Towards a Secure RA2DL Based Approach

Farid Adaili[1,2,3](✉), Olfa Mosbahi[1], Mohamed Khalgui[1,4], and Samia Bouzefrane[3]

[1] LISI Laboratory, INSAT Institute, University of Carthage, Tunis, Tunisia
olfamosbahi@gmail.com, khalgui.mohamed@gmail.com
[2] Tunisia Polytechnic School, University of Carthage, Tunis, Tunisia
[3] CEDRIC Laboratory, National Conservatory of Arts and Crafts, Paris, France
{farid.adaili,samia.bouzefrane}@cnam.fr
[4] Systems Control Laboratory, Xidian University, Xian, China

Abstract. This chapter deals with secured reconfigurable AADL based-control component of embedded system (to be named by RA2DL) that should be adapted their behaviours to environment execution according to user requirements. For various reasons, we propose a new method denoted by $RA2DL - Pool$ for guarantee and control the security of RA2DL component. $RA2DL - Pool$ is a container of sets of RA2DL components characterized by similar properties. Also, it holds well-defined methods for grouping RA2DL components together. To consolidate $RA2DL - Pool$ technology, we will put a set of security-mechanisms divided into two families: (i) Authentication Mechanism where all users must authenticate to access to the reserved services of $RA2DL - Pool$ or RA2DL components and (ii) Access Control Mechanism to control the access to the RA2DL components. We model and verify this solution and develop a tool for its simulation by taking a real-case study dealing with the Body-Monitoring System (BMS) as a running example.

Keywords: Pooling · Component-based approach · Dynamic reconfiguration · Security · Authentication · Access control · RA2DL · Implementation · Modelling · Evaluation

1 Introduction

Nowadays in the academy and manufacturing industry, many research works have been made to deal with real-time reconfiguration of embedded control systems. The new generation of these systems are addressing today a new criteria such as flexibility and agility. To reduce their cost, these systems have to be changed and adapted to their environment without any disturbance. We are interested in this chapter in the reconfigurable AADL technology. AADL component is a software unit to be encoded with a set of algorithms that implement its control functions. Each algorithm is activated by corresponding external event-data inputs, and generally produces the results of its execution on corresponding data-event outputs.

The usability of the embedded and reconfiguration technologies in the information systems is not only a concern of major corporations and governments but also an interest of individual users. Due to this wide use, many of these systems manage and store information that is considered sensitive, such as personal or business data. The need to have secured components for each system that contains such information becomes a necessity rather than an option [16]. The embedded components [17] are getting increasingly connected and are more and more involved in networked communications. The users of these components are now able to execute almost all the network/internet applications. These components are also increasingly involved in the transfer of secured data through public networks that need protection from unauthorized access. Thus the security requirements in embedded systems have become critical.

Traditional security research has been focusing on how to provide assurance on confidentiality, integrity, and availability [8]. However, with the exception of mobile code protection mechanisms, the focus of past research is not how to develop secured software that is made of components from different sources. Previous research provides necessary infrastructures, but a higher level perspective on how to use them to describe and enforce security, especially for component-based systems, has not received sufficient attention from research communities so far.

We define in a previous paper [10] a new concept of components named RA2DL as a solution for reconfigurable AADL components composed of controller and controlled modules. The first one is a set of reconfiguration functions applied in RA2DL to adapt its execution to any evolution in the environment, described by three reconfiguration forms:

(i) Form 1: Architectural level: modifies the component architecture when particular conditions are met. This is made by adding new algorithms, events and data or removing existing operations in the internal behaviors of the component.

(ii) Form 2: Compositional level: modifies the composition of the internal components (algorithms) for a given architecture.

(iii) Form 3: Data level: changes the values of variables without changing the component algorithms, and the second one is a set of input/output events, algorithms, and data as represented by reconfiguration modules.

However, securing an RA2DL component is not an easy task. With rapidly advancing hardware/software technologies and ubiquitous use of computerized applications [19], modern software is facing challenges that it has not seen before. More and more software is built from existing components which come from different sources. This complicates analysis and composition, even if a dominant decomposition mechanism is available. Additionally more and more software/hardware components are running in a networked environment. These network connections open possibilities for malicious attacks that were not possible in the past. These situations raise new challenges on how to handle security so that to design a component-based architecture that is more resistant to attacks and less vulnerable.

Facing the new challenges for security of reconfigurable RA2DL-based systems, we propose new solutions allowing the required authentification for the access control to components under a set of constraints such as the limitation in memory. These solutions are supported by a new concept called pool which is a container that gathers networked RA2DL under security constraints. The container allows the control of any operation allowing the reconfiguration of RA2DL components as well as the access to local algorithms and data.

The chapter's contribution is applied to a case study of an Body-Monitoring System (BMS) that will be followed as a running example. A tool is developed in a collaboration between LISI Lab at University of Carthage in Tunisia and CEDRIC Lab at CNAM in France to implement and simulate the security in the case study.

The current chapter is organized as follows: We discuss in Sect. 2 the originality of the chapter by studying the state of the art. Section 3 describes the background of RA2DL. Section 4 defines the new extension for secured RA2DL components. We expose in Sect. 5 the case study: Body-Monitoring System (BMS) and how the implementation is performed to secure RA2DL. Section 6 concludes the chapter and gives some perspectives as a future work.

2 State of the Art of Secured Component-Based Design Approaches

In this section, we present a state of the art of secured component-based design approaches. In [6], the authors present a classification of component-based systems by describing software components as independent units that interact to form a functional system. A component does not need/have to be compiled before it is used. Each component offers services to the rest of the system and adopts a provided interface that specifies the services that other components can use.

The authors in [19] present a treatment of an important security aspect, access control, at the architecture level and modeling of security subject, resource, privilege, safeguard, and policy of architectural constituents. The modeling language, Secure xADL, is based on the existing modular and extensible architecture description language.

In [7], the authors propose a QA (Quality Assurance) model for component-based software which covers component requirement analysis, component development, component certification, component customization, and system architecture design, integration, testing and maintenance. An extension of the Component Object Model (COM), Distributed COM (DCOM), is a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. DCOM is designed for use across multiple network transports, including internet protocols such as HTTP. When a client and its component reside on different machines, DCOM simply replaces the local interprocess communication with a network protocol. Neither the client nor the component is aware of the changes of the physical connections.

In [9], Rugina et al. present an iterative dependency-driven approach for dependability modeling using AADL. This approach is a part of a complete framework that allows the generation of dependability analysis and evaluation models from AADL models to support the analysis of software and system architectures in critical application domains.

AADL and OSATE tools can be used to validate the security of systems designed using MILS4 architecture [11]. The work in [13] uses two mechanisms to modularize or divide and conquer in secure systems: partitions, and separation into layers. The MILS architecture isolates processes in partitions that define a collection of data objects, code, and system resources and can be evaluated separately. Each partition is divided into the following three layers: Separation Kernel Layer, Middleware Service Layer and Application Layer each of which is responsible for its own security domain and nothing else.

In [14], the author presents the extension UMLsec of UML that allows to express security relevant information within the diagrams in a system specification. UMLsec is defined as an UML profile using the standard UML extension mechanisms. In particular, the associated constraints give criteria to evaluate the security aspects of a system design by referring to a formal semantic of a simplified fragment of UML.

