Evolution Style: Framework for Dynamic Evolution of Real-Time Software Architecture

Adel Hassan[1(✉)], Audrey Queudet[2], and Mourad Oussalah[1]

[1] University of Nantes, LINA CNRS UMR 6241, Nantes, France
{adel.hassan,audrey.queudet}@univ-nantes.fr
[2] University of Nantes, IRCCyN UMR CNRS 6597, Nantes, France
Mourad.Oussalah@univ-nantes.fr

Abstract. Software systems need to be continuously maintained and evolved in order to cope with ever-changing requirements and environments. Introducing these changes without stopping the system is a critical requirement for many software systems. This is especially so when the stop may result in serious damage or monetary losses, hence a mechanism for system change at runtime is needed. With the increase in size and complexity of software systems, software architecture has become the cornerstone in the lifecycle of a software system and constitutes the model that drives the engineering process. Therefore, the evolution of software architecture has been a key issue of software evolution research. Architects have few techniques to help them plan and perform the dynamic evolution of software architecture for real-time systems. Thus, our approach endeavors to capture the essential concepts for modeling dynamic evolution of software architectures, in order to equip the architects with a framework to model this process.

1 Introduction

With daily changes in technologies and business environments, software systems must evolve in order to adopt to the new requirements of these changes. Generally, the software evolution is a complex process that requires a great deal of knowledge and skills. This is due to the fact that all artifacts produced and used in the software development life-cycle are subject to changes. Since software systems change fairly frequently, it is essential that their architectures must be restructured. With the increase in size and complexity of software systems, the computing community acknowledges the importance of software architecture as a central artifact in the lifecycle of a software system. In this respect, the architecture is specified early in the software lifecycle, and constitutes the model that drives the engineering process [10]. In the evolution process, architecture can elucidate the reason behind design decisions that guided the building of the system. Moreover, it can permit planning and system restructuring at a high level of modeling, where business goals and quality requirements can be ensured and where an alternative scenario of evolution can be explored. Modeling architecture evolution process can support architects in representing reusable practices

B. Tekinerdogan et al. (Eds.): ECSA 2016, LNCS 9839, pp. 166–174, 2016.
DOI: 10.1007/978-3-319-48992-6_12

in a domain-specific architecture evolution. Accordingly, the term *evolution style* has been introduced (Oussalah et al. [2]) as an approach, the aim of which is to capture the main characteristics of a set of activities performed for evolving software architecture. It defines the set vocabulary of concepts necessary in order to model the potential scenarios for evolving a domain-specific software architecture. These scenarios can be grouped together as a library of evolution styles. Both the analysis and comparison of scenarios will assist architects in choosing an evolution scenario for a future evolution [1].

Software evolution is a complex and multifaceted process that requires a number of techniques and skills. This is particularly so when it is required to introduce these changes without halting the system. Some critical issues like synchronous task handling, schedulability analysis, consistency and integrity for dynamic evolution of a real-time system deserve further exploration.

In our previous work [1], we presented the Meta-Evolution Style MES for modeling software architecture evolution (static evolution style), but we did not delve into the dynamic aspects of this process. Therefore, in this paper, we endeavor to probe more deeply into the issues and rules which must be considered when handling dynamic evolution of software architecture in order to extract the needed information, which will be annotated with MES to fulfill the dynamic evolution style. The research ultimately intends to establish a foundational step of a generic process framework for dynamic evolution of software architecture that could provide a means to facilitate analysis and to formally model software architectures and their dynamic evolution processes. Likewise, this framework should provide reusable concepts that could express the way(s) to dynamically evolve software architecture and provide the means to compare these different trajectories.

The rest of the paper is organized as follows. Section 2 briefly reviews related works. Section 3 discusses the real-world issues in integrating architecture changes at run-time [3]. Section 4 extends the meta-evolution style to embrace the concepts of dynamic evolution. Finally, the paper contributions are summarized in Sect. 5.

