Safety component-based approach and its application to ERTMS/ETCS on-board train control system

Marc Sango, Christophe Gransart, Laurence Duchien

Safety component-based approach and its application to ERTMS/ETCS on-board train control system

Marc Sango[a,b,c*], Christophe Gransart[a,b], Laurence Duchien[b,c]

[a]*IFSTTAR, COSYS/LEOST, Villeneuve d'Ascq - France*
[b]*Univ Lille Nord de France*
[c]*INRIA Lille – Nord Europe*

Abstract

Safety-critical software is becoming more and more complex and at the same time it operates in frequently changing environments on which it reacts by reconfiguring its architecture. Thus, an appropriate modelling approach is needed to reduce the complexity of designing and to enable the verification of dynamic reconfiguration behaviour before the deployment at runtime. The paradigm of software component-based engineering provides an essential support for this. However, composing software from many reconfigurable components can lead to a huge number of possible compositional configurations difficult to handle at design time. Moreover, analysing all possible sequences of reconfiguration, including failure situations, is far beyond feasibility without an appropriate abstraction and granularity levels. In this paper, we propose a hierarchical component-based design approach to reduce the complexity of designing and to analyse the dynamic reconfiguration behaviour. We illustrate our approach with a case study derived from ERTMS/ETCS level 2.

Keywords: Software components; reconfiguration; safety-critical; railway domain; ERTMS/ETCS.

Résumé

Les logiciels des systèmes critiques deviennent de plus en plus complexes et dans le même temps, ils fonctionnent fréquemment dans les environnements dynamiques dans lesquels ils doivent s'adapter en reconfigurant leur architecture. Ainsi, une approche de modélisation appropriée est nécessaire pour réduire leur complexité de conception et faciliter la vérification de leurs comportements dynamiques avant leur déploiement. Le paradigme des composants logiciels offre un support adéquat à cet effet. Cependant, la composition d'un logiciel à partir de nombreux composants reconfigurables tout en analysant toutes les séquences possibles de reconfiguration, y compris les situations de défaillances, peut devenir rapidement infaisable sans un support d'abstraction et de granularité acceptable. Dans cet article, nous proposons une approche hiérarchique à base de composants qui permet de réduire la complexité de conception et de faciliter l'analyse des comportements dynamiques. Nous illustrons notre approche sur un cas d'étude tiré des exigences ERTMS/ETCS niveau 2.

Mots-clé: Composants logiciels; reconfiguration; sûreté; système critique ferroviaire; ERTMS/ETCS.

* Corresponding author information here. Tel.: +33-320-438-333; fax: +33-320-438-359.
E-mail address: marc.sango@ifsttar.fr

1. Introduction

Software in safety-critical embedded systems, such as aeronautic, railway and automotive, is becoming more and more complex, providing increasing requirements. Requirements are composed of functional requirements representing the functionalities themselves and non-functional requirements describing additional properties that have to be achieved by the software systems, such as dependable systems (named RAMS for Reliability, Availability, Maintainability and Safety). Moreover, in safety-critical systems, failures could result in the critical undesired situations that may lead to serious loss of life or unacceptable damage to the environment. Some of these undesired situations can depend on factors that are not under the control of the system. They are called hazards (EN 50128, 2011). Therefore, the safety must be taken into account to ensure that the risk of danger, including hazardous situations, is acceptably low. As a consequence, the widely agreed definition of safety combines the frequency of occurrences for the hazardous situations with the severity of their consequences in order to determine the acceptable risk levels. This combination is determined by risk analysis to identify the safety requirements. Therefore, the aim of safety-critical software system construction is to develop a system so that it fulfils all of its safety requirements. This includes the following global steps:

- Step 1: Identification of system level hazards based on risk criteria (identification of hazards),
- Step 2: Determination of the acceptable hazards to meet the risk acceptance criteria (safety requirements),
- Step 3: Taking constructive measures in order to respect the safety requirements (development activity),
- Step 4: Proof that all of these safety requirements are fulfilled (safety cases).

Considering steps 1 and 2, there is an established set of safety analysis techniques (e.g., Failure Modes and Effects Analysis (FMEA), Fault Tree Analysis (FTA), etc.). Most of them have been developed at a time when safety-critical tasks were exclusively performed by purely mechanical or electrical systems and do not especially consider the new aspects introduced by software control. However, some of them have been adapted, such as Component concept for Fault Trees (CFT) (Kaiser et al., 2003), in order to consider the new aspects introduced by system built from components. The different safety analysis techniques can be classified by different categories (kinds of qualitative and quantitative analysis) for different purposes (Clifton, 2005).

Considering step 3, software development activity in safety-critical embedded systems, many approaches have also been established. For instance, we can distinguish the diversification approaches, e.g., a safety-bag approach applied in Elektra (Kantz & Koza, 1995), from component-based approaches, e.g., in embedded systems (Crnkovic et al., 2011). One of the advantages of diversification approaches is that there is a clear separation of software control concern from safety conditions concern, each concern being developed in different channel. In Elektra, to keep common failure modes of the software design as small as possible, the two channels are implemented using diverse specifications and different programming paradigms. Indeed, the channel of software control functions is implemented according to the procedural paradigm based on a functional specification, while the channel of safety conditions is implemented according to the rule-based paradigm based on a specification derived by a specific national operating regulation (e.g., Austrian Federal Railways). Moreover, the required reliability and availability are achieved by actively triplicated hardware in each channel.