In [4], Bernstein define a Docker (www.docker.com) is an open source project providing a systematic way to automate the faster deployment of Linux applications inside portable containers. Basically, Docker extends LXC with a kernel- and application-level API that together run processes in isolation: CPU, memory, I/O, network, and so on. Docker also uses namespaces to completely isolate an applications view of the underlying operating environment, including process trees, network, user IDs, and file systems.

Docker containers are created using base images. A Docker image can include just the OS fundamentals, or it can consist of a sophisticated prebuilt application stack ready for launch. When building images with Docker, each action taken (that is, command executed, such as apt-get install) forms a new layer on top of the previous one. Commands can be executed manually or automatically using Dockerfiles.

Note that, no one in all related works deals with secured reconfigurable components. We propose in this chapter a new concept of security of RA2DL components to be named $RA2DL - Pool$ that allows: (i) Grouping of RA2DL components that have the same similar properties. (ii) Associating to each $RA2DLPool$ a security mechanism like authentication and access control mechanisms.

3 RA2DL Background

We defined in a previous paper [10] the concept of RA2DL components as an extension of reconfigurable AADL [21] (Architecture Analysis and Design Language). RA2DL as depicted in Fig. 1 is composed of controller and controlled modules where the first one is a set of reconfiguration functions applied in AADL, and the second one is a set of input/output events, algorithms, and data. The controlled module is described by the following four modules:

IEM (Input Events Module): This module processes the reconfiguration of input events (IE) stored in the $IEDB$ database of input events. It defines and activates at a particular time a subset of events to execute the corresponding algorithms in RA2DL.

OEM (Output Events Module): This module processes the reconfiguration of output events (OE) stored in the $OEDB$ database of output events. It defines and activates at a particular time a subset of events to be sent once the corresponding algorithms finish their execution in RA2DL.

ALM (Algorithms Module): This module processes the reconfiguration of the active algorithms (addition or removal) at a particular time in order to be coherent with active input and output events of IEM and OEM. These algorithms are stored in the $ALDB$ database of algorithms.

DM (Data Module): This module processes the reconfigurations of *data* in RA2DL in coherence with the rest of modules. It is stored in the DDB database of data values.

We focus on three hierarchical reconfiguration levels in RA2DL:

(i) Form 1: Architectural level: Deals with the changes of the architecture of the RA2DL component when particular conditions are satisfied. In this case, it is possible to add, remove or also change the internal behavior of the component in IEM, OEM, ALM and DM. We denote by Ψ_{Cmp} the big set in $ALDB$ of all the possible algorithms involved in the different implementations of the component *Cmp*, which is implemented at any particular time t by a subset ξ_{Cmp} that represents the set of algorithms involved in a particular implementation $\xi_{Cmp} \subseteq \Psi_{Cmp}$. We model the architectural level AL by a finite state machine $\mathbf{S}_{AL}$ such that each state of $\mathbf{S}_{AL}$ corresponds to a particular implementation of IEM, OEM, ALM and DM.

$\mathbf{S}_{AL}$= (Ψ_{Cmp}, $\mathbf{O}$, δ), where:

$\mathbf{O}$ is a set of n states in $\mathbf{S}_{AL}(\mathbf{O} = \{\mathbf{S}^i_{AL} \mid i \in 1..n\})$,
δ is a state-transition function Ψ_{Cmp} x $\mathbf{O} \rightarrow \Psi_{Cmp}$ x$\mathbf{O}$.

(ii) Form 2: Compositional level: This level keeps the same architecture in *Cmp* but just changes the composition of algorithms, input-output events in order to adapt the component to its environment. It is formalized by different Composition State Machines CSM, such that each one CSM corresponds to a particular state in the Architecture Level S_{AL}. For each state S^i_{AL} in S_{AL}, we define in the second hierarchical level (Composition Level CL) a particular state machine to be denoted by S^i_{CL}. Each state in $S^{i,j}_{CL}$ in S^i_{CL} defines a particular composition of the subset of algorithms and input-output events. This composition affects a priority to each algorithm in order to get a deterministic execution model of the AADL component *Cmp*. We denote by $\Gamma(\delta_{Cmp})$ the set of all possible execution models of algorithms of δ_{Cmp} at the composition Level.

$\mathbf{S}_{CL}$= ($\Gamma(\delta_{Cmp})$, $\mathbf{P}$, γ), where:

$\mathbf{P}$ is a set of m composition states in $\mathbf{S}_{CL}(\mathbf{P}= \{ \mathbf{S}^i_{CL} \mid i \in 1..m\})$,
γ is a state-transition function $\Gamma(\delta_{Cmp})$ x $\mathbf{P}\rightarrow \Gamma(\delta_{Cmp})$ x$\mathbf{P}$.

(iii) Form 3: Data level: A reconfiguration scenario $R^{i,j}_{CL}$ at Composition Level CL, is a transition from a state S^i_{CL} to another state S^i_{CL} of S_{CL}. The reconfiguration of the AADL component Cmp at the third hierarchical level DL corresponds to the update of *data*. We define for each state S^i_{AL} of S_{AL} and for each state S^j_{CL} of S_{CL} a new state machine S_{DL} where each state corresponds to new values to be affected to *data* belonging to μ_{Cmp} under the composition S^i_{CL}. Let $\Gamma(\mu_{Cmp})$ be the set of all possible values of data under the composition S^j_{CL}.

This level deals with the light reconfiguration of data of the RA2DL component. It is formalized by a set of Data State Machines where each state of them corresponds to particular values of data. We define for each state $\mathbf{S}^i_{AL}$ of $\mathbf{S}_{AL}$ and for each state $\mathbf{S}^{i,j}_{CL}$ of $\mathbf{S}^i_{CL}$ a new state machine $\mathbf{S}^{i,j,k}_{DL}$ where each state corresponds to new values of data.

$\mathbf{S}_{DL}$= ($\Gamma(\mu_{Cmp})$, $\mathbf{Q}$, ϑ), where:

$\mathbf{Q}$ is a set of k composition states in $\mathbf{S}_{DL}(\mathbf{Q}=\{ \mathbf{S}^i_{DL} \mid i \in 1..k\})$,
ϑ is a state-transition function $\Gamma(\mu_{Cmp})$ x $\mathbf{Q} \rightarrow \Gamma(\mu_{Cmp})$ x$\mathbf{Q}$.

