# Bringing Runtime Verification Home ⋆

Antoine El-Hokayem and Yliès Falcone

Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble, France
firstname.lastname@univ-grenoble-alpes.fr

**Abstract.** We use runtime verification (RV) to check various specifications in a smart apartment. The specifications can be broken down into three types: behavioral correctness of the apartment sensors, detection of specific user activities (known as activities of daily living), and composition of specifications of the previous types. The context of the smart apartment provides us with a complex system with a large number of components with two different hierarchies to group specifications and sensors: geographically within the same room, floor or globally in the apartment, and logically following the different types of specifications. We leverage a recent approach to decentralized RV of decentralized specifications, where monitors have their own specifications and communicate together to verify more general specifications. This allows us to re-use specifications, and combine them to: (1) scale beyond existing centralized RV techniques, and (2) greatly reduce computation and communication costs.

Sensors and actuators are used to create "smart" environments which track the data across sensors and human-machine interaction. One particular area of interest consists of homes (or apartments) equipped with a myriad of sensors and actuators, called *smart homes* [11]. Smart homes are capable of providing added services to users. These services rely on detecting the user behavior and the context of such activities [7], typically detecting activities of daily living (ADL) [29,9] from sensor information. Detecting ADL allows to optimize resource consumption (such as electricity [1]), improve the quality of life for the elderly [27] and users suffering from mild impairment [30].

Relying on information from multiple sources and observing behavior is not just constrained to activities. It is also used with techniques that verify the correct behavior of systems. *Runtime Verification* (RV) [20,5,3,4] is a lightweight formal method which consists in verifying that a run of a system is correct wrt a specification. The specification formalizes the behavior of the system typically in logics (such as variants of Linear Temporal Logic, LTL) or finite-state machines. Based on the provided specification, monitors are automatically synthesized to run alongside the system and verify whether or not the system execution complies with the specification. RV techniques have been used for instance in the context of automotive [10] and medical [26] systems. In both cases, RV is used to verify communication patterns between components and their adherence to the architecture and their formal specifications.

⋆ This work is supported by the French national program "Programme Investissements d'Avenir IRT Nanoelec" (ANR-10-AIRT-05). The authors thank the Amiqual4Home (ANR-11-EQPX-0002) team, in particular S. Borkowski and J. Crowley for assisting in the case study and J. Cumin, for providing the collected data. This article is based upon work from COST Action ARVI IC1402, supported by the European Cooperation in Science and Technology.

While RV can be used to check that the devices in a smart home are performing as expected, we believe it can be extended to monitor ADL, and complex behavior on the activities themselves. We identify three classes of specifications for applying RV to a smart home. The first class pertains to the system behavior. These specifications are used to check the correct behavior of the sensors, and detect faulty sensors. Ensuring that the system is behaving correctly is what is generally checked when performing RV. However, it is also possible to use RV to verify other specifications. The second class consists of specifications for detecting ADL, such as detecting when the user is cooking, showering or sleeping. The third class contains combinations of the other two. These specifications can be seen as meta-specifications for both system correctness and ADL, they can include specifications such as ensuring that the user does not sleep while cooking, or ensuring that certain activities are only done under certain conditions.

However, standard RV techniques are not directly suitable to monitor the three classes of specifications. This is mainly due to scalability issues arising from the large number of sensors, as typically RV techniques rely on a single large formula to describe all behavior. Synthesizing centralized monitors from certain large formulas considered in this paper is not possible using the current tools. Instead, we make use of RV with decentralized specifications [16], as it allows monitors to reference other monitors in a hierarchical fashion. The advantage of this is twofold. First, it provides an abstraction layer to relate specifications to each others. This allows specifications to be organized and changed without affecting other specifications, and even to be expressed with different specification languages. Second, it leverages the structure and layout of the devices to organize the hierarchies. On the one hand, we have a geographical hierarchy resulting from the spacial structure of the apartment from a given device, to a room, a floor, or the full apartment. On the other hand, we have a logical hierarchy defined by the interdependence between specifications, i.e. ADL specifications that use other ADL specifications, and specifications that combine sensor safety with ADL. For example, informally, consider checking two activities: sleeping and cooking, which can be expressed using formulae $\varphi_{\mathtt{s}}$ and $\varphi_{\mathtt{c}}$ respectively. A monitor that checks whether the user is sleeping and cooking requires to check $\varphi_{\mathtt{s}} \wedge \varphi_{\mathtt{c}}$ and as such will replicate the monitoring logic of another monitor that checks $\varphi_{\mathtt{s}}$ alone, instead of re-using the output of that monitor. The formula will be written twice, and changing the formula for detecting sleeping requires changing the formula for the monitor that checks both specifications.

Overall, we see our contributions as follows[1]:

- Applying decentralized RV to analyze traces of over 36,000 timestamps spanning 27 sensors in a real smart apartment (Sect. 1.1).
- Going beyond system properties, to specify ADL using RV, and more complex inter-dependent specifications defined on up to 27 atomic propositions (Sect. 1.2).
- Taking advantage of hierarchies, modularity and re-use of decentralized specifications (Sect. 2) to both be able to synthesize monitors and to reduce overhead when monitoring complex inter-dependent specifications (Sect. 3.1).
- Using RV to effectively monitor ADL and identifying some insights and limitations inherent to using formal LTL specifications to determine user behavior (Sect. 3.2).

[1] An artifact [15] that contains data, documentation, and software, is provided to replicate and extend on the work. An extended version of this paper is available in [18].

