QoS-Driven Self-Adaptation for Critical IoT-Based Systems

Arthur Gatouillat, Youakim Badr, Bertrand Massot

QoS-Driven Self-Adaptation for Critical IoT-Based Systems

Arthur Gatouillat[1], Youakim Badr[1] and Bertrand Massot[2]

[1] Univ Lyon, INSA Lyon, LIRIS, UMR5205
(arthur.gatouillat,youakim.badr)@insa-lyon.fr
[2] Univ Lyon, INSA Lyon, INL, UMR5270
bertrand.massot@insa-lyon.fr

Abstract. The Internet-of-Things, which designates the interconnection of numerous physical devices, is a growing research direction faced with many challenges. One of these challenges is to provide constant quality-of-service despite IoT devices being used in a constantly changing physical environment. In order to answer this problem, we introduce a quality-of-service driven self-adaptation framework, which can simultaneously handle changing adaptation strategies, monitoring infrastructure and physical environment while guaranteeing constant quality-of-service. Because of its formal guarantees, our system is particularly suited for the control of critical IoT-based systems, and we thus demonstrated its practicality by applying it to an e-health case-study where the safety of the monitored patients must be assured.

Keywords: Self-adaptive systems, Adaptive IoT, Controller synthesis.

1 Introduction

The Internet-of-Things (IoT) enables the interconnection of virtually every object of the physical world, such as a variety of sensors, actuators, robots or wearable devices. This interconnection of various physical-world devices into IoT systems, coupled with the strong hardware constraints of IoT devices, mandates the study of adaptation strategies to deal with devices failure, especially when dealing with critical systems (e.g. healthcare systems). To deal with such requirements, self-adaptation software and autonomic frameworks were proposed [1–4]. In particular, self-adaptive software systems (SAS), which deal with distributed applications in changing environments, normally require human supervision to sustain despite changes during their executions. These systems rely on a closed feedback loop to adjust themselves to changes. Based on observed context variables and thresholds, they monitor themselves, their context and entities from the target system environment to decide when and how to apply adaptation strategies in order to ensure expected behavior (i.e., functional requirements) and guarantee quality of services (i.e., non-functional requirements) [5].

In the context of the Internet-of-Things, self-adaptation is a salient property of smart objects. It allows them to be self-configured and adapted to extreme conditions while

ensuring the target system objectives such as comfort, automation, security and safety goals. Self-adaptation mechanisms driven by adaptation goals modify smart objects behavior dynamically. However, the IoT is a dynamic and global network infrastructure, in which "things" are expected to be autonomous and self-configurable. This feature that the IoT should strive to achieve is not a trivial task since adaptation goals may also evolve continuously during run-time due to changes in functional or non-functional requirements and contextual information. These changes affect observed variables, their thresholds and even alter the monitoring logic when context variables are added or deleted. As a result, when adaptation goals change at run-time, both monitoring and self-adaptation mechanisms may become inapplicable because either the adaptation mechanism deals with outdated goals, or the monitoring mechanism addresses monitoring requirements that are irrelevant to the actual adaptation goals.

In this paper, we present a discrete controller synthesis system that ensures self-adaptation behavior based on quality-of-service properties (QoS) of smart objects during their run-time. Our system is resource-aware and ensures simultaneously the separation of concerns of adaption objectives, context monitoring and adaptation strategies. By such, our system makes possible to guarantee the service level agreement (SLA) which defines the level of smart object services as expected by end-users expressed them as constrains on non-functional properties. Without loss of generality, our system focuses on the safety quality as a crucial non-functional property in the context of healthcare monitoring to detect heart malfunctions with wearable sensors (heart rate sensor and electrodermal activity sensor) and ambient sensors (occupancy detection sensor, noise sensor, etc.). In this context, we model the safety quality as a qualitative property in the SLA of wearable and ambient sensors. The safety quality is initialed defined as the resource-awareness factor of wearable sensors in response to their battery consumption levels. During the execution, the safety quality may be renegotiated by adding resilience as an additional new factor. This new contracted factor states that the defected objects are subsumed by alternative smart objects (same types) or inferring their observed values through data collected from other smart objects (different types). Yet another adaptation of the safety property may include the health-awareness factor, which enables the detection of critical health crisis from contextual information, medical sensor values, and patient history patterns. Our discrete controller synthesis system implements a dynamic monitoring approach that deploys, at run-time, new context gatherers and new monitoring requirements for new quality properties. These elements are automatically generated from the non-functional quality-of-service properties in the SLA and deployed at run-time without interrupting the target system execution or the adaptation mechanism.