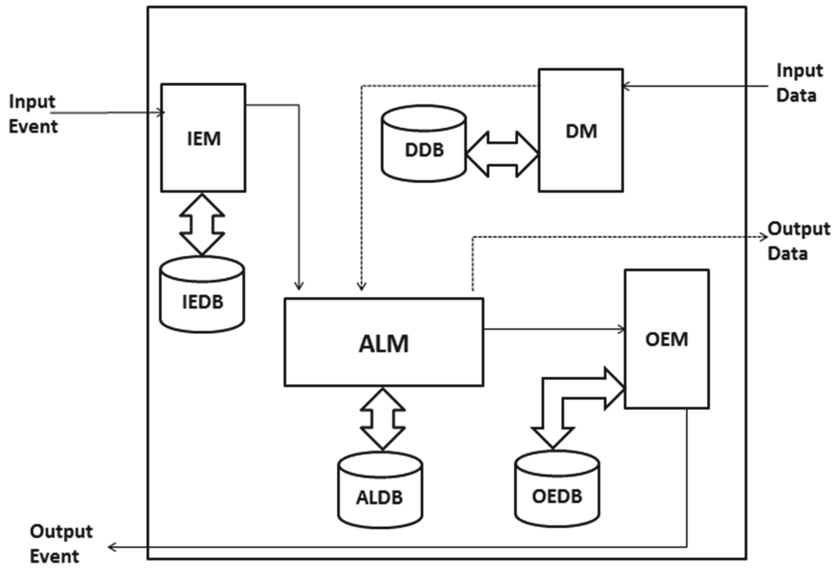


Fig. 1. Architecture of an RA2DL component.

In another extension in [1] for enhancing the execution of RA2DL components, a new execution model is proposed which is composed of three layers: **(i) Middleware Reconfiguration level** that handles the input reconfiguration flows, **(ii) Execution Controller level** to control the execution and reconfiguration of RA2DL and **(iii) Middleware Synchronization level** that controls and manages the synchronization of the reconfiguration. Additionally, we proposed a new approach to coordinate several RA2DL components in a distributed architecture based on a coordination matrix.

Because of the resource limitations in adaptive systems, satisfying a non-functional requirement such as security requires careful balance and trade-off with other properties and requirements of the system such as performance, memory usage and access rights of the RA2DL. This further emphasizes the fact that security cannot be considered as a feature that is added later to the design of an RA2DL component. It needs to be considered from early stages of development and along with other requirements. In fact, the security by design approach as defined by Ray and Cleaveland [18] in software engineering ensures that security is addressed at the point of conception to avoid the security vulnerabilities. Considering the characteristics of RA2DL components, major impacts of security features in these systems are based on performance, power consumption, flexibility, maintainability and cost [15]. Therefore in the design of RA2DL components, implications of introducing security decisions should be taken into account and analyzed. Several related works do not provide solutions to develop security of RA2DL components of adaptive embedded systems. The current chapter proposes new extended solutions to secure an RA2DL component. However, in this work we want to extend this study by considering a new architecture of secured RA2DL-based pools.

4 New Extension for Secured RA2DL

In this section, we enrich RA2DL by security mechanisms that undergo such a failure to enhance their execution and simulation.

4.1 Motivation: RA2DL-Pool

Security is an aspect that is often neglected in the design of adaptive systems. However, the use of these systems for critical applications such as controlling power plants, vehicular systems control, and medical devices [20] makes security considerations even more important. Also because of the operational environment of adaptive systems and the reconfiguration actions applied by an RA2DL component. To allow the required security, we introduce the concept of $RA2DL-Pool$ as a container which is an abstract class that offers different services dealing with security, where each $RA2DL-Pool$ has a level of sensitivity of the information of its RA2DL components. $RA2DL-Pool$ container serves as a general purpose holder of other components. It holds well-defined methods for grouping RA2DL components together. *RA2DL – Pool* is represented by the following elements:

- **Controller:** it is the crucial part of the pool that contains methods and represents firstly the interface between the user and the pool, and secondly between the pool and the RA2DL components,
- **Tables:** there are three kinds of tables: use table (UT), reconfiguration table (RT) and security table (ST),
- **Database:** is the database containing the sets of RA2DL components,
- **Reconfiguration Scenarios:** define the set of reconfiguration scenarios realized in pool or in its RA2DL components. Each scenario will be applied in relation with the three tables (UT, RT and ST),
- **RA2DL:** it is the RA2DL component with its algorithms and input/output ports.

Figure 2 reproduced from [2] presents the class diagram of *RA2DL – Pool*. An *RA2DL – Pool* container holds a set of RA2DL components with a set of methods. This set of components has a set of methods that describe how to examine and add or delete components to the *RA2DL – Pool*. It contains the following methods described in Table 1 presented in [2].

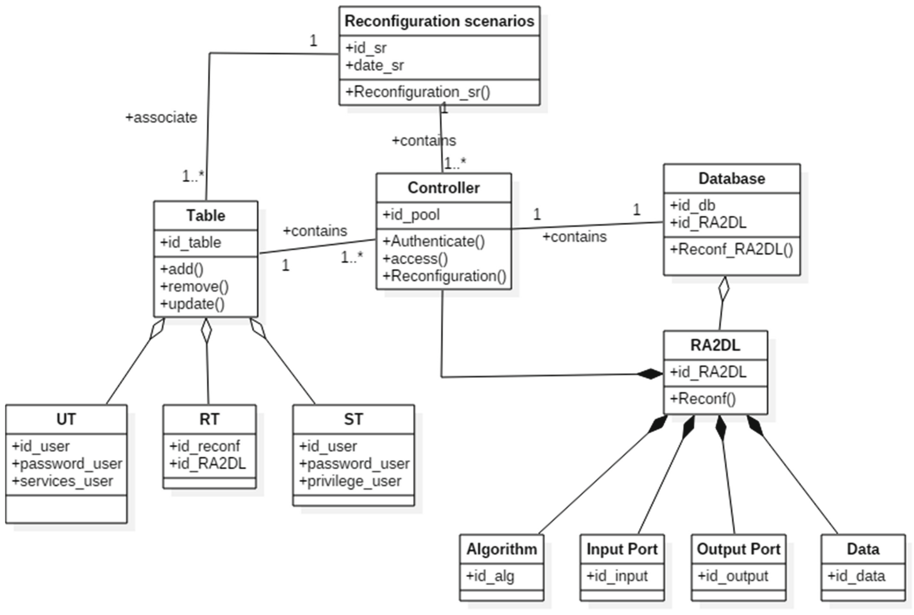


Fig. 2. Class diagram of RA2DL-Pool.

Table 1. RA2DL-pool methods.

Method	Description
getRA2DL ()	Number of components within the $RA2DL - Pool$
Component-getRA2DL(int position)	Component at the specific position
Component[] getRA2DL ()	Array of all the RA2DL components held within the container
RA2DL-add (Component RA2DL, int position)	Adds RA2DL component to $RA2DL - Pool$ at position
add (Component RA2DL, RA2DL constraints)	Layouts that require additional information
public void remove (int index)	Deletes the RA2DL at position index from the $RA2DL - Pool$
remove (RA2DL component)	Deletes the RA2DL from the $RA2DL - Pool$
removeAll ()	Removes all RA2DL from the $RA2DL - Pool$
boolean isAncestorOf (RA2DL)	Checks if the RA2DL is a parent of container
addContainerListener (pool)	Registers listener as a controller of RA2DL-Pool
removeContainerListener (pool)	Removes listener as an interested listener of RA2DL-Pool
processEvent (RA2DLEvent e)	Receives RA2DL events with $RA2DL - Pool$ as its target
addNotify ()	Creates the peer of all the components within it
removeNotify ()	Destroys the peer of RA2DL contained within it
Insetsgetinsets()	Gets the containers current insets
list()	Useful method to find out what is inside a container

4.2 Security Mechanisms for RA2DL

To consolidate the $RA2DL - Pool$ technology, we will put a set of security-mechanisms divided into two families are described in Fig. 3 reproduced from [2]:

