Formal Analysis of an IoT-based Healthcare Application

Maissa Elleuch*¶, Sofiène Tahar§
* Digital Research Center of Sfax, Technopark of Sfax, Sfax, Tunisia
¶ CES Laboratory, National School of Engineers of Sfax, Sfax University, Tunisia
§ Dept. of Electrical & Computer Engineering, Concordia University, Montreal, Quebec, Canada
maissa.elleuch@crns.rnrt.tn, tahar@ece.concordia.ca

2023 IEEE Symposium on Computers and Communications (ISCC) | 979-8-3503-0048-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISCC58397.2023.10218130

***Abstract*—In the healthcare context, remote monitoring based on the Internet of Things (IoT) technology is a widespread application. Underlying entities are interacting to bring up various services, so that their communication has to be ensured without defects such as deadlocks. The correct validation of these IoT applications is a major concern because of their distributed and concurrent features, as well as, the safety-critical nature of the health context. In this paper, we show how we use a model checking approach to accurately validate the behavior of an IoT-based healthcare application. We then focus on verifying three important classes of properties namely safety, liveness, and absence of deadlock. The verification is guaranteed by means of the UPPAAL model checker.**

***Index Terms*—Remote Monitoring System, IoT, Model Checking, UPPAAL**

I. Introduction

The healthcare domain has benefited from the technological Internet of Things (IoT) advancements [1] and their combination is nowadays a reality. Remote monitoring is considered one of the most common e-health application [2]. This kind of monitoring generally relies upon a Wireless Body Sensor Network (WBSN) [2] designed to collect vital signs, such as heart rate and body temperature. Such IoT-based monitoring can have many purposes like personal healthcare and fitness.IoT systems are usually built upon many distributed entities, that are completely concurrent bringing up various services, and communicating over potential faulty channels. Specifying such complex systems, can be thus error-prone. Nevertheless, in a safety-critical context like healthcare, the correct operation of controlling software is primordial, and any misfunctionality can cause serious losses in money, time and even lives [3]. It is imperative to ensure that systems meet all expected requirements, especially for the critical components of the system. There is thus an urgent need for efficient validation approaches that can effectively detect defects at all stages of the life cycle of the system since its specification.

For validation purposes, simulation-based testing has been heavily conducted for many kinds of healthcare systems [4], and in particular concurrent software. Simulation is considered as well-established and practical, but it suffers from many shortcomings. For example, it is almost unbelievable to achieve an exhaustive testing, while covering all possible scenarios for complex systems. A complete set of input cases is impossible to test. Consequently, many defects can be left undetected. In addition, simulation can be also error-prone.

To cope with all these issues, more rigorous design tools are required to enhance the safety of IoT-based applications in general, and checking their functional correctness prior to execution. Formal methods [5] have been highly recommended as an efficient solution to ensure the analysis and correctness of concurrent software at different levels of its life cycle, improving the satisfaction of the requirements that can impact the quality of the service delivery. Research in formal methods is constantly growing leading to the development of various verification techniques with powerful support tools for an early detection of defects. Model checking [6] is one of the most powerful formal methods for verifying the logical correctness of such concurrent systems. Based on a solid mathematical foundation, efficient formal support is applied to find out potential problems at the design stage and catch up errors. Classical as well as advanced properties can be thus validated.

In this paper, we show how we use a model checking approach to validate the correctness of an IoT-based healthcare application deployed in smart spaces. The application provides a classical service of data collection for single values. The presented application offers a service of data aggregation, combining values from monitored people and environment. Both monitored sides are seamlessly integrated into a single unit, where data can be extracted from. Our approach will check if the system behaves according to its specification. The verification is achieved in UPPAAL ; a tool for formal verification of real-time systems using timed automata [7].

The remainder of this paper is organized as follows. Section II is dedicated to review some related work. We overview the required preliminaries in Section III. In Section IV, a global presentation of the verified application. In Section V, we describe the formalized IoT system in terms of finite state automata within UPPAAL. Thereafter, the main verification results are discussed in Section VI. Section VII concludes the paper and points directions to future work.