Since common transport systems, particularly railway systems, are becoming increasingly complex (including complex software control functions and being mass-produced), solutions like n-version-programming and expensive explicit hardware redundancy channels are too expensive to be applied and are not always the cost-efficient development solutions. As a consequence, the cost-efficient development for the complex dependable systems remains one of the major challenges of the railway industry. For example, a promising approach for the cost-efficient development of safe and available systems consists in keeping up functionalities as long as possible without requiring expensive explicit redundancy channels. This approach can be provided by exploiting the implicitly available redundancy of software control functions defined in Systems Requirement Specification (SRS) and by using dynamic reconfiguration in the component-based approach to realise survivability. In fact, in the case of errors, system safety-critical functionalities are kept alive while uncritical functionalities can be switched off in order to adapt to failure situations. The approach, which consists in guaranteeing the continuous satisfaction of functional and extra-functional contracts under changing conditions of software execution, is addressed by self-adaptive software engineering. In this context, the software control loop model (e.g., IBM's feedback control loop model, named MAPE-K for Monitoring-Analysis-Planning-Execution and shared Knowledge) has been adopted as a central concept to self-adapt software architecture (Kephart & Chess, 2003).

Our goal is to use the codified IBM MAPE-K, inspired by the principles of control theory, in the paradigm of Component-Based Software Engineering, named CBSE (Szypersky et al., 2002), in order to save the costs of development, ensure the safety of the system and achieve the acceptable availability. However, when applying the component-based approach in the context of safety-critical embedded systems, some new challenges arise. In this paper we address some of these challenges.

- Challenge 1: Due to the nature of safety-critical embedded systems, an appropriate component model should be able to describe the correct behaviours and failure behaviours at design time. Indeed, the predetermination of the failure behaviours at design time facilitates the analysis of component reconfiguration behaviours before their deployment at runtime.
- Challenge 2: The second challenge corresponds to the complexity of designing by composition. The benefit of the component-based approach increases with the number of components. However, when the associated composition of components becomes particularly complex, the resulting set of compositional configurations is no longer efficiently analysable without an appropriate abstraction and granularity levels.

To overcome these challenges, we define a new hierarchical component-based approach for safety-critical software. Its benefits are to find the right abstraction and granularity levels that make a model expressive enough and yet analysable. The contribution relies on two parts, which can be integrated into the overall development process of the safety-critical embedded system.

- Part 1 concerns the predictable specification of failure behaviour of components, based on the predefined basic quality types, which capture the common failure modes.
- Part 2 concerns dynamic reconfiguration abstraction mechanisms, based on dynamic assignment of priority on configurations. Dynamic reconfiguration in the safety-critical software is acceptable as long as the safety of the system is assured and if it does not lead to a user-perceivable degradation of system functionality.

The remainder of this work is structured as follows. In section 2, we motivate the application of our approach in the railway safety-critical application domain. In section 3, we present the contribution for Part 1; after giving an overview on our approach, we introduce the modular concept for the specification of predetermined failure behaviour of components. Section 4 addresses the contribution for Part 2, by providing the compositional reconfiguration behaviour mechanism that will be elaborated to realise components reconfiguration. In section 4, we use some established component-based safety requirements to evaluate our approach and to compare it with some related works. Finally, section 6 concludes and points out the perspectives.

2. Railway application domain

The safety-critical system is characterised by a Safety Integrity Level (SIL), which defines an acceptable rate of safety-critical failures (IEC 61508). For example, according to railway standard EN 50128, the railway software is fail-safe software if the probability of failures is less than 10^{-9} per hour. Automatic train control systems, whose purpose is to run a train by protecting it from dangerous situations, must be in the highest SIL level (i.e., SIL 3 or SIL 4). To explain our approach, we derived our use case from European Rail Traffic Management System/European Train Control System requirements (ERTMS/ETCS, 2006). Due to the nature of the required control functions, the SRS of ERTMS/ETCS defines two sub-systems: the on-board and the trackside. Depending on the application level, the trackside can be composed of many devices (e.g., balises or RBC: Radio Bloc Center) that generate a Movement Authority (MA) and transmit it to the on-board (e.g., ETCS), see Fig. 1a. For example, in application level 2, the train is controlled by balises and radio, while in application level 3 the train are expected to be self-sustaining vehicle localised by radio. This level 3 is not yet completely specified.

In the case of fault, in application level 2, it shall be the responsibility of the trackside sub-system to send additional information in the MA to the on-board sub-system. The trackside equipment can send the distance to the End Of Authority (EOA) with target speed equal to zero or target speed higher than zero. In the latter case, the EOA is called the Limit of Authority (LOA). It is the responsibility of the trackside to ensure that the safe distance beyond the EOA/LOA is long enough to brake the train from the target speed to a standstill in order to avoid any hazardous situations. Therefore, it is the responsibility of the on-board equipment to apply the brake if no new information is received when the EOA/LOA is passed. According to the value of information, it shall be possible for the on-board equipment to apply three reactions: no reaction, service brake and emergency brake.