The remaining of the paper is organized as follows. In section 2, we present related works. Before introducing our discrete controller synthesis system, we briefly introduce a motivation case-study dealing with safety quality in wearable sensors in section 3. In section 4, we present our QoS-driven self-adaptation approach which is based on the DYNAMICO reference model [3] and includes SLA and stated-based failure ontologies. These ontologies along with knowledge-based rules are used to generate non-

functional device labeled transition graphs and to generate our discrete controller synthesis system as described in section 5. In section 6, we present our implementations and conclude our work in section 7.

2 Related Works

The work detailed in this paper is located at the intersection of three distinct but related communities: home automation, classical control theory and software auto-adaptation.

The home automation community deals with the integration of smart devices such as sensors, actuators and gateways into houses in order to better monitor and control living environments. Home automation is a broad concept, including smart home and ambient intelligence (AmI), which generally refers to architectures, practices, and controllers for proper management of the home life-cycle to address home safety, energy efficiency, entertainment, ambiance, assisted living, fall detection, elderly care or patient monitoring [6]. Controllers are key components of home automation. Rule-based controllers have been widely explored in an AmI context and more particularly in the context of remote health-monitoring: fuzzy rules were mixed with case reasoning in [7] to provide both home-automation and health monitoring to elderly patients. Ontology-derived rules are also used in coordination with rule engines to provide functional intelligent building behavior [6]. Rule-based framework in the context of remote monitoring where explored by [8] to provided assistance in decision-making for healthcare providers. Many contributions in the field of rule-based home automation have focused on the management of functional properties, but they lack consideration of system non-functional properties and adaption to situation where some sensors are failing but the system must still perform. The main concern of these contributions is the functional coordination of distributed sensors and actuators, instrumenting a house in order to achieve predefined goals. A rule language approach allows, thanks to its relative expressiveness, the specification of control objectives that can then be executed using rules engines.

The software adaptation community deals with the integration of strategies enabling better handling of changing digital and physical environment to modular software systems. These contributions can be divided into three categories: contributions which consider the adaptation of classical control frameworks to software systems, contributions which consider the use of monitor analyzer planner executor and knowledge feedback (MAPE-K) loops [9] as the basic building block to enable software adaptation, and finally contributions combine classical control and MAPE-K loops to implement adaptive software solutions.

When only classical control solutions are applied to adaptation problems, the main research challenge resides in the accurate state space modelization of software processes. MPC-based [10] and PID-based [11] adaptation framework were able to provide results in terms of software adaptation. However, we believe that the cumbersome modeling process that must occur for each software system is not suitable for IoT applications. Indeed, self-adaptation for the IoT must be able to handle potentially very heterogeneous devices, that are not easily modeled using state space representations. Indeed,

the feedback loop modelization of classical control systems does not provide good modularity, as elements of the feedback loop are not standard, and can vary between systems. Standardization of the feedback loop elements is provided by the MAPE-K self-adaptation framework.