# 1 Writing Specifications for the Apartment

## 1.1 Devices and Organization

We consider a single actual apartment, with multiple rooms, where activities are logged using sensors. Amiqual4Home is an experimental platform consisting of a smart apartment, a rapid prototyping platform, and tools for observing human activity.

**Overview of Amiqual4Home.** The Amiqual4Home apartment is equipped with 219 sensors and actuators spread across 2 floors [25]. Amiqual4Home uses the OpenHab 6 integration platform for all the sensors and actuators installed. Sensors communicate using KNX, MQQT or UPnP protocols sending measurements to OpenHab over the local network, so as to preserve privacy. The general layout of the apartment consists of 2 floors: the ground and first floor. On the ground floor (resp. first floor), we have the following rooms: `entrance`, `toilet`, `kitchen`, and `livingroom` (resp. `office`, `bedroom`, and `bathroom`). Between the two floors, there is a connecting `staircase`. This layout reveals a geographical hierarchy of components, where we can see the rooms at the leaves, grouped by floors then the whole apartment. While in effect all device data is fed to a central observation point, it is reasonable to consider the hierarchy in the apartment as a simpler model to consider hierarchies in general, as one is bound to encounter a hierarchy at a higher level (from houses, to neighborhoods, to smart cities, etc.). Furthermore, hierarchies appear when integrating different providers for devices in the same house.

**Reusing the Orange4Home dataset.** Amiqual4Home has been used to generate multiple datasets that record all sensor data, this includes an ADL recognition dataset [25] (ContextAct@A4H), and an energy consumption dataset [12] (Orange4Home). In this paper, we reuse the dataset from [12]. The case study involved a person living in the home and following (loosely) a schedule of activities spread out across the various rooms of the house, set out by the authors. This allows us to nicely reconstruct the schedule from the result of monitoring the sensors. Furthermore, the person living in the home provided manual annotations of the activities done, which helps us assess our specifications. We chose to use [12] over [25] as it involves only one person living in the house at a time which simplifies writing and validating specifications.

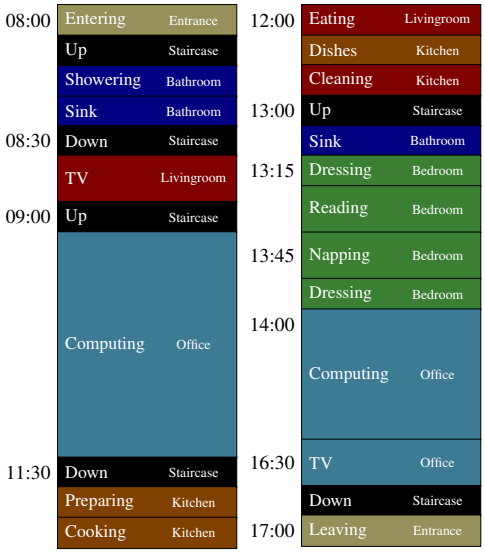



Fig. 1: Schedule for Jan 31 2017

Table 1: Specifications considered in this paper. (*) indicates added ADL specifications. G indicates specification group: system (S), ADL (A), and meta-specifications (M). $|\mathrm{AP}|^{\mathrm{d}}$ (resp. ($|\mathrm{AP}|^{\mathrm{c}}$): atomic propositions needed in formula for decentralized (resp. centralized) specifications. $\mathrm{d}$ is the maximum depth of monitor dependencies.

| G | Scope | Name | Description | $\lvert\mathrm{AP}\rvert^{\mathrm{d}}$ | $\lvert\mathrm{AP}\rvert^{\mathrm{c}}$ | $\mathrm{d}$ |
|---|---|---|---|---|---|---|
| S | Room | `sc_light`$(i)$ | light switch turns on light ($i \in [0..3]$). | 2 | 2 | 1 |
| M | House | `sc_ok` | All light switches are ok. | 4 | 8 | 2 |
| A | Toilet | `toilet`* | Toilet is being used. | 1 | 1 | 0 |
| A | Bathroom | `sink_usage` | Sink is being used. | 1 | 2 | 1 |
| A | Bathroom | `shower_usage` | Shower is being used. | 1 | 2 | 1 |
| A | Bedroom | `napping` | Tenant is sleeping on the bed. | 1 | 1 | 1 |
| A | Bedroom | `dressing` | Tenant is dressing, using the closet. | 2 | 3 | 1 |
| A | Bedroom | `reading` | Tenant is reading. | 3 | 5 | 2 |
| A | Office | `office_tv` | Tenant is watching TV. | 1 | 1 | 1 |
| A | Office | `computing` | Tenant is using the computer. | 1 | 1 | 1 |
| A | Kitchen | `cooking` | Tenant is cooking food. | 2 | 2 | 1 |
| A | Kitchen | `washing_dishes` | Tenant is cleaning dishes. | 2 | 3 | 1 |
| A | Kitchen | `kactivity`* | Using cupboards and fridge. | 4 | 9 | 1 |
| A | Kitchen | `preparing` | Tenant is preparing to cook food. | 2 | 11 | 2 |
| A | Living | `livingroom_tv` | Tenant is watching TV. | 2 | 2 | 1 |
| A | Floor 0 | `eating` | Tenant is eating on the table. | 2 | 2 | 1 |
| M | Floor 0 | `actfloor`$(0)$ | Activity triggered on floor 0. | 6 | 16 | 3 |
| M | Floor 1 | `actfloor`$(1)$ | Activity triggered on floor 1. | 7 | 11 | 3 |
| M | House | `acthouse` | Activity triggered in house | 2 | 27 | 4 |
| M | House | `notwopeople` | No 2 simultaneous activities on dif. floors. | 2 | 27 | 4 |
| M | House | `restricttv` | No watching TV for more than 10s. | 2 | 3 | 3 |
| M | House | `firehazard` | No cooking while sleeping. | 2 | 3 | 2 |

