-automated" kind, is typically expensive: being labor-intensive and requiring specialized expertise. It must be re-performed whenever the model is changed. Furthermore, it typically only verifies the *model* (an abstraction) of the implementation as opposed to the actual executable implementation. Due to these considerations we do not further consider FV as a viable routine activity in SmartCLIDE.

Conventional testing is one of the standard methods of discovering and correcting the sources of errant behaviours, and this method is also applied in SmartCLIDE by doing automated testing for SmartCLIDE-created services in addition to the unit and integration tests of the SmartCLIDE components themselves. Testing of SmartCLIDE-created services has the benefit that the actual service implementation is exercised in the tests rather than a model of the implementation as would be the case in FV. Testing involves identification of a finite number of test scenarios and test cases. As always, the issue with finite testing of a reasonably complex system, which has a potentially, and practically infinite number of distinct behaviours, is one of *confidence in the adequacy of the testing*, in particular that of chosen test cases and the test data. When test cases and test data are chosen automatically an additional source of automation-induced error or incompleteness is a source of adequacy concern.

Another potential source of runtime misbehavior has nothing to do with the construction or the functionality of a service but with the *assumptions* that underlie, possibly implicitly, the implementation of a component or a service. The implementation is only valid as long as these assumptions are satisfied and maintained. When assumptions are violated, either through incorrect composition of components, or through dynamically changing conditions at runtime, the implementation is likely to misbehave or completely fail.

Runtime monitoring and verification (RMV) is an aspect of the SmartCLIDE QA strategy that is used primarily at run time but also may be beneficial in the latter stages of development. RMV is able to check the monitored service at every step to confirm that it's behavior in the *current run* is consistent with its specifications and the, necessarily limited, results of prior finite testing. One of the main strengths of runtime verification is that it has the potential to detect a deviation from the required behaviour due either to an incorrect implementation of the specified behavior or to the, possibly dynamic, invalidity of an assumption.

Assurance of the runtime behaviour is addressed by validating that the service actually exhibits behavior consistent with the user's specification and with development-time test results, and that the assumptions made about the runtime environment, which were made at design time and thus built into the construction of the service, continue to be valid as the service executes.

Runtime Monitoring & Verification

Figure 1 shows an overview of the components of the SmartCLIDE RMV subsystem. The RMV subsystem interacts with other SmartCLIDE components through Message Oriented Middleware (MoM) or direct IPC, and uses an external tool, NuRV [CTT19b], to generate property monitor state machines. Property monitors are synthesized from a formal model of the nominal behavior of the created service and a specification of required properties using the method of assumption-based runtime verification (ABRV) [CTT19a].