MAPE-K is considered as a gold-standard for self-adaptive systems [12, 3, 4]. The idea behind the MAPE-K control scheme is to define autonomic elements defining an adaptation loop from monitors (i.e. sensors) to executors (i.e. actuators) that perform system reconfiguration using knowledge shared between all the feedback loop elements. Because this feedback loop is only a reference model, multiple implementations have been studied such as agent-based implementations [12] or using formal frameworks such as FORMS [4]. However, such control feedback loops are purely software based, and are not appropriate for the control of hybrid software-hardware systems such as IoT systems. Contributions considered the mixed use of classical control loops and MAPE-K control loops to enable software systems with self-adaptation properties. This is the case of DYNAMICO [3], an architectural reference model equipped to deal with changing system requirements and monitoring infrastructure thanks to separation of concerns. In this architecture, three independent MAPE-K loops are interconnected using classical control feedback loops in order to deal with changing system objectives, changing monitoring framework and system adaptation. This reference model is however oriented at purely software self-adaptive systems, and mandates some modifications to be used in the context of hybrid IoT systems. Another field of interest when it comes to control theory is discrete controller synthesis (DCS). In this field, discrete models of target to-be-controlled systems are used to build correct-by-construction discrete controllers [13]. Such control strategy has been successfully used in the IoT context for functional adaptation of simple home-automation systems [1]. The control objectives in this contribution are given as first order-logic rules, and some non-functional concerns are integrated under the form of controlled energy consumption. The strategy behind such control framework is to divide the systems into a set of controllable and non-controllable states, and to use the controllable states to guarantee system objectives fulfilment given non-controllable states. Originally, such discrete control systems were built using synchronous programming languages such as SIGNAL [13] or BZR [1, 14, 15]. Event-condition-action rules based discrete controller synthesis was explored, more particularly using an event-condition-action (ECA) rules based high level description language to BZR translation to perform controller synthesis [16, 17]. Asynchronous controller synthesis methods in the context of cloud-based autonomic manager was studied using the LNT framework [18]. The main advantage about these contributions is that the controller verification part can be avoided because controller synthesis is assumed to produce a correct controller. Table 1 summarizes all the contributions described in this section.

Our contribution is at the center of all the before-mentioned contributions since it adapts the DYNAMICO reference model to the context of IoT and uses MAPE-K and classical feedback control loops to enable the capability of dealing with changing system objectives and monitoring infrastructures. By using a state-chart model of non-functional properties and high-level ECA-rules, we generate IoT system controllers with well-established DCS tools such as the Heptagon/BZR toolbox.

Table 1. Related contribution synthesis.

Community	Concerns	Tools	Papers
Self-adaptive software	Enabling software to feature adaptive behavior to changing environment.	MAPE-K, DYNAMICO, FORMS	[9], [12, 3, 4]
Discrete controller synthesis	Build correct controllers using discrete models of controlled systems.	BZR/Heptagon, LNT, Signal	[1], [14], [15] [16, 17], [18]

3 Motivation Case-Study

We consider the surveillance of a patient at risk of myocardial infarction recurrence as a motivation case study. In this context, the patient requires continuous monitoring of physiological parameters to detect potential recurrences and to urge a rapid medical response if a heart failure is detected. Continuous monitoring is achieved using wearable wireless sensors, which are battery powered, resulting in different resources constraints. In addition, the living environment of the patient is instrumented with sensors and actuators that are either continuously powered or battery powered. As the case of most IoT-based applications, connected objects are strongly constrained in terms of resources: all sensing and actuation devices feature limited computing abilities (CPU frequency up to a few hundreds of megahertz), storage (up to a few megabytes) and volatile memory (up to a few hundreds of kilobytes). Constrained resources also imply limitations in terms of communication protocol, which must be lightweight in order to avoid the introduction of processing overhead and limit energy consumption.

In this case-study, we particularly focus on the robust detection of heart malfunctions, and the triggering of emergency medical response if such a situation occurs. There are two main robustness requirements: the avoidance of false positive detection of heart malfunction (i.e., the system detects a cardiac malfunction while there is none), but more importantly the detection of cardiac failure even if the system does not operate at full capacity.