**Monitoring environment.** In total, we formalize 22 specifications that make use of up to 27 sensors, and evaluate them over the course of a full day of activity in the apartment [2]. That is, we monitor the house (by replaying the trace) from 07:30 to 17:30 on a given day, by polling the sensors every 1 second, creating a trace of a total of 36,000 timestamps. Specifications are elaborated in Sect. 1.2 and expressed as decentralized specifications [16] (introduced in Sect. 2.2). Traces are replayed using the `THEMIS` tool [17] which supports decentralized specifications and provides a wide range of metrics. We elaborate on the trace replay in Sect. 2.4.

### 1.2 Specification Groups

We now specify specifications that describe different behaviors of components in the smart apartment. Specifications can be subdivided into 3 groups: system-behavior properties, user-behavior specifications, and meta-specifications on both system and user behavior. The specifications we considered are listed in Table 1.

[2] [19] is a more detailed version of this paper including all the specifications.

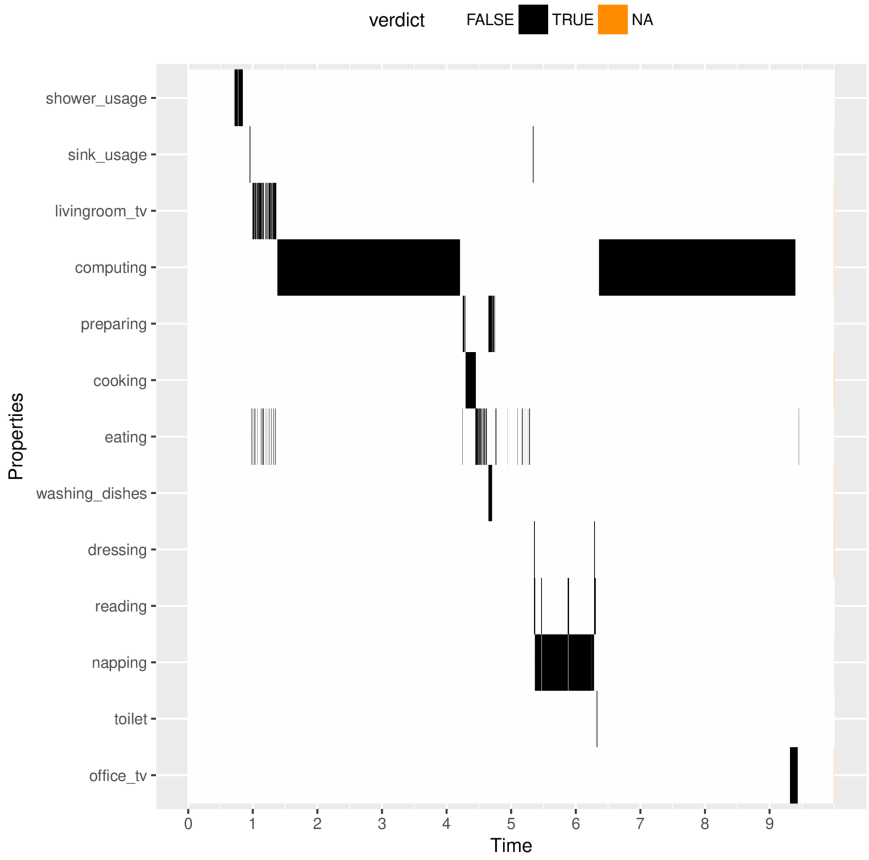



Fig. 2: Detected ADL for Tuesday, Jan 31 2017. Time is in hours starting from 7:30.

**System behavior.** The first group of specifications consists in ensuring that the system behaves as expected. That is, verifying that the sensors are working properly. These specifications are the subject of classical RV techniques [20,6] applied to systems. For the scope of this case study, we verify light switches as system properties. We verify that for a given room $i$, whenever the switch is toggled, then the light must turn on until the switch is turned off. We verify the property at two scopes, for a given room, and the entire apartment. While this property appears simple to check, it does highlight issues with existing centralized techniques applied in a hierarchical way. We develop the property in Sect. 2.1, and show the issues in Sect. 2.2.

**ADL.** The second group of specifications is concerned with defining the behavior of the user inferred from sensors. The sensors available in the apartment provide us with a wealth of information to determine the user activities. The list of activities of interest is detailed in [24] and includes activities such as cooking and sleeping. By correctly identifying activities, it is possible to decide when to interact with the user in a smart setting [1], provide custom care such as nursing for the elderly [27], or help users who suffer from mild impairment [30]. Inferring activities done by the user is an