2 Related Work

The necessity of introducing change at runtime has resulted in different architecture centric approaches for dynamic evolution. Dowling and Cahill [17] present the K-Component model as a reflective framework for building self-adaptive systems. K-Components are components with an architecture meta-model and adaptation contracts to support their dynamic reconfiguration. Cuesta et al. [16] present a reflective Architecture Description Language (ADL) named PiLar which provides a framework to describe the dynamic change in software architecture. It consists of a structural part and a dynamic part, which defines patterns of change. Costa-Soria et al. [12] define a reflective approach for supporting dynamic evolution of architectural types in a decentralized and independent way. Their approach is applied to ADL, in particular to the PRISMA meta-model, in order to develop an evolveable component type that is provided with

an infrastructure to support its evolution at run-time. Romero in his PhD thesis [13], develops a component-based framework support safety replacing the real-time components. The approach implementation is integrated in the OSGi platform in order to exploit the OSGi capabilities like load and unload of code at runtime, and enhances the framework with all the required elements to provide a safe component replacement for real-time characteristics by supporting on-line schedulability analysis. Richardson [9] presents an extension of the OSGi Framework in order to be able to perform dynamic reconfiguration of real-time systems. He proposes an RT-OSGi that can be used to develop real-time systems which are dynamically reconfigurable: by integrating the OSGi Framework with the Real-Time Specification for Java (RTSJ). Unlike these approaches, our work attempts to develop a framework to model such activities and techniques in dynamic evolution of software architecture. This can support architects in analyzing and better understanding the process of introducing changes at run-time.

3 Issues in Dynamic Software Architecture Evolution

Irrespective of the mechanism of an evolution model that is used to perform the dynamic architecture evolution, some issues [3] should be addressed by any approach in order to efficiently handle the dynamic evolution process. This section presents these issues in order to annotate MES with the required information, which will be extracted from these issues to fulfill the requirements of dynamic evolution modeling.

3.1 Safe Stopping of Running Artifacts

One of the main issues that must be considered when handling dynamic evolution is to leave systems in a consistent state after a change is performed. Evolving an artifact at runtime without considering its thread may disrupt or suspend its service for an arbitrarily long time, which can lead real-time tasks to miss some deadlines. Detecting when it is safe to actually evolve the artifacts is a crucial key to guarantee that the system will not encounter an inconsistent state. Therefore, various strategies have been defined in order to tackle this issue, namely Quiescence [4] and Tranquility [5]. They differentiate the passive state from the active state of software artifact and assume that an affected artifact should be placed into a passive state before performing the evolution operation.

Generally, a real-time system consists mainly of a set of elements which provide or/and create real-time services (threads). These real-time tasks can be periodic, aperiodic or sporadic [15], depending on how their corresponding jobs are activated. The passive elements in a real-time system are those that do not have any execution thread, but typically provide services for other elements. The quiescence and tranquility techniques can fit the dynamic evolution of these passive elements. On the contrary, the active elements are those that have active real-time threads; these techniques [4,5] require that elements should be shifted

into a passive state in order to be modified. This means that the real-time threads in the element would need to be suspended, thus potentially resulting in deadline misses for the threads of the element being under evolution. This behavior is, of course, undesirable for hard real-time systems. Therefore, the evolution operation should respect the timing constraints of the active element that is subject to change. In this respect, the evolution operation execution time is part of the timing constraints of the real-time system itself. Therefore, it must not exceed the safe state time of this element, i.e. the maximum duration of the evolution operation should be less than the minimum separation between two consecutive jobs of this element's task.

3.2 Transferring State

Another crucial issue of system consistency that should be considered when addressing a dynamic evolution is that of handling stateful elements (components that may have internal information, or connectors that may have buffers full of messages [6]). In the case of replacing elements, information integrity requires that the state of the old element must be transferred, or possibly transformed (in the case the data structure is different), to the new element. Meanwhile, this step is not required for the replacement of stateless elements. This activity can be more complex if the internal structure of the two elements is different, which requires identifying and extracting the relevant data from the old element. These date will be modified to fit the new element.

Practically, it is difficult to develop a generic abstract that can fit the internal data structure for all the system elements, in order to store the state of any element during its evolution. Therefore, preserving and transferring or transforming the internal state of elements is a specific step. Thus, if a transfer state is required, this process should be specifically remedied with each operation.