Considering self-adaptive properties of such a system, this case-study is of particular interest: the adaptation goal is to ensure the safety property while satisfying quality of service of a continuous and reliable monitoring. To satisfy the safety goal, the adaptation strategy is based on the resource-awareness factor, resilience factor (i.e., substitution of defected objects with alternatives) and healthcare awareness factor such the request for medical assistance (i.e., myocardial infarction detection) or technical intervention (i.e., abnormal values). Consequently, a safety-enabled smart home is implemented to support self-adaptation objectives based on resources consumption, resilience and external assistance. The adaptation strategies (i.e., the mechanisms that affect the target system) consist of modifying smart sensor parameters based on resource monitoring, substituting defected objects with alternatives or inferring their values from nearby smart-objects as well as call for medical assistance when detecting abnormal values. The context variables mandating observation are battery levels, absence or abnormal values, exceeding medical thresholds that define myocardial infarction.

In the following sections, we limit ourselves to examples using only three sensors: the battery-operated heart rate (HR) and heart rate variability (HRV) sensor, the battery-operated electrodermal activity (EDA) sensor, and the continuously powered occupancy sensor. The cardiac risk is thus evaluated by a cardiac health estimation service including inputs:

- Continuous streams of HR and HRV values (optimal mode)
- Continuous streams of EDA values with instantaneous or average HR values (failsoft mode)
- Continuous streams of presence values (critical mode)

Depending on these inputs, the monitoring controller uses its internal cardiac health estimator model to infer cardiac health status and request medical help for cardiac malfunctions. Different cardiac health estimation models are out of scope since we only focus on how the controller is self-adapted to provide estimators with correct inputs at all time, and how it triggers external tiers notification, if deemed necessary.

4 QoS-Driven Self Adaptation

4.1 Managing Changing SLA and Monitoring Environment

Purely functional and static adaptation was successfully studied in IoT [1, 17]. However, insuring that IoT-based systems behaves as specified under changing control objectives of non-functional properties and limited resource-awareness is a relatively unexplored research field. The DYNAMICO reference model [3] provides a prominent solution to design and implement self-adaptive software systems where both adaptation and monitoring infrastructures are enabled with self-adaptive capabilities using three types of feedback-loops:

- The *objectives feedback loop*, which governs changes in adaptation goals, also called control objectives (e.g. SLAs);
- The *target system adaptation feedback loop*, which regulates the target system requirements satisfaction and the preservation of the adaptation properties;
- The *dynamic monitoring feedback loop*, which infers context variables from the contracted quality of service (QoS) conditions and adapts the architectural reconfiguration of the monitoring infrastructure to implement the monitoring logic associated with context variables.

DYNAMICO characterizes the separation of concerns and interactions among different types of feedback loops. Despite its prominent advantages, DYNAMICO implementations (i.e., SMARTERCONTEXT monitoring infrastructure with the QoS-CARE/FRASCATI middleware) are not relevant to distributed smart-objects with limited resources.

Based on the based on the DYNAMICO reference model, we propose an IoT-targeted self-adaptive system, including a distributed adaptation infrastructure and the controlled IoT system, consisting of gateways, each of which interacts with one or more devices or sensors (see Fig. 1). Gateways invoke device services to get their data streams, adjust their functional parameters, monitor their non-functional properties and perform adaptation operations when deemed necessary, as defined in their SLAs. More

precisely, the IoT-based self-adaptive system relies on a SLA for each sensor, and a global SLA for the system as a whole to respectively set adaptation objectives and deploy monitors to gateways to observe each sensor's QoS. Monitors informs the discrete controller with events to decide whether that a self-adaption strategy should be applied to meeting end-users' SLAs. The adaptation infrastructure in Fig. 1 illustrates the causal relationships between adaptation objectives, monitors and discrete controller, and interactions between different feedback loops.