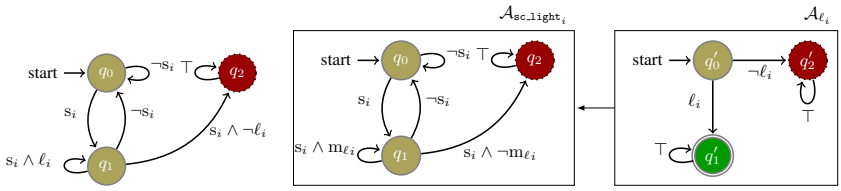
interesting problem typically addressed through either data-based or knowledge-based methods [9]. The first method consists in learning activity models from preexisting large-scale datasets of users behaviors by utilizing data mining and machine learning techniques. The built models are probabilistic or statistical activity models such as Hidden Markov Model (HMM) or Bayesian networks, followed by training and learning processes. Data-driven approaches are capable of handling uncertainty, while often requiring large annotated datasets for training and learning. The second method consists in exploiting prior knowledge in the domain of interest to construct activity models directly using formal logical reasoning, formal models, and representation. Knowledge-driven approaches are semantically clear, but are typically poor at handling uncertainty and temporal information [9]. We elaborate on such limitations in Sect. 3.2. Writing specifications can be seen as a knowledge-based approach to describe the behavior of sensors. As such, we believe that runtime verification is useful to describe the activity as a specification on sensor output. We formalize a specification for the following ADL activities described in [12] (see Table 1). We re-use the traces to verify that our detected activities are indeed in line with the schedule proposed. Figure 2 displays the reconstructed schedule after detecting ADL with runtime verification. Each specification is represented by a monitor that outputs (with some delay) for every timestamp (second) verdicts $\top$ or $\bot$. To do this, the monitor finds the verdict for a timestamp $t$ then respawns to monitor $t+1$. Verdict $\top$ indicates that the formula holds, that is, the activity is being performed. The reconstructed schedule shows the eventual outcome of a specification for a given timestamp ignoring delay. In reality, some delay happens based on the specification itself, and the dependencies on other monitors.

**Meta-specifications.** Specifications of the last group are defined on top of the other specifications. That is, we refer to a meta-specification as a specification that defines the interactions between various other specifications. While one can easily define specifications by defining predicates over existing ones, such as checking that the light switch property holds in all rooms or whether or not detecting an activity was performed on a specific floor or globally in the house, we are interested more in specifications that relate to each other. We consider a meta-specification that reduces fire hazards in the house. In this case, we specify that the tenant should not cook and sleep at the same time, as this increases the risk of fire. In addition to mutually excluding specifications, we can also constrain the behavior of existing specifications. For example, we can write a specification regulating the duration of watching TV to be at most 10 timestamps.

## 2 Monitoring the Appartment

### 2.1 Monitor Implementation

To monitor the apartment, we use LTL3 monitors [6]. An LTL3 monitor is a complete and deterministic Moore automaton where states are labeled with the verdicts $\mathbb{B}_3 = \{\top, \bot, ?\}$. Verdicts $\top$ and $\bot$ respectively indicate that the current execution complies and does not comply with the specification, while verdict $?$ indicates that the verdict has not been determined yet. Verdicts $\top$ and $\bot$ are called final, as once the monitor outputs $\top$ or $\bot$ for a given trace, it cannot output a different verdict for any suffix of that trace. Using LTL3 monitors for representing our specificaitons allows us to take advantage of the multiple RV tools that convert different specification languages to LTL3 monitors.




(a) Centralized (b) Decentralized

Fig. 3: Monitor(s) for $\texttt{sc\_light}(i)$, for a given room $i$ in the house. The verdicts associated with the states are $\bot$: dotted red , $\top$: double green, and ?: single yellow.

For our monitoring, we use the `THEMIS` tool which is able to use both `ltl2mon` [6] and `LamaConv` [22] to generate monitors.

*Example 1 (Check light switch).* Let us consider property $\texttt{sc\_light}(i)$ (sensor check light): "Whenever a light switch is triggered in a room $i$ at some timestamp $t$, then the light must turn on at $t+1$ until the switch is turned off again". Figure 3a shows the Moore automaton that represents the property. Starting from $q_0$ with verdict ?, the automaton verifies that the property is falsified (as it is a safety property). That is, upon reaching $q_2$ the verdict will be $\bot$ for all possible suffixes of a trace.

For the scope of this paper and for clarity, we use LTL extended with two (syntactic) operators, mostly to strengthen and relax time constraints. We consider the operator *eventually within* $t$ ($\Diamond_{\leq t}$) which considers a disjunction of next operators. It is defined as: $\Diamond_{\leq t} ap \stackrel{\text{def}}{=} ap \vee \mathrm{X}ap \vee \mathrm{XX}ap \vee ... \mathrm{X}^t ap$. Where $ap$ is an atomic proposition. Intuitively, the eventually within states that $ap$ holds within a given number of timestamps. Operator $\Diamond_{\leq t}$ allows us to relax the time constraints for a given atomic proposition. Similarly, we consider the operator *globally within* $t$ ($\Box_{\leq t}$) which the dual of the previous operator. The operator $\Box_{\leq t}$ is a conjunction of next operators. $\Box_{\leq t} ap \stackrel{\text{def}}{=} ap \wedge \mathrm{X}ap \wedge \mathrm{XX}ap \wedge \mathrm{X}^t ap$.

*Example 2 (Check light switch modalities).* The property expressed in Example 1 can be expressed in LTL as: $\texttt{sc\_light}(i) \stackrel{\text{def}}{=} \Box(\mathrm{s}_i \implies \mathrm{X}(\ell_i \,\mathrm{U}\, \neg \mathrm{s}_i))$. The property can be modified with the extra operators relax or constrain the time on the light. The relaxed property $\texttt{sc\_light}'(i) \stackrel{\text{def}}{=} \Box(\mathrm{s}_i \implies \Diamond_{\leq 3}(\ell_i \,\mathrm{U}\, \neg \mathrm{s}_i))$ allows the right-hand side of the implication to hold within any of the next 3 timestamps instead of immediately after. The bounded property $\texttt{sc\_light}''(i) \stackrel{\text{def}}{=} \Box(\mathrm{s}_i \implies \Box_{\leq 3}(\ell_i))$ states that the light is on starting from the timestamp the switch is turned on and the subsequent two (for a total of 3). An example of such a property is the restriction on watching TV for a specific duration (Table 1) where $\texttt{restricttv} \stackrel{\text{def}}{=} \Box(\texttt{tv} \implies \Diamond_{\leq 10} \neg \texttt{tv})$.