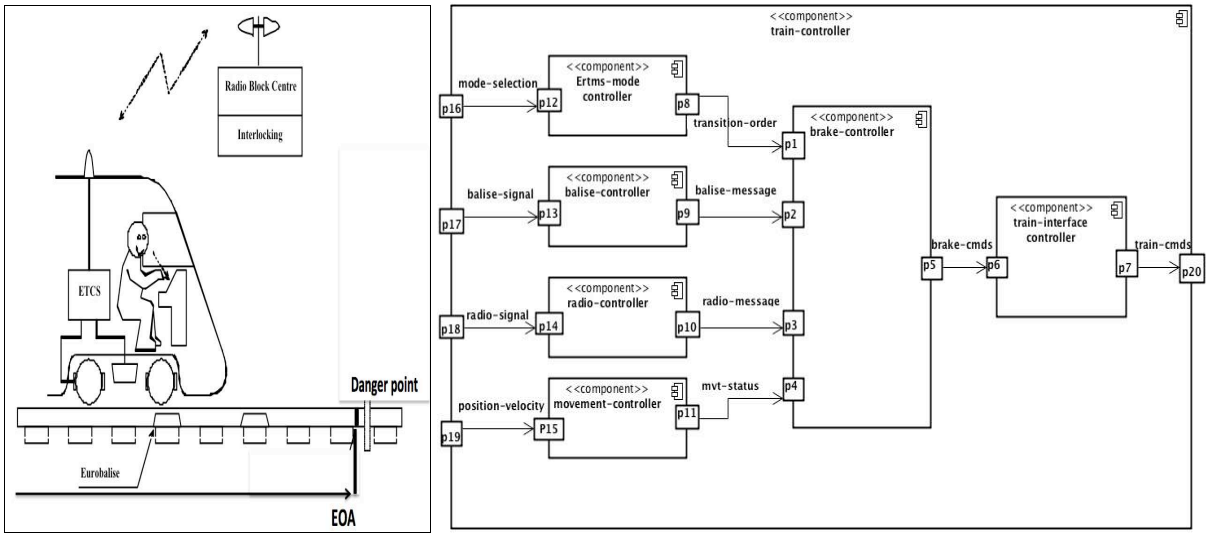

Fig. 1. (a) ERTMS/ETCS application level 2 (left); (b) The basic structure diagram of train controller (right)

For example, according to the SRS of ERTMS/ETCS paragraph 3.14.1.2. (in short SRS §3.14.1.2): *"In case only the application of (the non-vital) service brake has been commanded and the service brake fails to be applied, the emergency brake command shall be given"*. As a consequence, one of the main safety-critical components concerning the on-board control software is the brake controller software named brake-controller, shown in Fig. 1b. It communicates with other components by means of mobile radio communication from RBC or telegram communication from balise group. Fig. 1b shows a very basic component-based architecture, as an embedded software engineer would use it to describe the basic train-controller software containing the brake-controller. This component-based architecture describes the set of software components and their ports of communication (e.g., p1, p2, etc.). The interconnections between ports are the paths of information flow (e.g., radio-message, transition-order, etc.). In our consideration, the functional information flow is the data flow (e.g., radio-message) or the control flow (e.g., transition-order) built from ERTMS/ETCS language (ERTMS/ETCS, 2006).

During the design phase, the functional behaviour model, for example the state machine model, is used to specify the reaction of components to information flow received via their ports. It is also necessary to specify the model of degraded behaviour (or failure behaviour) in order to predict the possible abnormal deviations from the intended functional behaviour that can lead to dangerous failure situations. The specification of components reconfiguration behaviour can be used to describe the degraded behaviour. However, composing a system from many reconfigurable components can lead to a huge number of possible system configurations, inducing a complexity that can be difficult to handle during system design time. For example, a reconfiguration framework of our basic component-based architecture (Fig. 1b) must control 6 components; if each component has 3 different states, then there are $6^3 = 216$ different states. We can easily imagine the consequences for real on-board ERTMS/ETCS systems; it is very complex to consider all states and analyse all possible sequences of reconfiguration. Moreover, analysing all sequences of reconfiguration, including failure situations, is far beyond feasibility without an appropriate abstraction and granularity levels. In order to have an expressive enough and yet analysable model, we introduce our hierarchical component-based approach specialised for SAfety-critical RAilway system, named the SARA approach.

3. SARA approach

SARA is founded on general component-based software engineering principles and inspired by the predictability-enabled component technology approach (Hissam et al., 2005) and reflective Fractal component model (Bruneton et al., 2006). The leitmotiv of our approach is the separation of concern and the preservation of separated requirements throughout the software life cycle in order to simplify the process of verification or simulation, which is the basis to facilitate the certification process. In this section, after giving an overview of our approach in term of metamodel, we introduce the modular concept to specify the predetermined failure behaviour in a concrete component instance example.