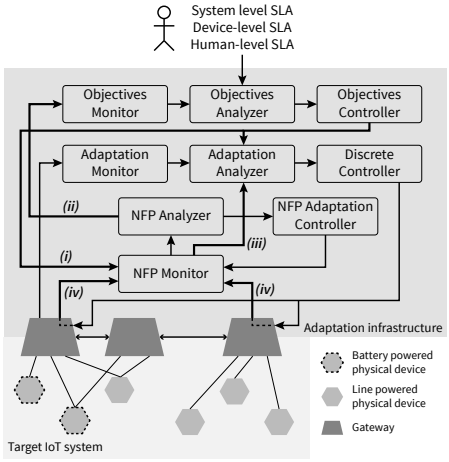

Fig. 1. An IoT-based self-adaptive system based on the DYNAMICO reference model

*Interactions **(i)** between the objectives feedback-loop and the adaptation feedback-loop (the monitoring feedback-loop resp.)*: These interactions feed the reference control input computed by the objectives controller to the adaptation loop and the monitoring loop. For instance, in our use-case, the first resource-aware adaptation to be considered is related to the sensors' battery levels by which the reference input will thus be under the form BatteryLevel > 20%. This reference will be used by the adaptation feedback loop as an element to be analyzed to decide potential adaptation, and by the monitoring feedback loop as a reference input for context monitoring.

*Interactions **(ii)** between the Monitoring Feedback Loop and the Objectives Feedback Loop*: These interactions characterize the detection of the need of a change in the control objectives by the monitoring feedback loop to be fed to the control objectives feedback loop. In our use case, if the battery is drained, and if the cardiac status monitoring must be inferred using other environmental sensors. This mandates a change of control objectives that must be decided by the objectives feedback-loop.

*Interactions **(iii)** between the Monitoring Feedback Loop and the Adaptation Feedback Loop*: These interactions feeds adaptation-triggering events from the monitoring loop to the adaptation loop. For instance, the consistent and more rapid than normal decrease of the battery level can be used as an event to trigger faster adaptation of the system and a quicker adoption of a battery-saving fail-soft mode in order to extend the duration of quasi-optimal system behavior.

*Interactions **(iv)** between the Adaptation Feedback Loop and the Monitoring Feedback Loop*: These interactions feed the internal context from the adaptation loop to the monitoring loop. This interaction can be used to insure system consistency after adaptation. For instance, in our use case and if the HR sensor has to be subsided by position sensor for health monitoring, this interaction is used to ensure that the position sensors are all in a functional state after the adaptation, thus insuring global system safety.

Fig. 2 describes the self-adaptation meta rules of the target IoT system behavior in response to changes in the SLAs (i.e. control objectives' changes) through interaction ***(i)*** and to changes in the monitoring infrastructure (e.g. sensor removed from network) through interaction ***(ii)***.

```
If new control objective (SLA renegotiated)
     Perform objectives analysis
     Generate new control contract
     Deploy monitors in gateways
     Perform discrete controller synthesis
If change in context (devices not available)
     Generate new control contract
     Deploy monitors in gateways
     Perform discrete controller synthesis
```

Fig. 2. Self-adaptation meta rules

4.2 Non-Functional Device Labeled Transition Systems

As illustrated in Fig 3(a), we propose an ontology to describe the global SLA for the IoT target system. Each non-functional property (i.e., safety) is thus defined in terms of QoS factors (resource-awareness, resilience, healthcare awareness) each of which has constraints expressed as service level objectives (SLO) and has a corresponding monitor deployed on the gateway of the sensors related to each QoS factor. We also propose a failure-adaptation ontology to describe the system global safety in terms of resources (i.e., low battery), resilience (unattached sensor, interrupted communication), abnormal data due to critical heart failures or hardware failures (digital/analog failures).