3.3 Change Management

Another issue in dynamic evolution is relevant to the mechanism to preform this process and how the changes are driven (activeness of change), how evolution events can be detected, then how the suitable reactions can be effected. More accurately, it relates to how this process can be managed.

Generally, the software system can be reactive (changes are driven externally), or proactive (drives changes to itself) [7]. Thus, either the system is instrumented with change management, or with an interface to allow an external agent to dynamically introduce the changes. In dynamic evolution, management can be represented as an evoluter (Role) who is responsible for dynamically performing the evolution. This can be formulated as a controlling system that monitors a controlled system in order to detect and analyse an evolution event when it occurs on the controlled system or its environment to select or synthesize the appropriate action or scenario of evolution. The dynamic MES should provide a modeling concept to express both the controlling techniques for proactive and reactive system.

3.4 Dynamic Evolution Scheduling

The issue of the timing constraints is more important when we handle a dynamic evolution of a hard real-time system, which needs to maintain high levels of application availability. In fact, whatever the change management system be used, the dynamic evolution operation is considered as a real-time task, and once this unscheduled task occurs, it should not affect the timing constraints of system's tasks.

System tasks are scheduled and executed according to their dynamic priorities. Indeed, whatever the system tasks are, the evolution operation should interact/behave without compromising the system tasks' completion. Generally, tasks in a hard real-time system have higher priority than the evolution operation, which is usually handled as a background task (with lower priority). In this aspect, if a background priority task is used to evolve an element, this evolution task can be preempted by any higher priority task (including a task from the element under evolution), which can lead to an unsafe state or loss of the internal state of the element. Therefore, an evolution task should directly derive its priority from the element that undergoes its change. Thus, the management change should be able to safely handle the evolution tasks while still guaranteeing the timing constraints of the system, e.g. it should dynamically prioritize this unexpected event (evolution operation) within the system threads.

Furthermore, in the replacement and addition operations, it is necessary to guarantee that the new element threads have taken over the role of the old element threads without deadline violation, which also requires a dynamic rescheduling in order to integrate the new element threads with the rest of the system's threads in the scheduler.

4 Dynamic Meta Evolution Style

MES consists in defining foundational meta-concepts for describing a software architecture evolution. These essential concepts were used in modeling and analyzing static evolution styles [1]. MES can be refined to any other kind of architecture evolution. In this sense, the intent is not to define a new meta style for modeling dynamic evolution of real-time and embedded systems, but to annotate MES with information required to analyze and model this process. Hence, this work focuses on integrating the concepts of dynamic interaction and schedulability analysis into evolution styles. Figure 1 illustrates our proposition to extend MES with the necessary information to fulfill the requirement for modeling the dynamic evolution styles (gray boxes refer to MES elements; white boxes refer to proposed additional elements). Actually, dynamic evolution of a real-time system requires introducing changes in bounded time. Managing and performing this process without violating the timing constraints is more complex. This requires a fast, interactive Role (intelligent change management) which minimizes or eliminates the human intervention Role and shifts it from operational to strategic. Thus, the Role in dynamic MES should support the concept of

an automatic Role either as an internal instrument or an external agent. That satisfies the change management requirement for dynamic evolution.