### 2.2 Decentralized Specifications

While simple specifications can be expressed with both LTL and automata, it quickly becomes a problem to scale the formulae or account for hierarchies (see Sect. 2.3). As such, we use decentralized specifications [16].

Informally, a decentralized specification considers the system as a set of components, defines a set of monitors, additional atomic propositions that represent references

to monitors, and attaches each monitor to a component. A decentralized trace is a partial function that assigns to each component and timestamp an event. Each monitor is a Moore automaton as described in Sec. 2.1 where the transition label is restricted to only atomic propositions related to the component on which the monitor is attached, and references to other monitors. A monitor reference is evaluated as if it were an oracle. That is, to evaluate a monitor reference $\mathrm{m}_i$ at a timestamp $t$, the monitor referenced ($\mathcal{A}_i$) is executed starting from the initial state on the trace starting at $t$. The atomic proposition $\mathrm{m}_i$ at $t$ takes the value of the final verdict reached by the monitor.

*Example 3 (Decentralized light switch).* Figure 3b shows the decentralized specification for the check light property from Example 1. We have two monitors $\mathcal{A}_{\texttt{sc\_light}_i}$ and $\mathcal{A}_{\ell_i}$. They are respectively attached to the light switch, and light bulb components. In the former, the atomic propositions are either related to observations on the component ($\mathrm{s}_i$, switch on), or references to other monitors ($\mathrm{m}_{\ell_i}$). The light switch monitor first waits for the switch to be on to reach $q_1$. In $q_1$, at some timestamp $t$, it needs to evaluate reference $\mathrm{m}_{\ell_i}$ by running the trace starting from $t$ on monitor $\mathcal{A}_{\ell_i}$.

The assumptions of decentralized specifications on the system are as follows: no monitors send messages that contain wrong information; no messages are lost, they are eventually delivered in their entirety but possibly out-of-order; all components share one logical discrete clock marked by round numbers indicating relevant transitions in the system specification. While security is a concern in the smart apartment setting, the first two assumptions are met in this case study as the apartment sensor network operates on the local network, and we expect monitors to be deployed by the sensor providers, and users of the apartment. The last assumption is also met in the smart setting, as all sensors share a global clock.

### 2.3 Advantages of Decentralized Specifications

**Modularity and re-use.** Monitor references in decentralized specifications allow specifications writers to modularize behavior. Given that a monitor represents a specific behavior, this same monitor can be re-used to define more complex specifications at a higher level, without consideration for the details needed for this specification. This allows specification writers to reason at various levels about the system specification.

Let us consider the ADL specification `cooking` (resp. `sleeping`) which specifies whether the tenant is cooking (resp. sleeping) in the apartment. One can reason about the meta-specification `firehazard` using both `cooking` and `sleeping` specifications without considering the lower level sensors that determine these specifications, that is $\texttt{firehazard} \stackrel{\text{def}}{=} \Box(\texttt{sleeping} \implies \neg\texttt{cooking})$. While we can define `cooking` as $\texttt{cooking} \stackrel{\text{def}}{=} \texttt{kitchen\_presence} \wedge \Diamond_{\leq 5}(\texttt{kitchen\_cooktop} \vee \texttt{kitchen\_oven})$. Additionally, any specification that requires either `sleeping` or `cooking` can re-use the verdict outputted by their respective monitors. For example the specifications $\texttt{actfloor}(0)$ and $\texttt{actfloor}(1)$ require the verdicts from monitors associated with `cooking` and `sleeping`, respectively, since cooking happens on the ground floor while sleeping on the first floor. Furthermore, we can disjoin $\texttt{actfloor}(0)$ and $\texttt{actfloor}(1)$ to easily specify that an activity has happened in the house, $\texttt{acthouse} \stackrel{\text{def}}{=} \texttt{actfloor}(0) \vee \texttt{actfloor}(1)$. While specification `acthouse` can be seen as a quantified version of

$\texttt{actfloor}(i)$, we can use modular specifications for behavior, for example we can verify the triggering of an alarm in the house within 5 timestamps of detecting a fire hazard, i.e. $\texttt{checkalert} \stackrel{\text{def}}{=} \texttt{firehazard} \implies \Diamond_{\leq 5}(\texttt{firealert})$.