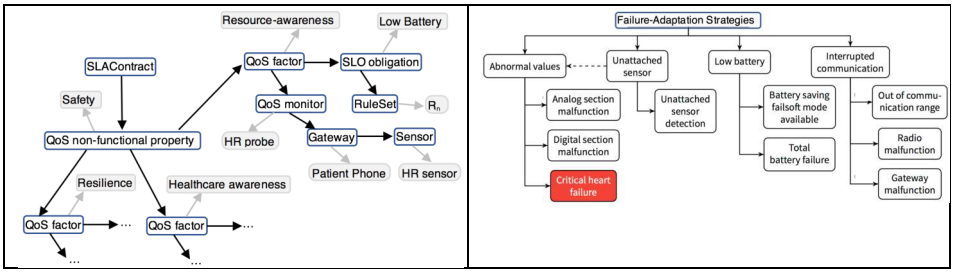

Fig. 3. (a) Global QoS Ontology (b) Failure-Adaption Strategies Ontology

A building block of our IoT-based self-adaptive system is the description of non-functional behaviors of each sensor using labeled state transition systems (LTS) based on the failure-adaption strategies' ontology. In fact, each state corresponds to a failure and transitions refer to adaptation strategies to ensure safety non-functional property of the target IoT system. This description allows the expression a qualitative quality-of-service in terms its quantitative factors which are observed with monitors at run-time and makes a correlate with appropriate adaptation strategies in case of failures. It also provides a discrete behavior using states and transitions, making possible to use toolbox (i.e., Heptagon/BZR) for synthesizing discrete controllers.

Generally speaking, a LTS is defined as a quadruple $(S, L, \rightarrow, s_{in})$, where S is a set of states, L a set of transition labels, $\rightarrow \subseteq S \times L \times S$ is a transition relation between two states, and s_{in} is an initial state. The set of transition labels is defined as $L = (events, actions, \backslash)$, where $\backslash \subseteq events \times actions$.

LTSs enable an accurate system description, where non-functional and non-controllable variables are members of the set *events*, and services calls are included in the set *actions*. This syntactic separation of non-functional and functional variables enables better expressivity: the statement "$e \backslash a$" can be interpreted as the control of event e by service a when event e during the firing of the transition. When no control service is provided for a given transition, it implies the non-controllability of the given transition. Battery operated 'smart' sensors are purely uncontrollable (because the system cannot automatically charge the battery, external human intervention is required for this operation). Fig. 4 describes the non-functional behavior of the heart-rate sensor used in our case study. It features both purely event-based and event \ action transitions. It is worth noticing that while some non-functional states can be self-detected by the sensor itself (such as the low-battery state or the unattached state), some other can only be inferred by the controller on a more global view (e.g. the analog malfunction state, which is accessed through the `abnormalValue` transition event). Because of space limitation, we will not detail what is considered as an abnormal value, nor will we detail the LTS of other sensors, which are assumed to be similar to the heart-rate sensor. The choice of LTS as a model of computation (MoC) of IoT-based devices was motivated by the genericity of this MoC, and because IoT-based devices traditionally present a discrete state-based behavior (e.g. a sensor can be on, measuring, off, etc.). Consequently, both personal and external-tier devices can be represented using this MoC.

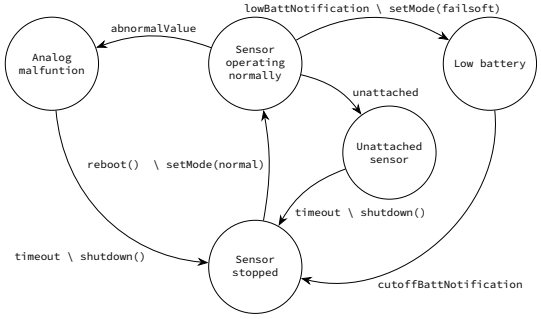

Fig. 4. Non-functional LTS of the Heart Rate sensor

4.3 Rule Based Modelization of Control Objectives and SLA

Since our self-adaptive approach is based on the DYNAMICO reference model, we express the control objectives in terms of rules. Rule-based definitions of objectives is declarative and easily express desired control objectives by end-users. The discrete controller (see Fig. 1) will be provided with rule-based control objectives to decide whether non-functional properties are guaranteed. Please note that SLA non-functional properties are considered to be control objectives and they are provided as inputs to the self-adaptive system. A rule refers to a condition-assertion statement and is defined as follows:

```
IF condition ASSERT action
```