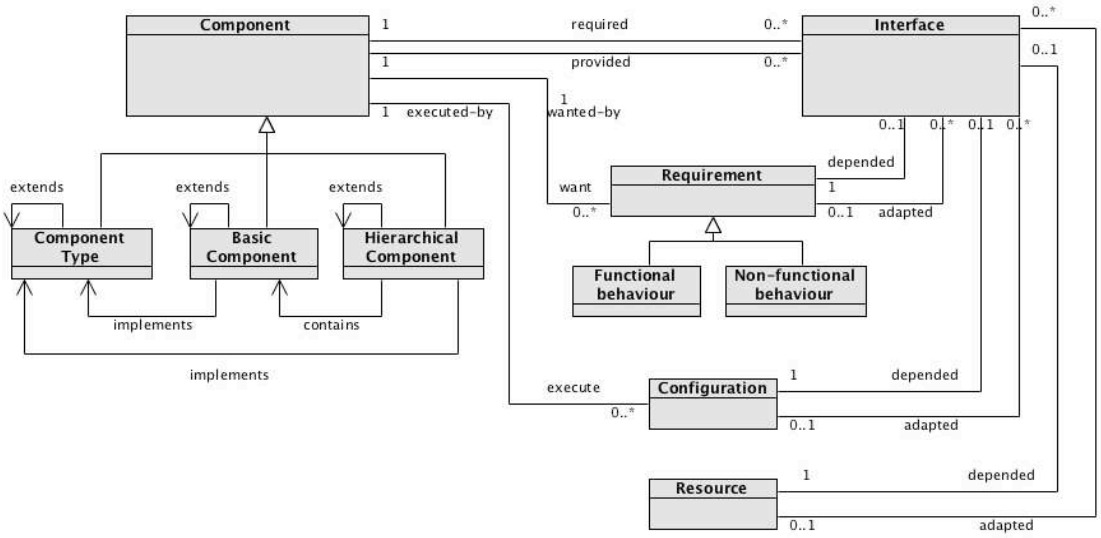

Fig. 2. SARA metamodel in UML class diagram

3.1. The Metamodel

Fig. 2 shows the core of SARA metamodel. The metamodel extends the regular internal behaviour of components (content that encapsulates its inner structure and functional behaviour derived from a requirement), as well as the regular external behaviour with the reconfiguration behaviour (interface annotated by non-functional properties). In fact, like (McKinley et al., 2004), we distinguish especially components functional behaviour from their reconfiguration behaviour. In our component model, in term of granularity, we explicitly distinguish the basic components that encapsulate directly a stately behaviour specification from the hierarchical components that do not contain an explicit behaviour specification themselves but contain other components called the subcomponents. A basic component and a hierarchical component are the two design entities that implement the abstract component type. A component type, used at design time, is instantiated to a component instance at runtime for simulation or runtime execution. Via a binding of required interfaces and provided interfaces, a component can trust safely to another component to fulfil behaviours, to achieve configurations or to share resources. When a component cannot achieve a predefined active configuration, it switches or reconfigures to another predefined adapted configuration.

A metamodel is instantiated in a component model and can have a graphical representation in term of components diagram and quality types diagram. A components diagram is derived from a system architecture. An example of components diagram is a component-based architecture shown by Fig. 1b, used to illustrate a very basic ERTMS/ETCS on-board software architecture. A quality types diagram can be derived from some established safety analysis techniques like the specific fault tree analysis. In the following, we will focus on the specification of a concrete basic component annotated by an example of quality type variables. The goal is to show that developers can use the component design entities to well define their software by predicting failure situations and the component instance configurations to analyse reconfiguration behaviours of components.

3.2. Specification of components with MAPE-K loop

To make the software architecture explicit and to enforce a clear separation between the control interfaces and the functional interfaces, as done in Fractal component model (Bruneton et al., 2006), we separate the component controller part, named membrane, from the functional implementation part, named content, as illustrated in Fig. 3a. We use the previously mentioned IBM's MAPE-K loop model as a feedback control support to provide the autonomous reconfiguration support for the components and to maintain the clear separation between component's functional behaviour and its reconfiguration behaviour. In our approach, the "K" element, i.e., Knowledge, encloses the content of a component and the "MAPE" elements, i.e., Monitor, Analyser, Planner and Executor, included in the membrane of a component, correspond to the dynamic reconfiguration manager of a component, as illustrated in Fig. 3a and described in the following.