In addition to providing a higher level of abstraction and reasoning about specifications, the modular structure of the specifications present three additional advantages. The first allows the sub-specifications to change without affecting the meta-specifications, that is if the sub-specification `cooking` is changed (possibly to account for different sensors), no changes need to be propagated to specifications `firehazard`, $\texttt{actfloor}(0)$, `acthouse`, and `checkalert`. The second advantage is controlling duplication of computation and communication, as such sensors do not have to send their observations constantly to all monitors that verify the various specifications. The specification `cooking` requires knowledge from the kitchen presence sensor, the kitchen cooktop (being enabled) and the kitchen oven. Without any re-use these three sensors (presence, cooktop, and oven) need to send their information to monitors checking: `firehazard`, $\texttt{actfloor}(0)$, `acthouse`, and `checkalert`. The third advantage is a consequence of modeling explicitly the dependencies between specifications. This allows the monitoring to take advantage of such dependencies and place the monitors that depend on each other closer depending on the hierarchy, either geographically (i.e., in the same room or floor) or logically (i.e., close to the monitors of the dependent sub-specifications). Furthermore, knowing the explicit dependencies between specifications allows the user to choose a placement for their monitors, adjusting the placement to the system architecture. In the case a placement is not possible, it is possible to create intermediate specifications that simply relay verdicts of other monitors, to transitively connect all components that are not connected.

**Abstraction from implementation.** Decentralized specifications define modular specifications that can be composed together to form bigger and more complex specifications. One setback for learning-based techniques to detect ADL is their specificity to the environment. That is, the training set is specific to a house layout, user profile (i.e., elderly versus adults) [23].

By using references to monitors, we leave the implementation of the specification to be specific for the house or user profile. Using our existing example, `cooking` is implemented based on the available sensors in the house, which would change for different houses. However, meta-specifications such as `firehazard` can be defined independently from the implementation of both `cooking` and `sleeping`.

Furthermore, using monitor references, which are treated as oracles, opens the door to utilizing existing techniques in the literature for non-automata based monitors. That is, as a reference is expected to eventually evaluate to $\top$ or $\bot$, any implementation of a monitor that can return a final verdict for a given timestamp can be incorporated to form more complex specifications. For example, one can use the various machine learning techniques [7,23,29] to define monitors that detect specific ADLs, then reference them in order to define more complex specifications.

**Scalability.** Decentralized specifications allow for a higher level of scalability when writing specifications, and also when monitoring. By using decentralized specifications, we restrict a given monitor to atomic propositions local to the component on which it is attached, and references to other monitors (see Sect. 2.2). This greatly reduces the

number of atomic propositions to consider when synthesizing the monitor and reduces its size, as the sub-specifications are offloaded to another monitor.

For example, let us consider writing specifications using LTL formulae. The classical algorithm that converts LTL to Moore automata is doubly exponential in the size of the formula including all permutations of atomic propositions (to form events) [6]. As such reducing both the size of the formula and the number of atomic propositions used in the formula helps significantly when synthesizing the monitors, allowing us to scale beyond the limits of existing tools. For a large formula, it becomes impossible to generate a central monitor using the existing synthesis techniques. Decentralized specifications provide a way to manage the large formula by subdividing it into subformulae. The decomposition ensures that the formula evaluates to the same verdict given the same observations, at the cost of added delay.

*Example 4 (Synthesizing check light).* Recall the system property $\texttt{sc\_light}(i)$ in Example 2 responsible for verifying that in a room $i$ a light switch does indeed turn a light bulb on until it is turned off. We recall the LTL specification $\texttt{sc\_light}(i) \stackrel{\text{def}}{=} \square(\mathrm{s}_i \implies \mathrm{X}(\ell_i \,\mathrm{U}\, \neg \mathrm{s}_i))$. To verify the property across $n$ rooms of the house, we formulate a property $\texttt{sc\_ok} \stackrel{\text{def}}{=} \bigwedge_{i\in[0..n]} \texttt{sc\_light}(i)$. In the case of a decentralized specification the formula will reference each monitor in each room, leading to a conjunction of at $n$ atomic propositions. However, in the case of a centralized specification, the specification needs to be written as: $\texttt{sc\_ok}^{\text{cent}} \stackrel{\text{def}}{=} \bigwedge_{i\in[0...n]} \square(\mathrm{s}_i \implies \mathrm{X}(\ell_i \,\mathrm{U}\, \neg \mathrm{s}_i))$, which is significantly more complex as a formula consisting of $4n$ operators (to cover the sub-specification), along $n$ conjunctions, and defined over each sensor and light bulb atomic propositions ($2n$). Given that monitor synthesis is doubly exponential, both `ltl2mon` [6] and `lamaconv` [22] require significant resources and time to generate the minimal Moore automaton (in our case we were unable to generate the monitor for $n = 3$ after an hour to timeout with both tools).

### 2.4 Trace Replay with `THEMIS`

To perform monitoring we use `THEMIS` [17]. `THEMIS` is designed to define and handle decentralized specifications. The trace from [12] is given as a database with a table for each sensor. We extract each table as a `csv` file for each sensor and treat them as observations, we then assign a logical component for multiple related sensors.

Each sensor is implemented as an input (`Periphery` in `THEMIS`) to a logical component. For example, for the shower water, we use both cold and hot water sensors but define only a single component ("shower water"), from an RV perspective, "hot" and "cold" are multiple observations passed to the "shower water" component. We implemented two peripheries to process sensor trace data: `SensorBool` and `SensorThresh`. The first periphery reads Boolean values from the `csv` file associated with timestamps, and associates them with an atomic proposition. The second periphery reads real (double) values, converts them Boolean values based on whether the number is below or above a certain threshold, and associates them with an atomic proposition.

Since the system has a global clock, to synchronize observations, our periphery implementations synchronize on a date at the start and an increment (in our case 1 second) and a default Boolean value for the observation. When polled, the periphery returns the default value if nothing is observed yet, or the last value observed otherwise.

Managing the trace length (36,000) is an issue for the monitoring techniques presented in [16] as they rely on eventual consistency and will wait on input for the length of the trace, which requires a lot of memory. We optimized the datastructure used to store observations (Memory) to add garbage collection and thus reduce memory usage.