II. Related Work

Over the years, a large body of work has been achieved for formally verifying various aspects of e-health systems. Chen et al. [8] studied the verification of an IoT system for elderly health cabin within UPPAAL by verifying the correct

activities of the system in terms of deadlock, functionalities, and timeliness. In [9], a formal approach for analyzing ambient assisted living solutions, is proposed using both exhaustive and statistical model checking. Recently, the authors of [10] assessed the correctness of patient activity in medical games. The system behavior is modelled as discrete-time Markov chains enriched with event occurrence probabilities within the PRISM and Storm frameworks. In [11], the authors provided an interesting approach to achieve accurate predictions about viral infection based on statistical model checking.

There exist also other similar related work to the above ones that use various approaches to validate several kinds of healthcare systems, by checking target properties using dedicated tools. In general, it has be noticed that while healthcare systems, based on IoT technology, can be saftey-critical, they occupy a major part of our living without an important focus on their verification.

III. Preliminaries

In this section, we introduce model checking, as well as, the UPPAAL model checker.

A. Model Checking

Model checking [6] is one of the most widely used technique for system verification. Model checking has a pure behavioral view of the system. The main idea is to build a state-based model of the target system, and to translate various properties into formulas in a given logic. The verification is achieved by an exhaustive checking of all pathways of an executable specification. In case the property is not satisfied, a counter-example is displayed. The system model can be more or less complex, depending on the abstraction level, but finite. Model checking outperforms simulation and testing providing larger coverage and more quality assurance. Model checking has the ability to easily deal with various properties such as deadlock, safety, and liveness. Nevertheless, the technique is subject to limitation regarding the state space explosion due to the system complexity.

B. The UPPAAL Tool

UPPAAL is a mature tool providing an accurate modeling and checking of real-time systems [7].The tool includes two parts, a graphical user-interface and a verification engine based on model-checking. Finite state automata are used to model the system behavior. In UPPAAL, the finite state automata communicate through channels.

Here are some of the most used notations in UPPAAL.

- **ch?**: there is a receive operation on channel ch
- **ch!**: there is a send operation on channel ch
- The start state is represented by a location with a circle
- The committed state is represented by a location with the letter **c**

A checking formula in UPPAAL is a combination of the following 'operators' [12]: E, A, $<>$, and $[\,]$. If p is the property to be checked, then the common formulas are:

- **E**<> **p**: there exists a path where p eventually holds.
- **E**[ ] **p**: there exists a path where p always holds.
- **A**[ ] **p**: for all paths p always holds.
- **A**<> **p**: for all paths p will eventually hold.
- **p** $\rightarrow$ **q**: whenever p holds q will eventually hold, where p and q are logical expressions.

UPPAAL supports the verification of various properties expressed in Computational Tree Logic (CTL), namely [7]:

- **Reachability:** this property checks if a certain state can be reached or not. The deadlock in the system is expressed through this property.
- **Safety:** it always prevents the system from the occurrence of "something bad", i.e., something good is always true.
- **Liveness:** UPPAAL checks if "something good eventually happens". A typical example is a request that will be at least satisfied once during the system execution.

IV. Overview of the Monitoring Application

A. Architecture

The studied application is an e-health application in smart spaces based on a WSN [13]. The system architecture is presented in Figure 1. There are mainly three subsystems: the operator, the Wireless Sensor Network (WSN), and the Body Area Network (BAN). The operator is making requests through the base station, the WSN is responsible of the communication part and is routing the requests, as well as, the responses, and the BAN is composed of the wearable devices monitoring the user body conditions. The WSN is mainly made of many nodes covering the monitored area and measuring several context parameters, such as temperature and humidity. The WSN is usually put in a hierarchical way so that nodes can have different functionalities: broker, orchestrator, and measurement nodes. Finally, the BAN, built upon the user, is a network composed of several wearable devices and a bridge sensor node. The monitoring motes should be able to interact with the WSN. The BAN can be used for coaching routines or vital signal monitoring. The user makes interaction with the BAN using different devices, such as smart phones, smart watches and tablets.