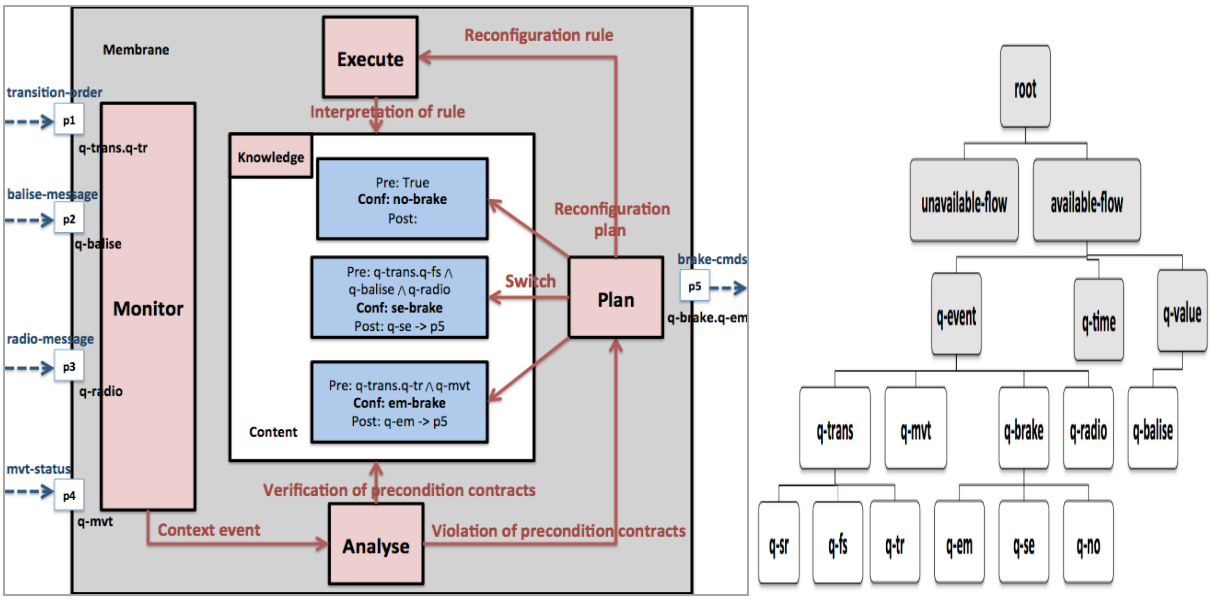

Fig. 3. (a) The basic brake controller component (left); (b) The basic quality type system (right)

Knowledge. The knowledge element is responsible to explicit at design time the relevant knowledge necessary to perform reconfiguration. Thus, the content of a basic component provides the necessary reconfiguration information in order to make the adequate decisions in case of failures. The component content is composed by its different modes of operations, called here configurations, not to be confused with the configuration in terms of components assembly. For example, considering the requirement SRS §3.14.1.2, mentioned in section 2, three configurations can be distinguished: the no brake reaction (no-brake), the service brake (se-brake) and the emergency brake (em-brake), listed in increasing priority order of brake intervention inside the content of the basic component named brake-controller (see Fig. 3a). Each configuration is guarded by a precondition and a postcondition, which is formed by the quality variables. The quality variables (e.g., q-brake) predict the possible deviations (i.e., deviations that can lead to dangerous failure situations) from the expected information flow (e.g., brake-cmds). Fig. 3b illustrates our formalisation of the common failure modes (service provision failure, service timing failure and service value failure) (Bondavalli & Simoncini, 1990). The formalisation is given via an inheritance relation between the basic quality types. The basic quality types capture the deviations from an available functional flow (available-flow). These basic quality types are expanded according to application requirement in order to capture the precise deviation. For example, the quality variable q-brake can be expanded by q-no for no brake, q-se for service brake and by q-em for emergency brake. Thus, one of the expanded quality variable (e.g., q-brake.q-em) can be attached to the output port p5 of the brake-controller, as illustrated in Fig. 3a.

Monitor. The monitor is responsible for sensing the potential change in the external context of a component. In our approach, this change is sensed by quality type, e.g., q-trans captures the deviation from input flow transition-order in Fig. 3a. Based on the change, the monitor notifies the relevant context event (e.g., "balise-message + q-trans") to the analyser. In relation with ERTMS/ETCS, "balise-message + q-trans" can correspond to a message including the variable T_NVCONTACT (SRS §7.5.1.74) that indicates the reaction to be performed when the timer elapses. For instance, the value "00" corresponds to train trip and the value "01" corresponds to service brake.

Analyser. The analyser, based on the context event value and the high-level safety requirements to fulfil (specified in term of precondition contract, e.g. q-trans.q-tr /\ q-mvt), determines whether a reconfiguration must be triggered (e.g., em-brake configuration). The switching would occur only when the reconfiguration events notify about changes that violate the precondition contract. The reconfiguration events can be classified by patterns. For example, in our basic brake-controller, the reconfiguration events can be three message types: driven-normal, driven-at-high-speed and driven-at-danger-point. The first one is only an informative message, while the two other are the request messages for reconfiguration. In relation with ERTMS/ETCS, driven-at-danger-point message can be an emergency message defined in ERTMS/ETCS language.

Planner. Once notified with a reconfiguration event from the context analyser, a reconfiguration event can only be treated if the reconfiguration plan is specified. The developer can specify the sequence of discrete operations in ECA rules, i.e., on Event if Control then Action. For example, if the current configuration is the no-brake configuration and if the driven-at-danger-point reconfiguration message is notified then send the reconfiguration rule become-priority-configuration (em-brake). The reconfiguration plan can be written in a script file interpreted by reconfiguration script interpreter like what is done in Fractal (Bruneton et al., 2006).