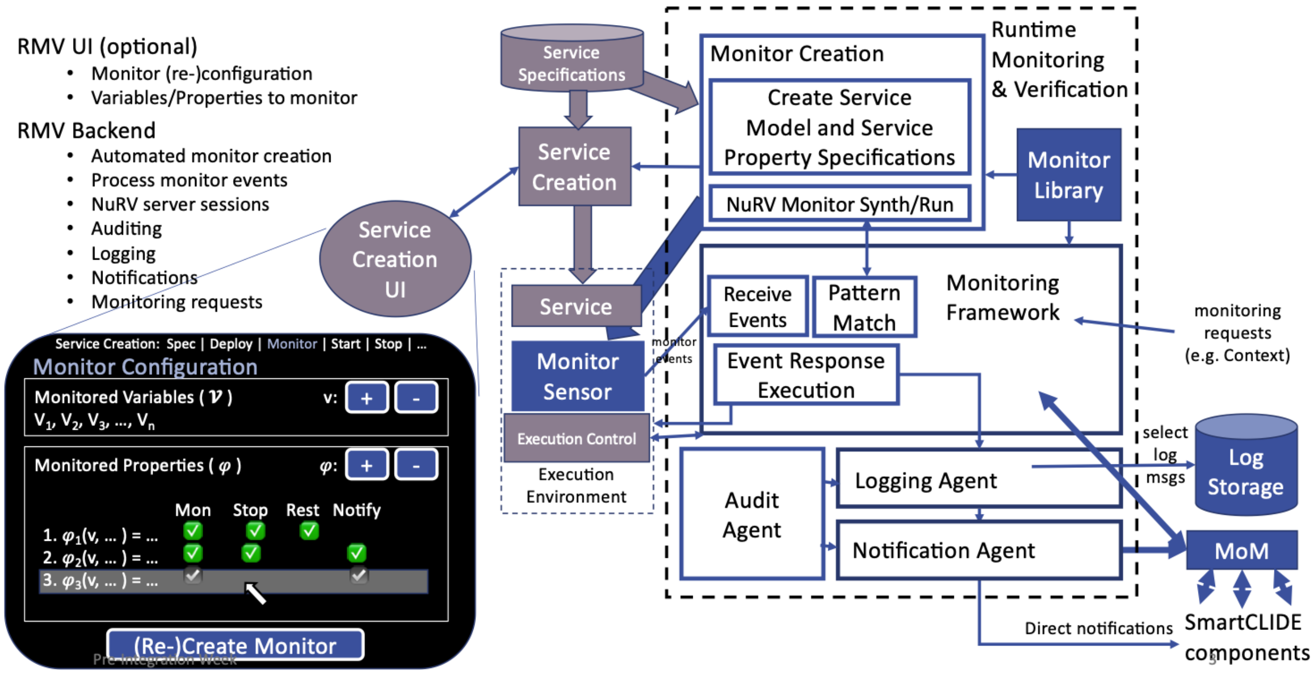

Figure 1: RMV Subsystem Overview

These components include:

1. Monitor Creation - Uses a Service Specification provided from SmartCLIDE along with elements contained in the Monitor Library to construct a property monitor using the NuRV monitor synthesis tool, and a configuration vector for the Monitor Sensor. It stores information about the created monitor in the Monitor Library.

2. Monitor Sensor - A component with versions implemented in various programming languages that provides presence for the monitor within the SmartCLIDE-created service. When the service starts the Monitor Sensor is customized with specifics from the configuration vector. Subsequently the Monitor Sensor generates messages to Monitor Event Processing conveying information about the configured variables it shares with the monitored service.

3. Monitor Event Processing - Receives messages from the Monitor Sensor, which it processes according to the configuration for that monitor that is stored in the Monitor Library. The

configuration may indicate that the values of logical conditions, based on the values of variables within the monitored service, are to be sent to a NuRV property monitor that will return a verdict on whether the monitored property is satisfied, violated, or (as yet) unknown. The result may be sent to other SmartCLIDE components that have registered for notifications.

4. Auditing, Logging, and Notification - Provides the ability to distribute monitoring data and results, to record security-relevant (or other property related) events in a persistent log, and to provide a consolidated auditing, logging and notification service to registered SmartCLIDE or application components.

5. Monitor Library - Contains global definitions and patterns for monitor construction as well as information about the specific monitors that have been constructed. The monitor library is access both at monitor construction time and at monitor execution time.

6. RMV User Interface within the SmartCLIDE Service Creation UI - An optional user interface that can be used by a service developer to modify the configuration of a service monitor, to enable/disable monitoring actions, add/delete monitored variables and properties, and regenerate a modified monitor. Without the UI such changes can also be achieved by editing the generated monitor's configuration vector.

In addition to the ability to monitor created services to assure that they operate within their specifications, the RMV framework provides the capability to construct bespoke monitoring services using the RMV monitor sensor to gather runtime data, that may be used in arbitrary ways by other system services or as part of application services.

References

[CTT19a] Alessandro Cimatti, Chun Tian, and Stefano Tonetta. Assumption-based runtime verification with partial observability and resets. In Bernd Finkbeiner and Leonardo Mariani, editors, *Runtime Verification*, pages 165–184, Cham, 2019. Springer International Publishing.

[CTT19b] Alessandro Cimatti, Chun Tian, and Stefano Tonetta. Nurv: A nuxmv extension for runtime verification. Berlin, Heidelberg, 2019. Springer-Verlag.