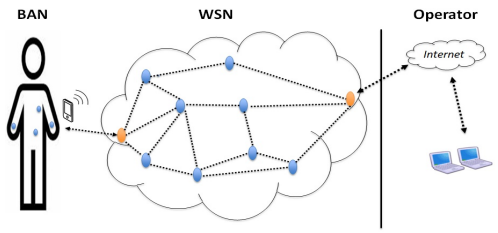

Fig. 1: The Application Architecture

B. Services

The application is designed to simultaneously monitor the user conditions, as well as the environment, offering two main services. The application provides a service of data collection for simple parameters. These simple parameters are extracted from the user environment, such as temperature or humidity,

or from a wearable device, that monitors the user physiological conditions (breathing rate, heart rate, body temperature) from a smart device (e.g. smart watch). The second service is the result of a data aggregation. This kind of service is also named as composed service. In this case, the system cannot deliver responses to the user, unless it communicates with other parties. The two data values collected from both the monitored context and the user are merged, and a representative message is displayed for the operator.

A typical scenario, given in [13], consists in a real application including a sportsman in a gymnasium. The basic service of data collection is to check the performance of the sportsman by measuring his physiological conditions through some wearable devices, and then suggest suitable coaching exercises. The second service of data aggregation consists in controlling the temperature conditions of the sportsman and the context simultaneously. The obtained data is evaluated according to the fixed thresholds, then a response about the temperature level is provided.

C. The Roles

Next, we focus on the roles implied to ensure both services of data collection (simple or aggregated). There are five roles:

- **The operator:** it is a surveillance officer equipped with a system capable to make requests from a web browser of any system (PC, laptop) or a Graphic User Interface (GUI) on a given device (smartphone, tablet).
- **The base station:** the main functionality of a base station is to establish a connection between the system making requests, i.e., the operator and the monitoring network. A base station works like a gateway, but it is not a real node with data storage capacities.
- **The broker agent:** the broker agent has usually the responsibility to forward the request to the node able to deliver the response. Once the request is received, the broker agent will send it either to the suitable data node or to the orchestrator.
- **The orchestrator agent:** The orchestrator handles a composed request by separately sending each of the simple request to the agents able to respond. Similar to the broker, the orchestrator should not measure anything.
- **The data agent:** it is usually a simple mote able to deliver the requested data.

D. The Flow

We provide a brief explanation of the procedure of how to retrieve the data for a data aggregation service, namely the interactions between the roles described above. When the operator sends a request to the WSN, it is first received by the broker agent. In the case of a data aggregation request, the broker will not transmit the request directly to the monitoring node, but it will forwarded it to the orchestrator agent. Indeed, the orchestrator has an idea about which nodes are able to respond the request and which data is needed. The orchestrator will thus decompose the current request into two simple services. Then, it will transfer the simple request one by one to the broker. The original composed request is handled as two simple service requests in a sequential manner. Once the simple request received by the broker, it will communicate with the suitable data agent in order to get the requested data. The data agent will send back the response to the broker, which in turn to the orchestrator. As soon as the required data aggregated, an evaluation is achieved using different levels. These levels are set according to given thresholds. If there are issues at any level, error messages are displayed and default values are used. A more detailed explanation of the flow of the present application can be found in [13].

V. Modelling in UPPAAL

Our goal is to achieve a formal verification of the IoT-based application, presented above, using the UPPAAL model checker. To achieve the invocation of UPPAAL, we first need to build the system model as finite-state machines and specify then the requirements to be verified in the target temporal logic. The main focus is to model the functionalities for the data aggregation service.