Fig. 5 briefly illustrates couple of rules for safety non-functional property. The goal of these rules is to make sure that if either of the cardiac sensor is not operating normally or the EDA sensor is shut down, the position sensor is turned on to allow a better health status estimation.

```
R1: IF Not(HR.state == normal) ASSERT Pos.state == normal
R2: IF EDA.state == shutdown ASSERT Pos.state == normal
                    …
Rn: IF HR.Battery == Low ASSERT HR.setMode(FailSoft)
```

Fig. 5. Control objectives for a resilient cardiac monitoring

5 Synthesizing the Discrete Controller

The labeled transition system description of each sensor in terms of functional and nonfunctional properties leads to a set of distributed systems that run concurrently. In this context, these descriptions are equivalent to BZR control contracts, that are composed of three elements: an *assumption* (keyword `assume`), an *enforcement* (keyword `enforce`) and a declaration of controllable variables (keyword `with`). Reader may refer to [14] for a complete description the Heptagon/BZR language. In our work, we propose to generate the discrete controller by firstly mapping label transition descriptions of all sensors into a Heptagon/BZR programs and secondly synthesizing them into a discrete controller. Indeed, input of sensor services can be seen as controllable variables where rules can easily be converted into first-order logic enforcements. Because Heptagon is a synchronous programming language, the synchrony of our target IoT-system must be clearly defined. In our context, the synchrony hypothesis states: *given a system, a set of inputs and a set of outputs, the system must be able to compute all of its outputs between two occurrences of inputs changes events* [19].

Considering our use-case where computational abilities of gateways (used as centralized controllers) are much greater than computational resources of sensors, the IoT target system tends to behave as a synchronous system. Indeed, because of their gateway processing speeds, controllers running on gateways will be able to process all sensor events before receiving new events. Please note that while this kind of behavior is true for smaller IoT systems such as our wearable and ambient system (made of tens of

devices), this assumption might not hold for much bigger IoT-based systems made of thousands of devices, considering the mass of generated events is much higher.

6 Implementation and simulation results

As a preliminary simulation, the LTS of the cardiac parameters sensor introduced in Fig. 4 was encoded into BZR along with the LTS of both the position and EDA sensors. A rule set, including rules introduced in Fig. 5, were translated into BZR contracts, and were then used to successfully synthesize a controller using SIGALI. The simulation was performed using the simulator included with the Heptagon/BZR compiler, which demonstrates a correct behavior of the controlled IoT-system with respect to the provided control objectives. Fig. 6 is a chronogram presenting the behavior of our controlled system in terms of normal states. For example, we can easily see that both rule r_1 and r_2 are respected. In fact, when the cardiac sensor leaves the normal mode (i.e. it goes into any of other failure or shutdown states), the position sensor is turned on. Similarly, when the EDA sensor leaves the normal state (i.e. it is turned off, since this sensor is modeled as a binary-state sensor), the position sensor stays on.

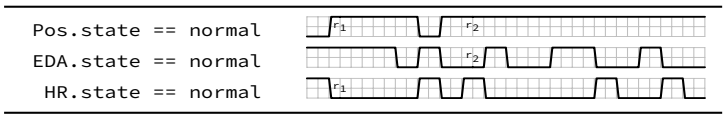

Fig. 6. Excerpt of the simulation chronograms

7 Conclusion and Perspectives

In this paper, we introduce a QoS-driven approach to self-adaptive IoT systems based on the DYNAMICO adaptation reference model and non-functional properties. Our IoT-based system relies on a set of rules and labeled state transitions to generate a discrete controller synthesis. Current ongoing work focuses on better sensor modeling by decoupling functional and non-functional behavior into distinct but interacting LTS to have better separation of concerns (non-functional LTS typically appearing in the monitoring feedback loop, while the functional LTS represents system functional adaptation). Integration of a domain specific rule-based language is also under investigation. This will enable us to express control objectives and thus improve separation of concerns between objectives feedback loop and the adaptation feedback loop.