Executor. Upon reception of a reconfiguration rule, the executor interprets it and executes the emergent configuration. For example, when receiving the reconfiguration rule become-priority-configuration (em-brake), the executor executes the code of the configuration em-brake (see Fig. 3a.). Finally, for the computation and communication models we have adopted the clocked synchronous model (Jantsch, 2004). At the beginning of a clock cycle, all components instantaneously read their inputs. Then, consuming one clock cycle, they compute their outputs, which are available at the beginning of the next cycle. Thus, the clocked-synchronous model allows decoupling the qualitative analysis of the reconfiguration behaviour from the analysis of the timing behaviour at design time. Then, if the designed reconfiguration behaviour has passed verification, a dedicated runtime framework coordinating all adaptation processes implements the behaviour specification. The advantage of the clocked-synchronous model is that it is an efficiently analysable abstraction of runtime reconfiguration.

4. Analysable abstraction of runtime reconfiguration

As shown in the previous section, a reconfiguration event can only be processed by the planner element of MAPE-K loop if the dedicated reconfiguration plan is specified. This specification remains reasonable for a basic component. However, as mentioned in section 2 and illustrated in the following, specifying the reconfiguration behaviour for a composition of components inside a hierarchical component is very complex without an appropriate abstraction support.

4.1. Scenario of compositional reconfiguration

In section 2, the basic component-based architecture of hierarchical component named train-controller is presented in Fig. 1b. All its subcomponents are specified as what is done for a basic component brake-controller (Fig. 3a.). Fig. 4 illustrates an abstraction of compositional reconfiguration sequences used to compensate the potential failures in train-controller. It shows a lifetime of failure modes and the lifeline for each subcomponent. For each time step, illustrated by a pair of horizontal lines, the currently active configuration of each subcomponent is shown. As we already mentioned, we assume that all components operate in a clocked-synchronous model, such that, in every clock cycle, all components simultaneously read their inputs and compute their outputs. We also assume that, at any start of mission for the train, the emergency brake command shall be tested. When the test is finished, the train-controller shall request an acknowledgement from the driver, via a digital interface machine (display-dim, see Fig. 4), for running under supervision of the selected mode. When the driver acknowledges, the brake is released and the mission starts in the staff responsible mode (sr-mode). Thus, at time step t1, all components are in their active configuration, as all required variables of a message have an appropriate quality.

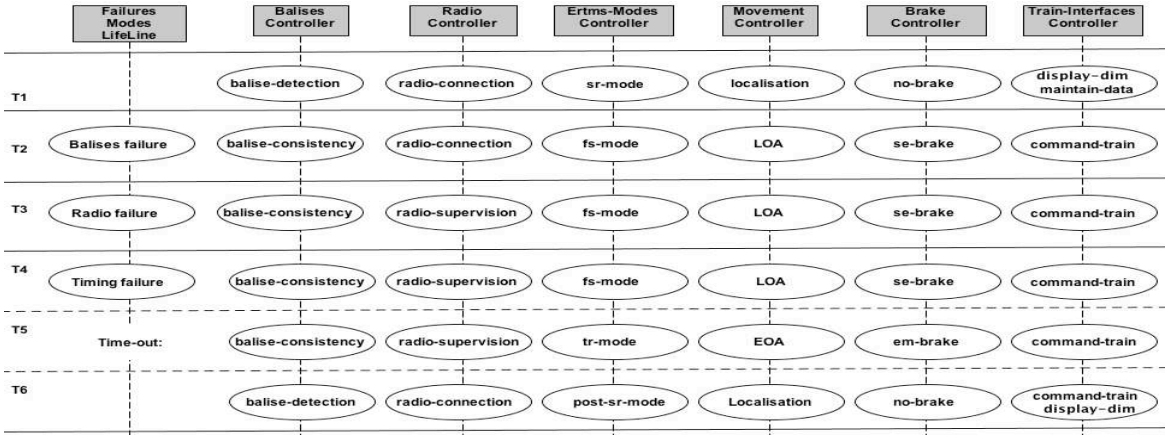

Fig. 4. Scenario of compositional reconfiguration

In an active ERTMS context of utilisation mode and in a specific failure situation, the train-controller initiates a compositional reconfiguration sequence to supervise a train in order to avoid a danger point. For example, at time step t2, we assume that the train and trackside data, which are required for a complete supervision of the train, are available on-board. Therefore, the train runs in full supervision mode (fs-mode, see Fig. 4). We also assume that a balise sensor fails such that the balise message becomes inconsistent. The component balise-controller compensates this by adapting to the configuration balise-consistency. A configuration balise-consistency notifies the balise failure with a specific quality description, for example, the limitation of speed. This quality allows a component movement-controller to switch to the configuration LOA (Limit of Authority, the target speed is not zero). Then, the component brake-controller reconfigures to the configuration service brake (se-brake) and finally the component train-interface-controller commands brake (command-train) to limit the actual speed.

This basic scenario demonstrates that the developer must also specify in the planner element of MAPE-K for the hierarchical component all the transient compositional configurations in order to reach a reasonable mode of operation for the composition. This becomes complex for a real-world system. In order to reduce this complexity, we define the dynamic priority assignment scheme, which redefines the default priority order for the configurations of all subcomponents and activates the configuration that gets a highest priority.