Authentication Mechanism. This is a critical mechanism where all users must authenticate to access to the reserved services of $RA2DL-Pool$ or RA2DL components. This mechanism is always in relation with the user table (UT), where the columns u are the identifiers of users (id_user) and lines s represent the services ($services\_user$). To implement the authentication mechanism, we use **RADIUS** *(Remote Authentication Dial-In User Service)* is a client/server protocol that runs in the application layer developed by Livingston Enterprise [22], which is a networking protocol that provides centralized Authentication, Authorization, and Accounting (AAA) management for users who connect and use a network service. The principle of the authentication of an RA2DL with RADIUS is as follows:

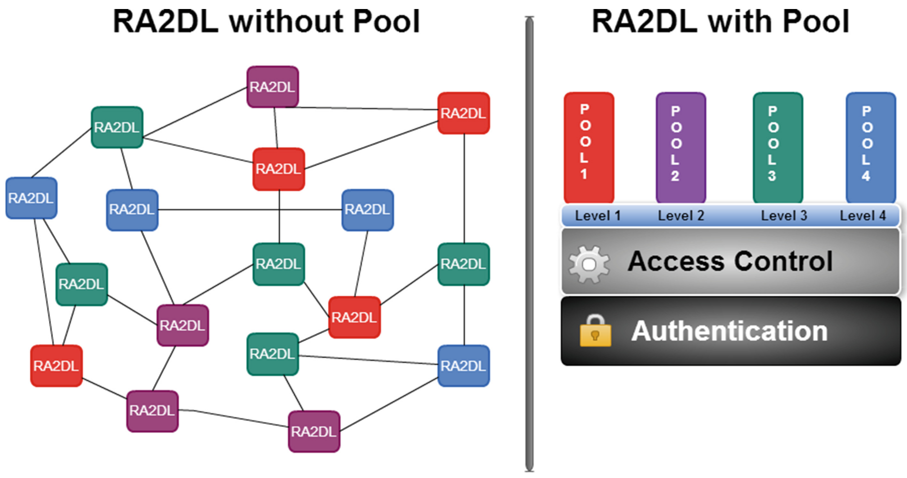


Fig. 3. Secured RA2DL method.

1. the *Controller* executes a connection request. *UT* table recovers the identification information,
2. the *Controller* transmits this information to the target service in RA2DL,
3. the target component receives the connection request from the *Controller*, controls and returns the configuration information required for the user to provide or deny access,
4. *Controller* refers to the user an error message if it fails an authentication.

Access Control Mechanism. This mechanism comes just after authentication to control the access to the RA2DL components. Two tables are used in this case: security and reconfiguration tables. The first one is the security table *ST* which contains in lines (p) all the user privileges ($privilege\_user$) and in columns (u) the (id_user). The second one is the reconfiguration table (RT) that contains in lines (r) reconfigurations identifiers (id_reconf) and in columns (c) the identifiers of RA2DL components (id_RA2DL).

This mechanism may be represented by a triplet (S, C, M_{sc}) where S denotes the service, C denotes the RA2DL component (or RA2DL-pool) and M_{sc} that maps each pair (C and S) to a set of access rights.

The matrix shown in Fig. 4 shows that the right of access r is associated with the service (Subject) S_j and C_j RA2DL component.

Figure 5 presents the sequencing of the interaction between the RA2DL components and the RA2DL-Pool. The main goal is to show this interaction and how to apply authentication and access control mechanisms.

Figure 6 highlights the activity of these two mechanisms and tests in order to achieve a secure RA2DL component.

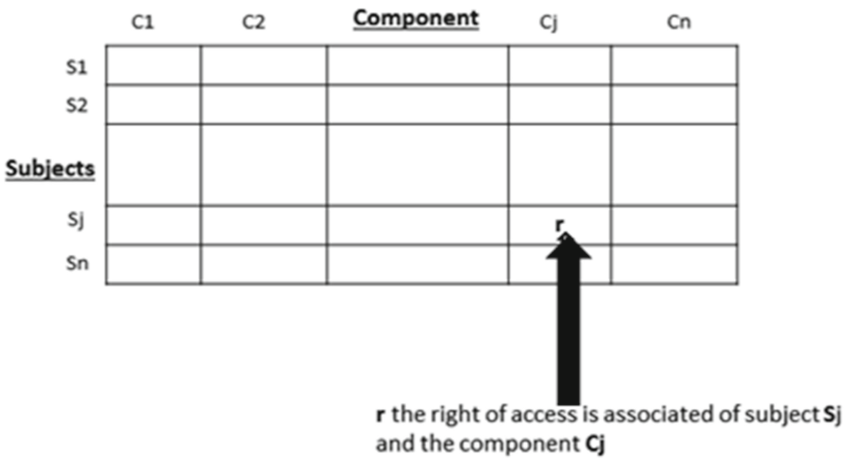

C1
C2
Component
Cj
Cn
S1
S2
Subjects
Sj
Sn
r
r the right of access is associated of subject Sj and the component Cj

Fig. 4. Access control matrix.

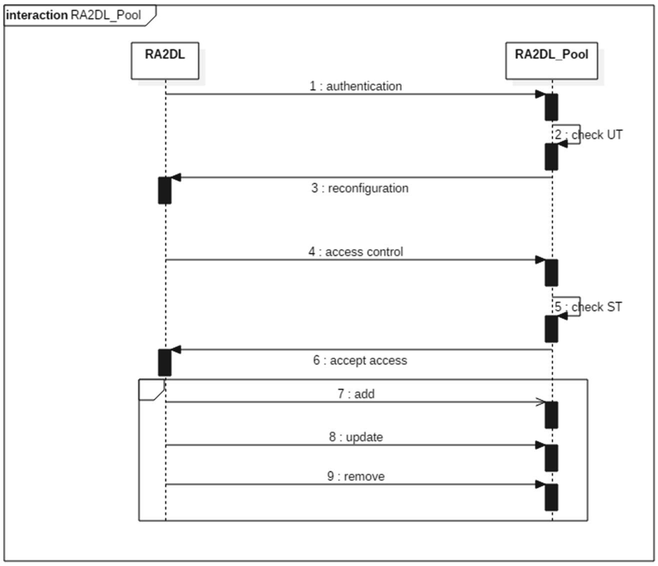


Fig. 5. Sequence diagram.

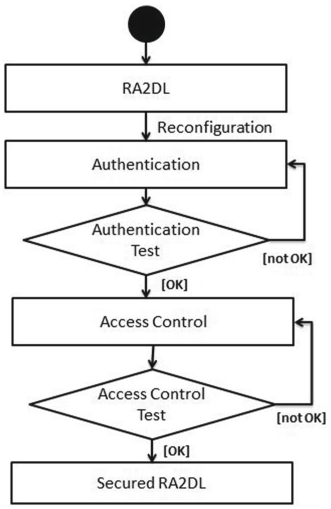


Fig. 6. Activity diagram.