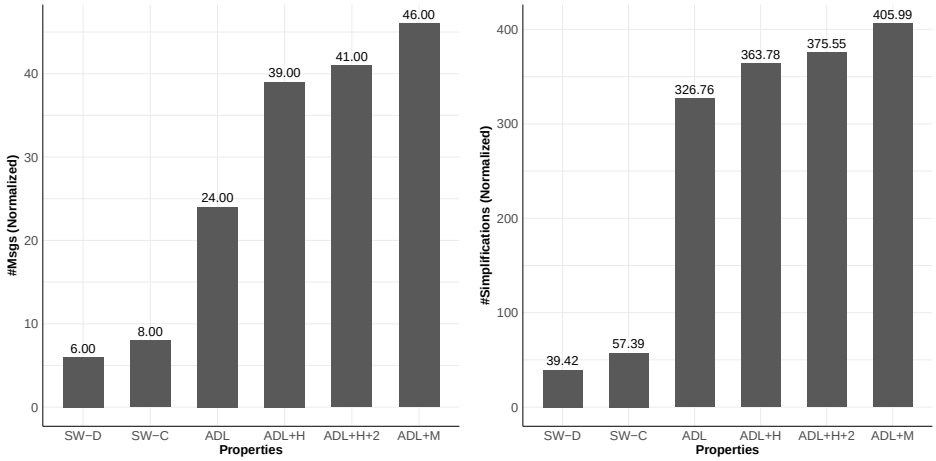
## 3 Assessing the Monitoring of the Appartment

Monitoring the smart apartment requires leveraging the interdependencies between specifications to be able to scale, beyond monitoring system properties, to more complex meta-specifications (as detailed in Sect. 1.2). We assess using decentralized specifications to monitor the apartment by conducting two separate scenarios. The first scenario (Sect. 3.1) evaluates the advantages of using decentralized specifications presented in Sect. 2.3 (modularity, scalability, and re-use) by looking at the complexity of monitor synthesis, and communication and computation costs when adding more complex specifications that re-use sub-specifications. The second scenario (Sect. 3.2) evaluates the effectiveness of detecting ADL by looking at various detection measures such as precision and recall.

### 3.1 Monitoring Efficiency and Hierarchies

**Monitor synthesis.** Table 1 displays the number of atomic propositions referenced by each specification for the decentralized ($|\mathrm{AP}^{\mathrm{d}}|$) and the centralized ($|\mathrm{AP}^{\mathrm{c}}|$) settings. Column $\mathrm{d}$ indicates the maximum depth of the dependencies directed acyclic graph, so as to assess how many levels of sub-specifications need to be computed. When $d = 0$, it indicates that the specification can be determined directly by the monitor placed on the component, while $d = 1$ indicates that the monitor has to pull at most 1 monitor (which typically relays the component observations). More generally, when $d = n$, it indicates that the specification depends on a monitor that has at most depth $n - 1$. The atomic propositions indicate either direct references to sensor observations (in the centralized setting) or references to either sensor observations and dependent monitors (in the decentralized setting). It is possible to notice that for certain specifications such as `toilet` which relies only on the water sensor in the toilet to be detected, there is no difference between using a centralized or decentralized specification, as it resolves to the observations. Reduction becomes more pronounced when specifications re-use other specifications. For example, specification $\texttt{acthouse} \stackrel{\text{def}}{=} \texttt{actfloor}(0) \vee \texttt{actfloor}(1)$, when decentralized, uses only 2 references (for each of the sub-specification). However, when expanded, it references all 27 sensors used to detect activities. Additionally, specification $\texttt{notwopeople} \stackrel{\text{def}}{=} \neg(\texttt{actfloor}(0) \wedge \texttt{actfloor}(1))$ does not re-use the sub-specifications and requires all sensors again. This greatly reduces the formula size and allows us to synthesize the monitors needed to check the formulae, as the synthesis algorithm is doubly exponential as mentioned in Sect. 2.3.

**Assessing re-use and scalability.** Reducing the size of the atomic propositions needed for a specification not only affects monitor synthesis, but also performance, as atomic propositions represent the information needed (Sect. 2.3). To assess re-use and scalability, we perform two tasks and gather two measures pertaining to computation and communication, and present results in Fig. 4. The first task compares a centralized (SW-C) and a decentralized (SW-D) version of property `sc_ok` presented in Example 4 using




(a) Communication (b) Computation

Fig. 4: Scalability of communication and computations in decentralized specifications.

only 2 rooms. The second task introduces large meta-specifications on top of the ADL specifications to check scalability. Firstly, we measure the communication and computation for monitoring ADL specifications (ADL). Secondly, we introduce specifications `actfloor`$(0)$, `actfloor`$(1)$ and `acthouse` (ADL+H) as they require information about all sensors for ADL. Thirdly, we add specification `notwopeople` (ADL+H+2), as it re-uses the same sub-specifications as specification `acthouse`. Lastly, we show all measures for all meta-specifications in Table 1 (ADL+M). We re-use two measures from [16]: the total number of simplifications the monitors are doing, and the total number of messages transferred. These measures are provided directly with `THEMIS` [17]. The total number of simplifications (**#Simplifications**) abstracts the computation done by the monitors, as they attempt to simplify Boolean expressions that represent automaton states, which are the basic operations for maintaining the monitoring data structures in [16]. The total number of messages abstracts the communication (**#Msgs**), as our messages are of fixed length, they also represent the total data transferred. Both measures are normalized by the number of timestamps in the execution (36,000).