4.2. Evaluation strategy

Based on the default priority order assigned to each configuration of each subcomponent of a hierarchical component, a defined priority assignment scheme lists the configurations of each subcomponent composing the hierarchical configuration in decreasing priority order by redefining the default priority order as following:

- For each basic subcomponent, the active configuration that precondition is violated by a quality of input flow is disable (i.e., it gets the lowest priority), else it keeps it highest priority. In fact, assigning the highest possible priority to the active and non-violated configuration ensures the stability of hierarchical component. As a consequence, the configurations whose preconditions are subsumed by the precondition of the active configuration are disabled. All other configurations keep their relative default priority order.
- For each hierarchical subcomponent, the emergent hierarchical configuration is enabled and all other hierarchical configurations are disabled because the configuration of a hierarchical component already determines the compositional reconfiguration behaviour. Indeed, as previously mentioned in section 3.1, the hierarchical components do not contain an explicit behaviour specification themselves but they contain the reconfiguration behaviour of their subcomponents.

To illustrate this, we consider the compositional reconfiguration behaviour at time step t4 of our scenario (see Fig. 4). The configurations of each subcomponent are listed in decreasing priority order (the left part of Fig. 5). The configuration that has the highest priority is activated. For example, at time step t4, the configuration LOA of subcomponent movement-controller is activated. This configuration sets a quality type q-mvt, shown in Fig. 3a. In relation with ERTMS/ETCS, this quality can be announced by the variable T_LOA, which announces that the valid time for the target speed at the LOA is overpassed. The quality type q-mvt is in contradiction with the precondition of configuration se-brake, i.e., q-trans.q-fs /\ q-balise /\ q-radio (see Fig. 3a). Therefore, for the subcomponent brake-controller, the dynamic assignment scheme disables the configuration se-brake, sets the highest priority to em-brake and keeps the default priority order for other configurations. Hence, at the next time step t5 (the right part of Fig. 5), the subcomponent brake-controller activates the configuration em-brake and notifies its provided quality q-brake.q-em to its environment. The hierarchical component train-controller exposes the new emergent compositional reconfiguration sequence as its compositional configuration. Thus, if this hierarchical component is defined at the top-level composition, only this compositional configuration is used and so on. To resume, the reconfiguration behaviour of a component-based architecture is realised in a bottom-up fashion, until the reconfiguration behaviour of the top-level component is fully specified.

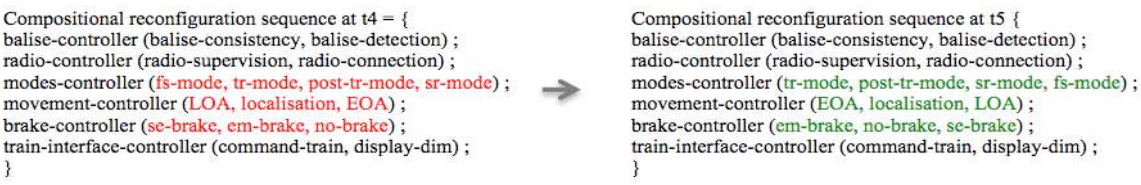

Fig. 5. Compositional reconfiguration sequence

Table 1. A comparative evaluation

Requirements for the evaluation of a component-based software engineering process in safety-critical system (Grunske et al., 2005)	SaveCCM (Hansson et al., 2004)	Pin (Hissam et al., 2005)	AUTOSAR (Fürst et al., 2009)	Mechatronic UML (Heinzemann et al., 2013)	SARA presented in this paper
Requirement 1 (Appropriate Component-Level Models): clear specification of component models	+	++	-	+	+
Requirement 2 (Encapsulation and Interfaces): separation of different types of interfaces	++	++	++	++	++
Requirement 3 (Dependencies on External Components): specification of failure propagation	+	+	+	+	++
Requirement 4 (Integration of Analysis Results): integration of safety analysis model	+	++	+	+	+
Requirement 5 (Practicable Granularity): practical granularity for flexibility and analysis	+	+	++	+	++
Requirement 6 (Tool Support): from manual application to tool application	++	+	+	++	--
Requirement 7 (Self-reconfiguration support) Analysis of dynamic reconfiguration behaviour	-	-	+	+	+

5. Related work and comparative evaluation

Due to the nature of the safety-critical embedded systems, the safety property on system level is influenced by different quality properties of (software and hardware) components. It is clear that failures at the software component level influence the safety property on system level. Consequently, Grunske et al. (2005) have investigated the applicability of component-based software engineering in the domain of safety-critical systems. As a result, they have identified a set of 6 requirements that are needed to evaluate safety properties for a system built with components. Based on the 6 identified requirements and on the one additional requirement regarding analysis of dynamic reconfiguration behaviours, our approach and some related safety-critical software component-based approaches are examined and compared. A brief overview of the 7 requirements has been mapped and given in first column of Table 1. Based on our knowledge of four component-based approaches, presented in first line of this table, their comparative evaluation is presented in the Table 1. We assign a quality mark, ranging from -- (requirements are not completely fulfilled) to ++ (requirements are completely fulfilled).