4.3 Architecture of Secured RA2DL-Based Pools

We present in Fig. 7 the class diagram of the secured RA2DL-based pool. This diagram represents the architecture of RA2DL-based pools with the static aspect of the relation between the RA2DL components and the pool. It does not provide any information about its behavior. The architecture of secured RA2DL-based pools is composed of the following distinct classes: (i) *RA2DL*: The main class of the architecture, the component concerned by the security concept, (ii) $RA2DL - Pool$: It is the container of RA2DL components, (iii) *Security*: Is an association between RA2DL and RA2DL-Pool which represents the security-mechanisms, (iv) $RA2DL - Soft$: It is the software component of RA2DL, (v) $RA2DL - Hard$: It is the hardware component of RA2DL, (vi) *Algorithm*: Is a set of methods to be executed by each RA2DL component, (vii)$Reconfiguration$: Represents all of the reconfiguration scenarios to execute with RA2DL, (viii) *Architecture*: Describes the reconfiguration scenarios that touch on the RA2DL architecture, (ix) *Structure*: Describes the reconfiguration scenarios that touch on the RA2DL composition or structure, (x) *Data*: Describes the reconfiguration scenarios that touch on the RA2DL data, (xi) *EventPort*: Port for input/output event of RA2DL, (xii) *DataPort*: Port for input/output data of RA2DL.

4.4 Modelling and Verification

We propose in this section the modelling and verification of the new architecture of secured RA2DL-based pools by using UPPAAL [3]. Firstly, we model the pool with its security aspect. Secondly we check a set of properties to ensure the security of the pool.

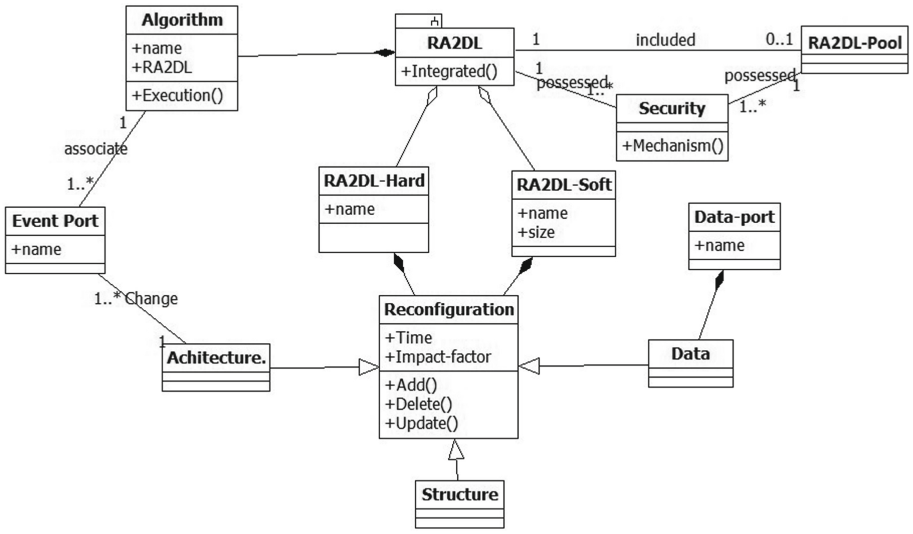


Fig. 7. Architecture of a secured RA2DL-based pool.

Modelling of Secured RA2DL-Based Pool. We propose in Fig. 8 Finite State Machine-based models of RA2DL-based Pool in order to show the interaction between the various states and to verify also some properties defined in user requirements. We present in the following a description of all the states and transitions characterizing this model. RA2DL-Pool is assumed to be a set of timed automaton, that run in parallel and communicate thanks to global variables.

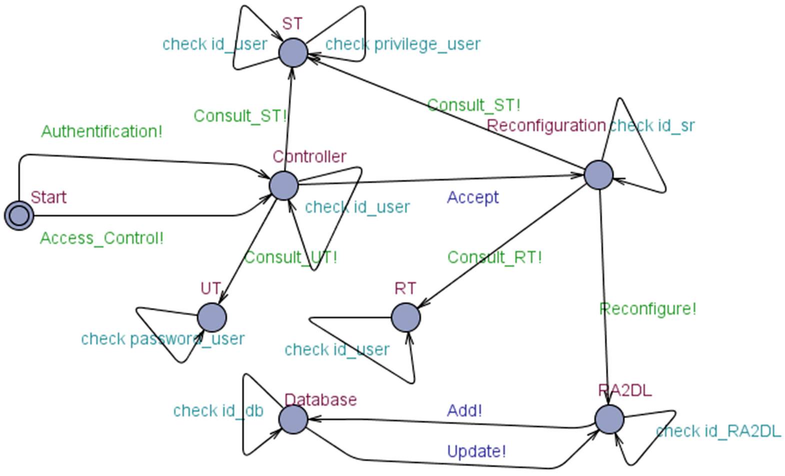


Fig. 8. Modelling of secured RA2DL-Based pools.

The states of this model are described as follows: *start* to start the querying or the connection of $RA2DL-pool$. $Controller$ represents the first contact with the pool, in this state the checking of *id_user* is important after verification of the *password_user* in the table UT. If the authentication is accepted and the password is checked, it can go to the state $Reconfiguration$ which represents all the reconfiguration scenarios. After the verification of the following parameters: (i) *id_sr* for the IDs of scenarios, *privilege_user* for the privilege of the user in the table ST and (iii) *id_reconf* for the IDs of the reconfiguration in the table RT. If all of the IDs are accepted, then the user may apply the reconfiguration in the target $RA2DL$ component after checking the *id_RA2DL*. A *database* is associated to this level to facilitate the reconfiguration of the RA2DL components.

Verification of Secured RA2DL-Based Pool. We propose in this section to check the relevance of the our solution and the contribution the following properties in order to verify the security of the RA2DL components.

- **Property 1:** *(Controller[].check id_user)AND (UT[].check passeword_user):* for each connection with the pool, we should check the user authentification by using the UT table,
- **Property 2:** *(Reconfiguration[].check id_sr) AND (RT[].check id_reconf):* before the execution of any reconfiguration scenario, it is important to check if it is registered in the reconfiguration table (RT),
- **Property 3:** *(Reconfiguration[].Reconfigure! ⇒ RA2DL[].check id_RA2DL) AND (ST[].check privilege_user):* this property concerns the verification of the access control mechanism,
- **Property 4:** *RA2DL[].save ⇒ Database[].check id_db*: each RA2DL component should be imperatively saved in a *Database* to facilitate the use of RA2DL components and to minimize the execution time,
- **Property 5:** *(Controller[] AND Reconfiguration[] AND RA2DL[] AND Database[] AND ST[] AND RT[] AND UT[]) not deadlock:* the system is deadlock-free.

The verification of these properties is summarized in Table 2 already shown in [2].

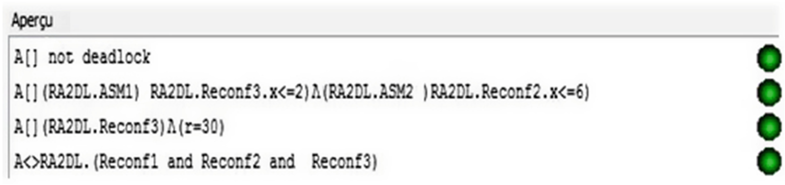
We show the validation of the all properties of our RA2DL component in Fig. 9.