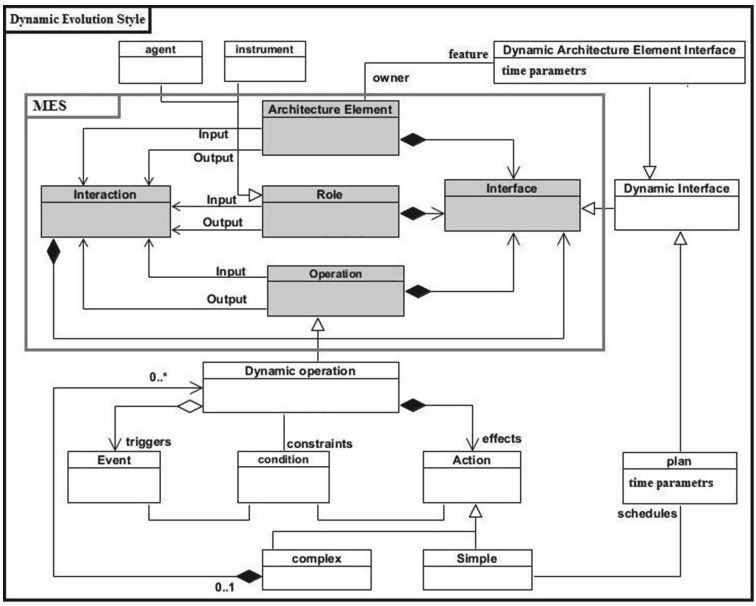

Fig. 1. Dynamic-evolution style

Indeed, the needs of an architectural element to change are required when an evolution event has occurred. Therefore, the unexpected evolution events must be assigned with their potential scenarios of reactions (evolution paths). A strategy to synthesize the suitable reactions is defined such that all the affected elements complete within their deadlines. In this sense, each evolution path consists of a series of evolution operations and represents one choice to evolve the architecture from the current state to the target state. Therefore, each simple evolution operation must have a dynamic interface which provides the necessary parameters (priority, time execution) in order to be safely handled and not cause timing misbehavior. Several scheduling methods for the handling of unexpected events in real-time systems have been proposed in the literature in order to service aperiodic requests, where a set of hard aperiodic tasks is scheduled using the Earliest Deadline First (EDF) algorithm [8]. Among them, the Total Bandwidth Server (TBS) [14] and Earliest Deadline as Late as possible server (EDL) [11] both provide an efficient aperiodic service under EDF. In TBS, worst-case execution time of aperiodic requests must be known in advance (which is not the case of care evolution operation). That is why we will turn to EDL to dynamically schedule this unexpected event jointly with the system threads. Thus, the Operation should provide the necessary parameters that are needed by a dynamic scheduling algorithm in order to be scheduled.

The architectural element that can be changed at its active state must also have the suitable (dynamic) interfaces to provide the required parameters in order to be dynamically evolved. An interface is needed to handle the internal state of the element during the replacement Operation. Another interface is also needed to provide the time parameters for guaranteeing the safe stopping. These scheduling parameters are required by the scheduler to dynamically schedule the evolution operation and the threads of the new elements: Worst Case Execution Time (WCET), deadline and release time. These parameters allow the schedulability analysis of dynamic evolution of hard real-time systems.

Role: Generally, a Role is responsible for the evolution operation that performs the changes. Managing and performing at run-time requires a highly interactive Role (external agent or internal instrument).

Dynamic Operation: A dynamic evolution operation can be a simple evolution like add or delete, or a composite one like replacement. A dynamic evolution process should be expressed in such a way that it supports both kinds of activeness, namely proactive and reactive. This can be achieved by separating evolution requests (Events) from the evolution mechanisms (Actions). Therefore, the construct of evolution operation is based of the ECA rules "On Event If Condition Do Action At Time" which means: when an evolution Event occurs, if Condition is verified, then execute suitable Action at appropriate time. The dynamic operation must offer a dynamic interface (plan) which provides relevant run-time parameters that are needed to schedule the operation as soon as possible within the system tasks.

Dynamic Architecture Elements: An architecture element must be evolutionary open, which means it has an interface with the necessary parameters that enables it to dynamically react to evolution operation. An element should be able to provide its scheduling parameters to allow the Role to dynamically effect the changes without breaking the timing constraints of the system.

Interaction: In fact, the dynamic evolution is a real-time task, so the interaction element must guarantee that evolution Operations are subject to the timing constraints. The interaction element ensures the availability of required interfaces and parameters among elements (Operation, Architecture Element, Role) in the process.

Dynamic Interface: A dynamic element should have appropriate interface which provides the required parameters to efficiently interact at run-time. Such an interface is required, for example, to allow the Instrument (the Role in self-managing system) to observe an architecture element in order to detect any evolution event or to determine the appropriate time to effect the changes.