The model consists of five state machines, one for each of the roles described earlier. Figures 2 and 3 represent the orchestrator and the broker, respectively. Below, we provide a brief explanation of the models. The data agent model has been omitted for space constraints. Once a request is received from the user, the base station side will switch it to the monitoring network through the broker agent on the channel req_BS_Br. The broker receives the main request from the base station on channel req_BS_Br. Through a Boolean value named $CreqBr$, the broker has first to check if it is a service requiring data aggregation or a simple service. In case $CreqBr$ is $true$, the broker forwards the request to the orchestrator on channel $Creq\_AvT\_Br\_Orch$. The orchestrator will thus decompose the aggregation request and sends each simple service to the broker. Indeed, the orchestrator has an idea about the required data, and the nodes from where to get the response. But, as we mentioned before, the orchestrator should not make any measurements by itself. From now, the initial request of data aggregation is handled as two simple services, performed in a sequential manner by the orchestrator instead of the human. The broker hence initiates a simple service to the suitable data agent, requesting the required data for each simple service. Afterwards, the data agent proceeds to submit the collected data to the broker on channel $DATA\_DA\_Br$.

Once data is received, the broker sends it to the orchestrator on channel $DATA\_Br\_Or$, and enters in a wait state. Based on a Boolean value named req, the orchestrator checks if the required data has been gathered. In case the required data is not yet collected, the orchestrator launches the second simple request towards the broker over the channel req_Or_Br. Within the orchestrator automata (Figure 2), this corresponds to the transition with the guard ($req >= 1$). Otherwise, the orchestrator evaluates the collected values over a threshold and then communicates the appropriate message to the broker

agent on channel $Dataok\_Or\_Br$. The resulting message will be displayed on the final user screen.

The flow described above is the common normal behavior of the system. Nevertheless, once one of the required conditions is not satisfied, the system fails to evolve leading to situations modeled as transitions to failure states. For example, if there is any problem with the request, an error message will be issued to the broker, who in turn informs the base station and then the user on the channel Err_BS_User. Another example of failure scenario is when the data cannot be retrieved, prompting the broker to send the default data in response.

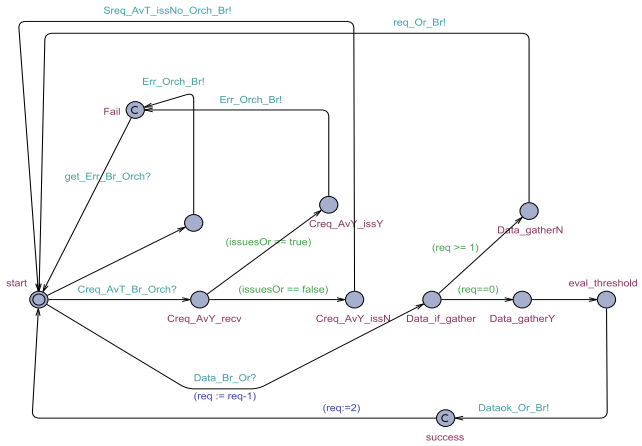

Fig. 2: The Orchestrator Automata

VI. Verification in UPPAAL

Once the finite automata developed in UPPAAL, we proceed with the verification of some interesting properties. Three kinds of properties were specified and formulated using propositional temporal logic [7]. Below, we provide explanations about these properties along with their UPPAAL formulas.

Deadlock-Freedom: This property guarantees that the developed models are all deadlock free. This means that the system never reaches a state where it cannot progress.

```
A[] not deadlock
```

Safety: This type of property ensures that whether or not something bad will happen. In the sequel, we specify various safety properties (SP-1-5) of our model.

SP1: The property verifies that the system is launched only if the user demands a service from the base station.

```
A<> (User.start imply BS.start) and
  (BS.start imply Broker.start) and
  (Broker.start imply Orchestrator.start) and
  (Broker.start imply DataAgent.start)
```

SP2: The data aggregation request, received by the broker, is always forwarded to the orchestrator for execution.

```
A[] (Broker.Creq_recv == true) imply
  (Orchestrator.start == true)
```

SP3: A request for data aggregation is decomposed into two simple requests only once.

```
A[] (Broker.Creq_av imply (CreqBr == false))
```

SP4: The orchestrator is invoked if the variable (req $\leq$ 2).

```
A[] (Orchestrator.start == true) imply
  (req <= 2)
```

SP5: Once the required data for a composed request is gathered, the variable req is automatically reinitialized.

```
A[] Orchestrator.Data_gatherY imply
  (req == 0)
```

Liveness: This kind of property ensures that "something good eventually happens". In following we describe several relevant liveness properties (LP1-5) to verify our system.