5 Case Study and Implementation

We use as a running example in the current chapter the body-monitoring system (BMS) to evaluate the chapter's contribution.

Table 2. Verification results.

Property	Result	Calculation time (sec)	Consumed memory (Mo)
Property 1	True	10.52	5.72
Property 2	True	9.12	4.82
Property 3	True	5.32	3.20
Property 4	True	13.25	6.56
Property 5	True	8.23	4.37



```
A[] not deadlock                                                        ●
A[](RA2DL.ASM1) RA2DL.Reconf3.x<=2)Λ(RA2DL.ASM2 )RA2DL.Reconf2.x<=6)    ●
A[](RA2DL.Reconf3)Λ(r=30)                                               ●
A<>RA2DL.(Reconf1 and Reconf2 and  Reconf3)                             ●
```

Fig. 9. Validation properties.

5.1 Case Study: Body-Monitoring System (BMS)

During the last few years there has been a significant increase in the number and variety of wearable health monitoring devices ranging from simple pulse monitors, activity monitors, and portable Holter monitors, to sophisticated and expensive implantable sensors. The Body-Monitoring System (BMS) [12] is designed as a mobile device that is able to collect measured data and to act according to instructions set by a supervisor. The system consists of a body-monitoring network. In order to recognise the monitored person's state, the monitor unit connects to various body sensors and i/o devices by using either wired or wireless communication technologies. Data from all sensors are collected, stored and analysed at real-time and, according to the analysis, actions may then be performed. A computer is used as an interface to the body-monitoring network, and developed software allow a supervisor to configure the monitor unit for the monitored person, to connect sensors and i/o devices, define and upload instructions for monitoring and download collected data describe in Fig. 10 reproduced from [2].

The monitor unit software consists of a communication module (responsible for connecting and controlling sensors, and for gathering and pre-processing measured data), a storage module (for storage of collected data), and a policy interpretation module responsible of controlling the behaviour of the monitor unit according to instructions defined by a supervisor.

Two types of drivers are introduced. The role of a communication driver is to hide the way in which data is transmitted. There is one driver for every type of communication interface, e.g. a Bluetooth driver or an IEEE 802.11b driver. The communication driver does not care about the data itself; this is the role of device drivers. Each type of sensor has its own device driver. When a device

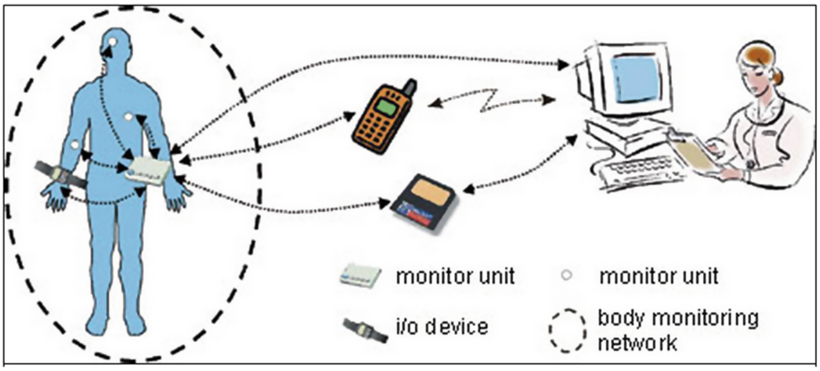


Fig. 10. Overview of the Body Monitoring System [5].

driver receives a message from one of its sensors it decodes the message and informs the policy engine about the state of the sensor. To send/receive a message to/from a sensor, the device driver uses the corresponding communication driver.

To secure this system, we must take into account these steps: (i) make the grouping of RA2DL components according to similar characteristics in RA2DL-Pool. (ii) assign for each RA2DL-pool a security level (depending on the degree of importance of the RA2DL components that they contain). (iii) allocate for each RA2DL-pool a security mechanism.

Running Example: We group the RA2DL components of BMS system in five RA2DL-Pools as shown in Fig. 11. (i) **RA2DL-Pool 1:** includes the following RA2DL components: *RA2DL-G* for the Glucose detection, *RA2DL-C* for the chloride detection and *RA2DL-W* for the water detection. (ii) **RA2DL-Pool 2:** includes the following RA2DL components: *RA2DL-L* for the lactate detection and *RA2DL-PH* for the PH detection. (iii) **RA2DL-Pool 3:** includes the following RA2DL components: *RA2DL-DM* for the Diabetes mellitus detection and the *RA2DL-BP* for the Blood pressure. (iv) **RA2DL-Pool 4:** contains the display device which is the component *RA2DL-Mobil*. (v) **RA2DL-Pool 5:** contains the *RA2DL-Soft* for the transmission of data with a protocol until *RA2DL-Mobil*.

5.2 Implementation

We present in this section the tool of the BMS system that we developed in LISI Laboratory at INSAT Institute of University of Carthage in Tunisia and CEDRIC Laboratory at National Conservatory of Arts and Crafts of Paris in France. Figure 12 reproduced from [10] shows the tool offers the possibility to create all reconfiguration scenarios of the RA2DL component (addition, removal and update of algorithms, events and data) when any problem occurs.

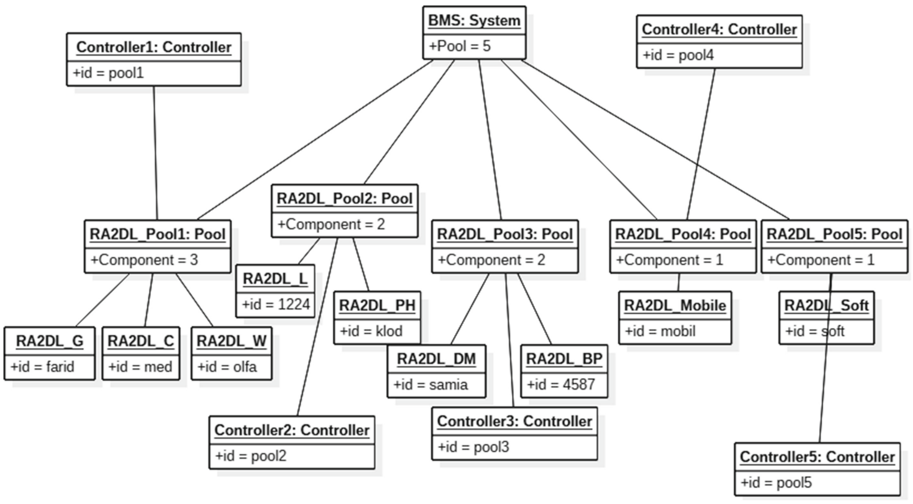


Fig. 11. Object diagram of BMS.