The comparative evaluation shows that each of the component-based approaches has its own strengths and limitations. For example, considering requirement 1, as the name AUTOSAR (AUTomotive Open System Architecture) indicates, the main focus of AUTOSAR is architecture level. It does not mainly focus at component level that is partial limitation (-) for requirement 1. However, our approach (SARA), SaveCMM and MechatronicUML approaches focus mainly at high level modelling. This is a partial strength (+) for requirement 1 since their purposes are specific in particular safety-critical domain (e.g., SaveCCM for vehicular and SARA for railway domain) compared to the Pin component model, whose purpose is general in safety-critical domain. Another limitation of our approach is that the tool support is not yet available, but it provides a constructive development methodology. In addition, our approach has been developed to support the adaptive applications i.e., the application which can switch among the predefined configurations at some well-defined points.

6. Conclusion

In addition to hardware redundancy, largely used in safety-critical systems to achieve reliability and availability, the reconfiguration behaviour of software components is also a flexible means to increase system availability, without additional costs, by changing, in case of failures, the active configuration in the class of predefined configurations. To this end, we have proposed a software component-based approach, which exploits the implicit redundancy of software control functions defined in system requirement specification in order to define the class of predefined configurations of the components. The annotation of the predefined configurations provides an essential support to abstract the runtime reconfiguration model in order to analyse the dynamic reconfiguration behaviour. As most of the specific safety component-based approaches, our component-based approach has its own strengths and limitations. To increase the strengths and to reduce its limitations, particularly tool support limitation, our next step is to go from manual application to tool application. Thus, we hope that this would

promote the use of the autonomous components computing approach across the railway safety-critical domain in order to save the costs of development, ensure the safety of the system, and achieve the acceptable availability.

References

Bondavalli, A., & Simoncini, L. (1990). Failure classification with respect to detection. ESPRIT BRA Project 3092: Predictably Dependable Computer Systems. Report, Task B: Specification and Design for Dependability.

Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., & Stefani, J.-B. (2006). The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.,* vol. 36, no. 11-12, pp. 1257–1284.

Clifton, A. E. (2005). Hazard Analysis Techniques for System Safety. John Wiley and Sons Inc., New Jersey.

Crnkovic, I., Sentilles, S., Vulgarakis, A., & Chaudron, M. (2011). A classification framework for software component models. *Software Engineering,* IEEE Transactions on, vol. 37, no. 5, pp. 593-615.

EN 50128 (2011): Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems. European Committee for Electrotechnical Standardization (CENELEC).

ERTMS/ETCS (2006). European Rail Traffic Management System/European Train Control System release notes to system requirements specification (subset 026) version 2.3.0. http://www.era.europa.eu/Document-Register/Pages/UNISIGSUBSET-026.aspx.

Fürst, S., et al. (2009). AUTOSAR - A Worldwide Standard is on the road, *14th International VDI Congress,* Electronics Systems for Vehicles, Baden-Baden.

Grunske, L., Kaiser, B., & Reussner, R. H. (2005). Specification and evaluation of safety properties in a component-based software engineering process. In Atkinson, C., Bunse, C., Gross, H.-G., and Peper, C., editors, *Component-Based Software Development for Embedded Systems*, pp. 249-274, Berlin. Springer-Verlag.

Hansson, H., Akerholm, M., Crnkovic, I., & Torngren, M. (2004). Saveccm – a component model for safety-critical real-time systems. In *Proceedings of the 30th EUROMICRO Conference*, pp. 627–635, Washington, DC.

Heinzemann, C., & Becker, S. (2013). Executing reconfigurations in hierarchical component architectures. In *Proceedings of the 16th International ACM Sigsoft symposium on CBSE '13*, pp. 3-12, New York, NY, USA.

Hissam, S., Ivers, J., Plakosh, D., & Wallnau, K. (2005). Pin component technology (v1.0) and its c interface (cmu/sei-2005-tn-00). *Technical report*, Carnegie Mellon Software Engineering Institute (SEI).

IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, International Electrotechnical Commission Std.

Jantsch, A. (2004). Modeling Embedded Systems and SoCs. Morgan Kaufmann, San Francisco, CA.

Kaiser, B., Liggesmeyer, P., & Mäckel, O. (2003). A new component concept for fault trees. In *Proceedings of the 8th Australian workshop on Safety critical systems and software* - SCS'03, vol. 33, pp. 37- 46, Australia.

Kantz, H., & Koza, C. (1995). The elektra railway signalling system: Field experience with an actively replicated system with diversity. In *25th International Symposium on Fault-Tolerant Computing*, pp. 453-458.

Kephart, J. O. & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, vol. 36 (1), pp. 41-50.

McKinley, P. K., Sadjadi, S. M., Kasten, E. P., & Cheng, B. H. C. (2004). Composing adaptive software. *IEEE Computer Society Press*, vol. 37 no. 7, pp.56-64. Los Alamitos.

Szyperski, C., Gruntz D., & Murer S. (2002). Component Software: Beyond Object-Oriented Programming. ACM Press.