LP1: This property verifies that there exists at least one path in which the node enters the final state after the user requests the BS service in the start state. This ensures that there is at least one path along which the user request is successful.

```
E<> (User.start imply BS.success) and
  (User.start imply Broker.success) and
  (User.start imply Orchestrator.success) and
  (User.start imply DataAgent.success)
```

LP2: When the Broker reaches the state $Serv\_Av$, the value of the Boolean variable controlling the availability of the composed request is always true.

```
A[] (Broker.Creq_av == true) imply
  (avcreqBr == true)
```

LP3: When the user reaches the state success, there are no issues from the orchestrator side, and the value of the Boolean variable controlling the composed request is true.

```
A[] (User.success == true) imply
  (avcreqBr == true) and (issuesOr == false)
```

LP4: When the user reaches the state success, there is are no issues encountered from the orchestrator and the data agent.

```
A[] (User.success == true) imply (issuesOr ==
  false) and (issuesDA == false)
```

LP5: If there are any problems for getting data from the agent node, a default value is displayed.

```
A[] ((avsreqBr == false) or (issuesDA ==
  true)) imply (DataAgent.get_Def == true)
```

We have successfully verified each of the above properties on our health monitoring IoT model. The verification by model checking has been conducted in UPPAAL version 4.1.24, running in Windows 10 OS on an Intel(R) Core(TM) i5-8250U CPU with 8 GB of RAM. Experimental results showed an average memory consumption of 38 MB per property, while the whole verification consumed 4744 MB of RAM. In terms of CPU time, UPPAAL reported an average of 1.2 sec. per property, which is more than 10 folds faster than comparative experimental results based on verification by simulation [14].

In general, IoT-based solutions for healthcare are theoretically studied, then experimental evaluation through simulation is achieved.