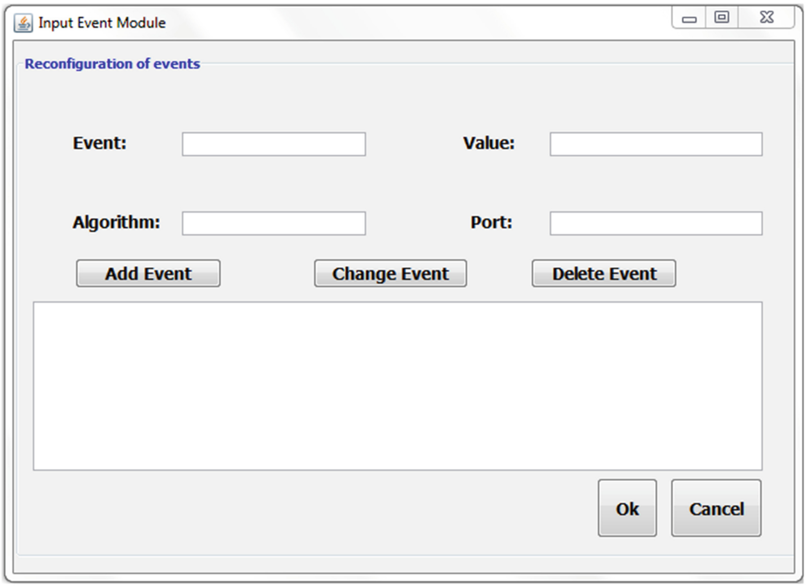
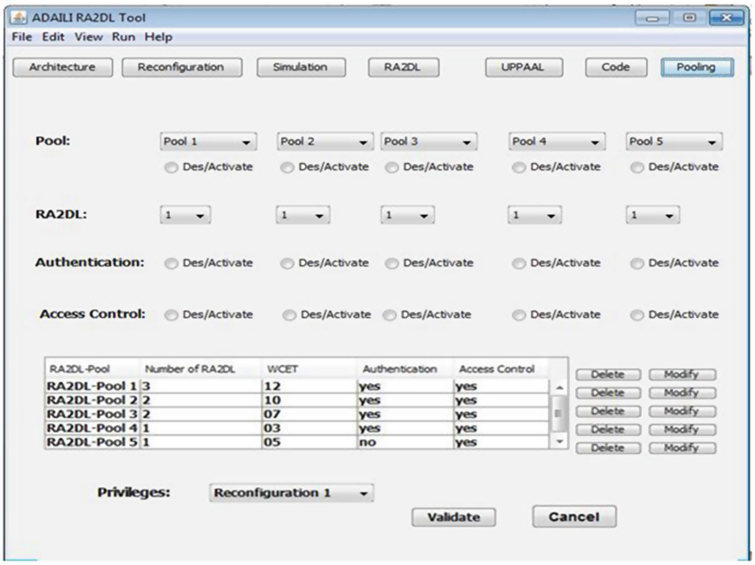


Fig. 12. Interface for reconfiguration architecture of RA2DL.

We assume five pools with their parameters such as the number of RA2DL components in pool, Worst Case Execution Time (WCET), the authentication and the access control mechanisms (Fig. 13).



Pool:	Pool 1	Pool 2	Pool 3	Pool 4	Pool 5
	Des/Activate	Des/Activate	Des/Activate	Des/Activate	Des/Activate
RA2DL:	1	1	1	1	1
Authentication:	Des/Activate	Des/Activate	Des/Activate	Des/Activate	Des/Activate
Access Control:	Des/Activate	Des/Activate	Des/Activate	Des/Activate	Des/Activate

RA2DL-Pool	Number of RA2DL	WCET	Authentication	Access Control
RA2DL-Pool 1	3	12	yes	yes
RA2DL-Pool 2	2	10	yes	yes
RA2DL-Pool 3	2	07	yes	yes
RA2DL-Pool 4	1	03	yes	yes
RA2DL-Pool 5	1	05	no	yes

Privileges: Reconfiguration 1

Fig. 13. RA2DL-Pools of BMS system.

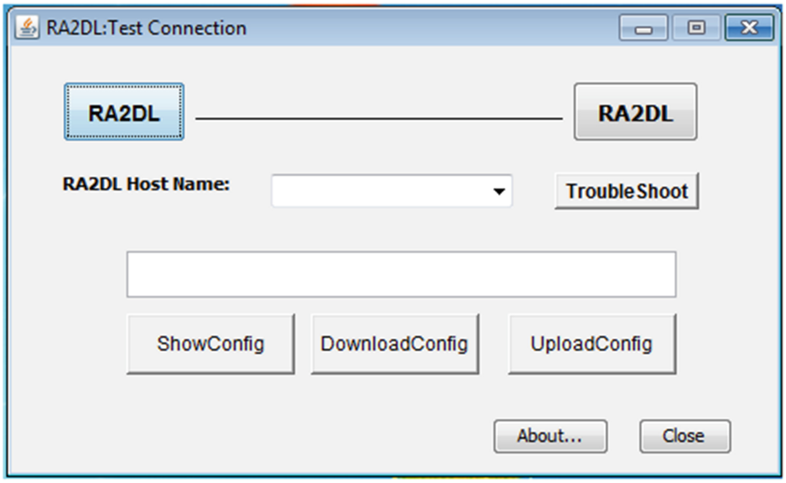


Fig. 14. Test of authentification mechanism.

Figure 14 reproduced from [2] shows the connectivity test of the different pools according to the authentication mechanisms and also to check the configuration between the various RA2DL components in each pool.

Running Example: The application of our approach to the BMS case study is illustrated in Table 3 reproduced from [2], where we give a security level (S.L)

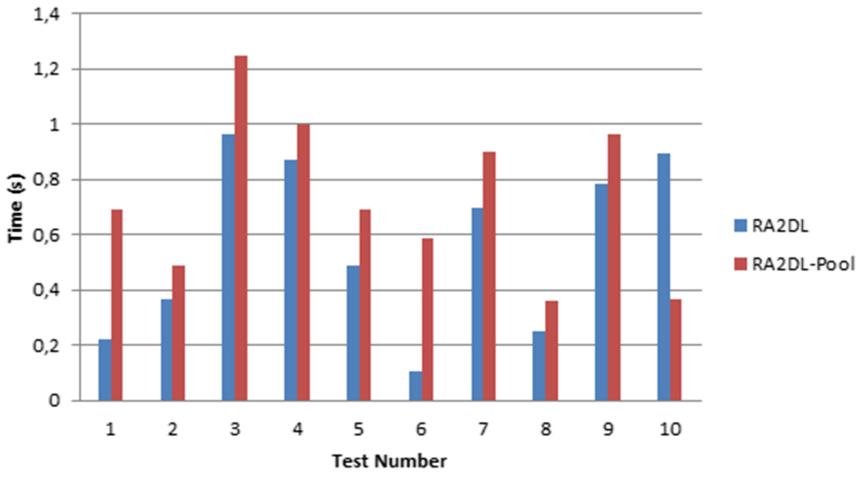


Fig. 15. Result of evaluation.

Table 3. Running example.

	Security level	Authentication mechanism	Access control mechanism	Security
Pool 1	1	No	Yes	Yes
Pool 2	2	Yes	No	Yes
Pool 3	6	Yes	Yes	Yes
Pool 4	5	Yes	Yes	Yes
Pool 5	5	Yes	Yes	Yes

for the five pools depending on the sensitivity of the comprising components. In the BMS system, the RA2DL-pool 3 is the most secured and RA2DL-pool 1 is the less secured one. Both security mechanisms are applied to the five pools.