Acknowledgement
Funding for this project was provided by a grant from la Région Rhône-Alpes.

References

1. Zhao, M., Privat, G., Rutten, É., Alla, H.: Discrete Control for the Internet of Things and Smart Environments. In: 8th International Workshop on Feedback Computing, San Jose, CA, USA, June 25, 2013 (2013).

2. Zhao, M., Privat, G., Rutten, E., Alla, H.: Discrete Control for Smart Environments Through a Generic Finite-State-Models-Based Infrastructure. In: Aarts, E. *et. al.* (eds.) Ambient Intelligence. pp. 174–190. (2014).
3. Villegas, N.M., Tamura, G., Müller, H.A., Duchien, L., Casallas, R.: DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems. In: Software Engineering for Self-Adaptive Systems II. pp. 265–293. (2013).
4. Weyns, D., Malek, S., Andersson, J.: FORMS: a formal reference model for self-adaptation. In: Proceedings of the 7th International Conference on Autonomic Computing. pp. 205–214. (2010).
5. Villegas, N.M., Müller, H.A., Tamura, G., Duchien, L., Casallas, R.: A framework for evaluating quality-driven self-adaptive software systems. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 80–89. (2011).
6. Bonino, D., Corno, F.: Rule-based intelligence for domotic environments. Automation in Construction. pp. 183–196 (2010).
7. Yuan, B., Herbert, J.: Context-aware hybrid reasoning framework for pervasive healthcare. Personal and Ubiquitous Computing. pp. 865–881 (2014).
8. Augusto, J.C., McCullagh, P., McClelland, V., Walkden, J.A.: Enhanced healthcare provision through assisted decision-making in a smart home environment. In: 2nd Workshop on Artificial Inteligence Techniques for Ambient Inteligence (2007).
9. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer. pp. 41–50 (2003).
10. Angelopoulos, K., Papadopoulos, A.V., Silva Souza, V.E., Mylopoulos, J.: Model predictive control for software systems with CobRA. In: Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 35–46. (2016).
11. Peng, X., Chen, B., Yu, Y., Zhao, W.: Self-tuning of software systems through dynamic quality tradeoff and value-based feedback control loop. Journal of Systems and Software. pp. 2707–2719 (2012).
12. Arcaini, P., Riccobene, E., Scandurra, P.: Modeling and analyzing MAPE-K feedback loops for self-adaptation. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 13–23. (2015).
13. Marchand, H., Bournai, P., Borgne, M.L., Guernic, P.L.: Synthesis of Discrete-Event Controllers based on the Signal Environment. Discrete Event Dynamic Systems. pp. 325–346 (2000).
14. Delaval, G., Marchand, H., Rutten, E.: Contracts for modular discrete controller synthesis. In: ACM Sigplan Notices. pp. 57–66. (2010).
15. Delaval, G., Rutten, E., Marchand, H.: Integrating discrete controller synthesis into a reactive programming language compiler. Discrete Event Dynamic Systems. pp. 385–418 (2013).
16. Cano, J., Delaval, G., Rutten, E.: Coordination of ECA Rules by Verification and Control. In: Kühn, E. and Pugliese, R. (eds.) Coordination Models and Languages. pp. 33–48. (2014).
17. Cano, J., Rutten, E., Delaval, G., Benazzouz, Y., Gurgen, L.: ECA Rules for IoT Environment: A Case Study in Safe Design. In: Proceedings of the 8th International Conference on Self-Adaptive and Self-Organizing Systems Workshops. pp. 116–121. (2014)
18. Abid, R., Salaün, G., De Palma, N.: Asynchronous synthesis techniques for coordinating autonomic managers in the cloud. Science of Computer Programming. pp. 87–103 (2017).
19. Gamatié, A.: Synchronous Programming: Overview. In: Designing Embedded Systems with the SIGNAL Programming Language. pp. 21–39. (2010).