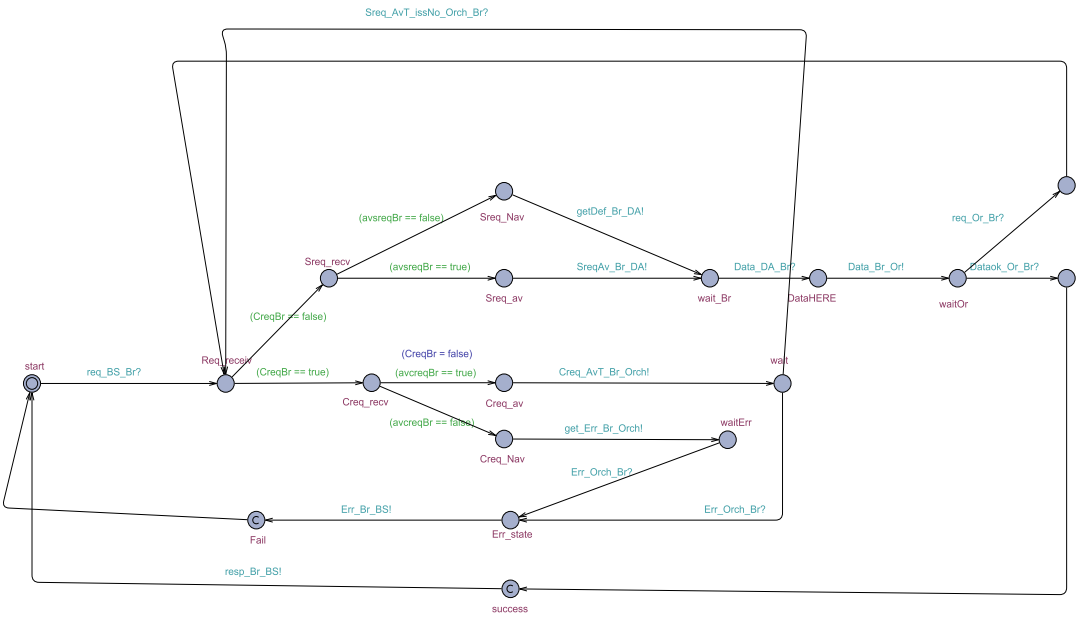

Fig. 3: The Broker Automata

Indeed, our study of related work, showed that paper-and-pencil analysis together with simulation are the most commonly used approach for the performance analysis of IoT-based e-health monitoring systems [15], [14]. Compared to these traditional techniques, our work uses formal verification, which provides a more generic and exhaustive validation while providing a more accurate behavioral view of the considered IoT application. In particular, we used model checking as a sound formal technique that automatically searches the entire state space in order to find out errors at an early stage of the IoT system development. In a summary, the evaluation under UPPAAL shows how the proposed formal verification technique outperforms standard validation approaches by ensuring an exhaustive verification of accurately specified system behaviors and hence providing more trust in the design of such safety-critical IoT-based healthcare application.

VII. Conclusion

Today, health infrastructure is able to provide high quality healthcare services, and health surveillance is considered as one of the most common applications. In the current work, we have presented an approach for the formal analysis of an IoT-based healthcare application deployed in smart spaces. We have been able to check the correctness of many safety-critical properties, within the UPPAAL model checker. The user friendly nature of UPPAAL leads to an attractive verification experiment. The proposed approach can be useful for the system developer in achieving better quality in the specifications and implementations. In addition, such approach can be adopted to verify various kinds of IoT-based systems in the healthcare context. As part of our future work, we plan to examine other services such as alarm activation.

References

[1] M. B. M.A. Feki, F. Kawsar and L. Trappeniers, "The internet of things: The next technological revolution," *Computer*, vol. 46, no. 2, pp. 24–25, 2013.

[2] B. Latré, B. Braem, and I. M. et al., "A survey on wireless body area networks," *Wireless Netw*, vol. 17, pp. 1–18, 2011.

[3] L. T. J. M. C. Kohn and M. S. Donaldson, *To err is human: Building a safer health system*. Institute of Medicine, Washington, DC: The National Academies Press, 2000.

[4] S. Brailsford, P. Harper, B. Patel, and M. Pitt, "An analysis of the academic literature on simulation and modelling in health care," *J. Simulation*, vol. 3, no. 3, pp. 130–140, 2009.

[5] O. Hasan and S. Tahar, "Formal verification methods," *Encyclopedia of Information Science and Technology*, pp. 7162–7170, 2015.

[6] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Trans. Program. Lang. Syst.*, vol. 8, no. 2, pp. 244–263, 1986.

[7] "The model checker UPPAAL," https://uppaal.org/, 2023, online.

[8] G. Chen, T. Jiang, M. Wang, X. Tang, and W. Ji, "Modeling and reasoning of iot architecture in semantic ontology dimension," *Comput. Commun.*, vol. 153, pp. 580–594, 2020.

[9] A. Kunnappilly, R. Marinescu, and C. Seceleanu, "A model-checking-based framework for analyzing ambient assisted living solutions," *Sensors*, vol. 19, no. 22, p. 5057, 2019.

[10] T. L'Yvonnet, E. D. Maria, S. Moisan, and J. Rigault, "Probabilistic model checking for human activity recognition in medical serious games," *Sci. Comput. Program.*, vol. 206, p. 102629, 2021.

[11] S. Chehida and J. Mfumu, "Analysis and prediction of viral infections using statistical model checking," in *Management of Digital EcoSystems*. ACM, 2021, pp. 43–48.

[12] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *Lectures on Concurrency and Petri Nets*, ser. LNCS, vol. 3098. Springer, 2003, pp. 87–124.

[13] P. Castillejo, J. Martínez, J. Rodríguez-Molina, and A. Cuerva, "Integration of wearable devices in a wireless sensor network for an e-health application," *IEEE Wireless Commun.*, vol. 20, no. 4, 2013.

[14] S. Ghanavati, J. Abawajy, D. Izadi, and A. Alelaiwi, "Cloud-assisted IoT-based health status monitoring framework," *Clust. Comput.*, vol. 20, no. 2, pp. 1843–1853, 2017.

[15] S. Kafhali and K. Salah, "Performance modelling and analysis of internet of things enabled healthcare monitoring systems," *IET Networks*, vol. 8, no. 1, pp. 48–58, 2019.