5.3 Evaluation

This section is devoted mainly to test our approach and evaluate the execution time. Ten assessments are applied to the two mechanisms that are focused on two stolen: RA2DL without pool and RA2DL with pool of the BMS system. We show in Fig. 15 reproduced from [2] the results of the evaluation. We are interested in response time gains for secured and not secured RA2DL components.

The proposed approach has the following advantages:

(a) Functionality: RA2DL component in $RA2DL-Pool$ are at a functional level much more adaptable and extendable than traditional RA2DL components.

(b) Reusability: A reusability is an important characteristic of a high-quality RA2DL component. Programmers should design and implement RA2DL components in such a way that many different programs can reuse them.

Table 4. Comparison between Pool and Docker.

	Pool	Docker
Main goal	Secure RA2DL Component	Secure portable applications
Continent	RA2DL component	Applications
System	RA2DL-Based system	OS
Relationship between them	Yes	No
Security mechanism	Yes	No

(c) Maintainability: In BMS system a piece of functionality ideally is implemented just once. It is self-evident that this results in easier maintenance of system, which leads to lower cost, and a longer life.

We shows in Table 4 a comparative study between our approach *Pool* containers and *Docker* containers.

The RA2DL-Pool is a solution to secure in run-time each RA2DL component-based systems. By this solution the RA2DL component has become dynamic and secured. None of the existing works has treated the security of the RA2DL components as our method did.

6 Conclusion

Our work consisted, through this chapter, in proposing a novel approach for a required security in adaptive RA2DL control component based systems, to model and verify security control systems sharing adaptive resources. Whence, we chose to enhance RA2DL component to support security check. We proposed, then, a new and original solution to securing RA2DL component. Firstly, we define a new grouping methodology entitled RA2DL-Pool which has its own methods for the grouping of RA2DL components according to their similarities and security techniques. Secondly, we propose two crucial mechanisms to control the security in RA2DL-Pool: Authentication and access control mechanism. The relevance of our solution was proved thanks to model-checking using UPPAAL tool. This approach is original since RA2DL-Pool is a new formalism dedicated to secure RA2DL based control component.

The next step is to apply this contribution on Body-Monitoring system (BMS) by the grouping of these RA2DL components in RA2DL-Pool, we assign for each Pool a sensitivity level of these components. We plan in the future works to study the flexibility of RA2DL component in the network that links different devices of RA2DL-based systems. This work will be extended for different real-time aspects of RA2DL or in the run-time tests of components once deployed on the target devices.

References

1. Adaili, F., Mosbahi, O., Khalgui, M., Bouzefrane, S.: New solutions for useful execution models of communicating adaptive RA2DL. In: Fujita, H., Guizzi, G. (eds.) SoMeT 2015. CCIS, vol. 532, pp. 87–101. Springer, Cham (2015). doi:10.1007/978-3-319-22689-7_7
2. Adaili, F., Mosbahi, O., Khalgui, M., Bouzefrane, S.: Ra2dl-pool: new useful solution to handle security of reconfigurable embedded systems. In: Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering (ENASE), pp. 102–111, Rome, Italy (2016)
3. Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Yi, W.: UPPAAL — a tool suite for automatic verification of real-time systems. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) HS 1995. LNCS, vol. 1066, pp. 232–243. Springer, Heidelberg (1996). doi:10.1007/BFb0020949
4. Bernstein, D.: Containers and cloud: from LXC to docker to kubernetes. IEEE Cloud Comput. **1**(3), 81–84 (2014)
5. Bieliková, M.: A body-monitoring system with EEG and EOG sensors. J. ERCIM News **49**, 50–52 (2002)
6. Brereton, P., Budgen, D.: Component-based systems: a classification of issues. Computer **33**(11), 54–62 (2000)
7. Xia Cai, M.R., Lyu, Wong, K.-F., Ko, R.: Component-based software engineering: technologies, development frameworks, and quality assurance schemes. In: Seventh Asia-Pacific Software Engineering Conference (APSEC 2000), Proceedings, pp. 372–379 (2000)
8. Clements, P.C.: A survey of architecture description languages. In: Proceedings of the 8th International Workshop on Software Specification and Design (IWSSD 1996), p. 16, Washington, DC, USA. IEEE Computer Society (1996)
9. Rugina, A.E., Kanoun, K., Kaâniche, M.: An architecture-based dependability modeling framework using AADL. In: 10th IASTED International Conference on Software Engineering and Applications (SEA 2006) (2006)
10. Adaili, F., Mosbahi, O., Khalgui, M., Bouzefrane, S.: Ra2dl: new flexible solution for adaptive AADL-based control components. In: Proceedings of the 5th International Conference on Pervasive and Embedded Computing and Communication Systems, pp. 247–258 (2015)
11. Hansson, J., Feiler, P.H., Morley, J.: Building secure systems using model-based engineering and architectural models. CrossTalk J. Defense Softw. Eng. **21**(9), 12 (2008)
12. Husemann, D., Steinbugler, R., Striemer, B.: Body monitoring using local area wireless interfaces. US Patent Ap. 10/406,865, 7 October 2004
13. Oman, P., Alves-Foss, J., Harrison, W.S., Taylor, C.: The MILS architecture for high assurance embedded systems. Int. J. Embedded Syst. **2**, 239–247 (2006)
14. Jürjens, J.: UMLsec: extending UML for secure systems development. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 412–425. Springer, Heidelberg (2002). doi:10.1007/3-540-45800-X_32
15. Kocher, P., Lee, R., McGraw, G., Raghunathan, A.: Security as a new dimension in embedded system design. In: Proceedings of the 41st Annual Design Automation Conference (DAC 2004), New York, NY, USA, pp. 753–760. ACM (2004). Moderator-Ravi, Srivaths

16. Mouratidis, H., Kolp, M., Faulkner, S., Giorgini, P.: A secure architectural description language for agent systems. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), New York, NY, USA, pp. 578–585. ACM (2005)
17. Anoop, M.S.: Security needs in embedded systems. Cryptology ePrint Archive, Report 2008/198 (2008). http://eprint.iacr.org/
18. Ray, A., Cleaveland, R.: A software architectural approach to security by design. In: 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), Chicago, Illinois, USA, 17–21 September, vol. 2, pp. 83–86 (2006)
19. Ren, J., Taylor, R.: A secure software architecture description language. In: Workshop on Software Security Assurance Tools, Techniques, and Metrics, pp. 82–89 (2005)
20. Salem, M.O., Ben Mosbahi, O., Khalgui, M., Frey, G.: ZiZo: modeling, simulation and verification of reconfigurable real-time control tasks sharing adaptive resources - application to the medical project bros. In: Proceedings of the International Conference on Health Informatics, pp. 20–31 (2015)
21. Vergnaud, T., Pautet, L., Kordon, F.: Using the AADL to describe distributed applications from middleware to software components. In: Vardanega, T., Wellings, A. (eds.) Ada-Europe 2005. LNCS, vol. 3555, pp. 67–78. Springer, Heidelberg (2005). doi:10.1007/11499909_6
22. Yoon, E.-J., Lee, W.-S., Yoo, K.-Y.: Secure PAP-based RADIUS protocol in wireless networks. In: Huang, D.-S., Heutte, L., Loog, M. (eds.) ICIC 2007. CCIS, vol. 2, pp. 689–694. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74282-1_77