Process: Represents the dynamic configuration of the evolution elements which transfers a software architecture from its current architecture style to a target style. This configuration provides the temporal and topological organizing of evolution operations while respecting the consistency and integrity of the architecture elements.

5 Conclusions

In this paper, we propose a dynamic evolution style for specifying the dynamic evolution for software architecture. Our intent is to provide a style sufficiently rich to model the dynamic changes in software architecture of a real-time system and to be able to represent the potential ways of performing these changes. To better realize this intent, we integrate the behavior concepts of dynamic changes into the MES so we can have a sound understanding of dynamic evolution issues and constraints, which is a prerequisite to developing a modeling environment that supports dynamic evolution styles. Our ongoing work is devoted to developing this environment.

References

1. Hassan, A., Oussalah, M.: Meta-evolution style for software architecture evolution. In: Freivalds, R.M., Engels, G., Catania, B. (eds.) SOFSEM 2016. LNCS, vol. 9587, pp. 478–489. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49192-8_39
2. Oussalah, M., Tamzalit, D., Le Goaer, O., Seriai, A.: Updating styles challenge updating needs within component-based software architectures. In: SEKE (2006)
3. Oreizy, P.: Issues in modeling and analyzing dynamic software architectures. In: Proceedings of the International Workshop on the Role of Software Architecture in Testing and Analysis (1998)
4. Kramer, J., Magee, J.: The evolving philosophers problem: dynamic change management. IEEE TSE **16**(11), 1293–1306 (1990)
5. Vandewoude, Y., Ebraert, P., Berbers, Y., D'Hondt, T.: Tranquility: a low disruptive alternative to quiescence for ensuring safe dynamic updates. IEEE Trans. Softw. Eng. **33**(12), 856–868 (2007)
6. Oreizy, P., Medvidovic, N., Taylor, R.N.: Runtime software adaptation: framework, approaches, and styles. In: Companion of the 30th International Conference on Software Engineering, pp. 899–910. ACM (2008)
7. Buckley, J., Mens, T., Zenger, M., Rashid, A., Kniesel, G.: Towards a taxonomy of software change. J. Softw. Maint. Evol. Res. Pract. **17**(5), 309–332 (2005)
8. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM (JACM) **20**(1), 46–61 (1973)
9. Richardson, T.: Developing dynamically reconfigurable real-time systems with real-time OSGi (RT-OSGi). Ph.D. dissertation, University of York (2011)
10. Garlan, D., Perry, D.E.: Introduction to the special issue on software architecture. IEEE Trans. Softw. Eng. **21**(4), 269–274 (1995)
11. Chetto, H., Chetto, M.: Some results of the earliest deadline scheduling algorithm. IEEE Trans. Softw. Eng. **15**(10), 1261–1269 (1989)
12. Costa-Soria, C. Hervás-Muñoz, D., Pérez, J., Carsí, J.Á: A reflective approach for supporting the dynamic evolution of component types. In: 14th IEEE International Conference, pp. 301–310 (2009)
13. Romero, C., J.Á: Contributions to the safe execution of dynamic component-based real-time systems. Ph.D. dissertation, Carlos III University of Madrid (2012)
14. Spuri, M., Buttazzo, G.: Scheduling aperiodic tasks in dynamic priority systems. Real-Time Syst. **10**(2), 179–210 (1996)
15. Li, Q., Yao, C.: Real-Time Concepts for Embedded Systems. CRC Press, Boca Raton (2003)

16. Cuesta, C.E., de la Fuente, P., Barrio-Solórzano, M., Beato, E.: Coordination in a reflective architecture description language. In: Arbab, F., Talcott, C. (eds.) COORDINATION 2002. LNCS, vol. 2315, pp. 141–148. Springer, Heidelberg (2002). doi:10.1007/3-540-46000-4_15
17. Dowling, J., Cahill, V.: The K-component architecture meta-model for self-adaptive software. In: Yonezawa, A., Matsuoka, S. (eds.) Reflection 2001. LNCS, vol. 2192, pp. 81–88. Springer, Heidelberg (2001). doi:10.1007/3-540-45429-2_6