**Results.** Figure 4a shows the normalized number of messages sent by all monitors. For the first task, we notice that the number of messages is indeed lower in the decentralized setting, SW-D sends on average 2 messages per timestamp less than SW-C, which corresponds to the difference in the number of atomic propositions referenced (6 for SW-D and 8 for SW-C). For the second task, we notice that on the baseline for ADL, we observe 24 messages per timestamp, a smaller number than the sensors count (27). This is because some ADL like `toilet` are directly evaluated on the sensor without communicating, and other ADL like `preparing`, re-use other ADL specifications like `kactivity`. By introducing the 3 meta-specifications stating that an activity occurred on a floor or globally in a house, the number of messages per round only increases by 15. This also coincides with the number of atomic propositions for the specifications (6 for `actfloor`$(0)$, 7 for `actfloor`$(1)$, and 2 for `acthouse`) as

Table 2: Precision, Recall, and F1 of monitoring Tuesday, Jan 31 2017.

(a) Monitoring all ADL specifications.

| Specification | Precision | Recall | F1 |
|---|---|---|---|
| computing | 0.98 | 0.99 | 0.99 |
| office_tv | 1.00 | 0.80 | 0.89 |
| cooking | 0.88 | 0.88 | 0.88 |
| shower_usage | 1.00 | 0.50 | 0.67 |
| washing_dishes | 1.00 | 0.47 | 0.64 |
| livingroom_tv | 1.00 | 0.43 | 0.60 |
| dressing | 1.00 | 0.41 | 0.58 |
| toilet* | 1.00 | 0.18 | 0.30 |
| sink_usage | 1.00 | 0.13 | 0.23 |
| eating | 0.61 | 0.35 | 0.44 |
| napping | 0.43 | 0.95 | 0.60 |
| preparing | 0.23 | 0.77 | 0.35 |
| reading | 0.37 | 0.04 | 0.06 |

(b) Variations on the napping specification.

| Formula | Precision | Recall | F1 |
|---|---|---|---|
| $\Box_{\leq 25}(\texttt{weight})$ | 0.43 | 0.95 | 0.60 |
| $\Box_{\leq 3}(\texttt{weight})$ | 0.43 | 0.99 | 0.60 |
| $\Diamond_{\leq 3}(\texttt{weight})$ | 0.43 | 1.0 | 0.60 |
| $\Box_{\leq 3}(\texttt{pres} \wedge \texttt{weight})$ | 0.34 | 0.14 | 0.20 |
| $\Box_{\leq 3}(\neg\ell \wedge \texttt{weight})$ | 1.00 | 0.97 | 0.99 |

weight: bed pressure sensor
pres : bedroom presence sensor
$\ell$ : bedroom light sensor

those monitors depend in total on 15 other monitors to relay their verdicts. This costs much less than polling 16 sensors to determine $\texttt{actfloor}(0)$, 11 sensors to determine $\texttt{actfloor}(1)$, and 27 (a total of 54) to determine acthouse. To verify this, we notice that the addition of notwopeople (ADL+H+2) that needs information from all 27 sensors, only increases the total number of messages per timestamp by 2. The specification notwopeople reuses the verdicts of the two monitors associated with each actfloor specification. After adding all the meta-specifications (ADL+M), the total number of messages per timestamp is 46, which is less than the number needed to verify adding actfloor, and acthouse in a centralized setting (54). We notice a similar effect for computation (Fig. 4b).

### 3.2 ADL Detection using RV

**Measurements.** Table 2a displays the effectiveness of using RV to monitor all ADL specifications on the trace of Tuesday, Jan 31 2017. To assess the effectiveness, we considered the provided self-annotated data from [12], where the user annotated the start and end of each activity. We measure precision, recall and F1 (the geometric mean of precision and recall). To measure precision, we consider a true positive when the verdict $\top$ of a monitor for a given timestamp fell indeed in the self-annotated interval for the activity. To measure recall, we measure the proportion of the intervals that have been determined $\top$ using RV. This approach is more fine-grained than the approach used in [25] where the precision and recall are computed for the start and end of intervals.

**Results.** The effectiveness of detection depends highly on the specification. Our approach performs well for the specifications computing, cooking, office_tv, as it exhibits high precision and high recall. The second group of specifications contains specifications such as shower_usage, and livingroom_tv. It exhibits high precision but medium recall, that is, we were able to determine around 40 to 50% of all the timestamps where the specifications held according to the person annotating, without any

false positives. The third group is similar to the second group but has very low recall (13-18%) and contains the specifications `toilet` and `sink_usage`. The fourth group, which includes the specifications `napping` and `preparing`, shows high recall but a high rate of false positives. And finally, specification `reading` is not properly detected, as it has a high rate of false positives and covers almost no annotated intervals.

**Limitations of RV for detecting ADL.** The limitations of using RV to detect ADL are due to the modeling. As mentioned in Sect. 1.2, RV can be seen as a knowledge-based approach to activity detection, as such it suffers from similar weaknesses and limitations [9]. The activity is described as a rigid formal specification over the sensor data, and this has two consequences. Firstly, since RV relies purely on sensor data, activities which cannot be inferred from existing sensors will be poorly detected or not detected at all. This is the case for `reading`, as there are no sensors to indicate that the tenant is reading. We infer reading by checking that the light is on in the room and no other specified activity holds. Secondly, given that specifications are rigid, we expect the user to behave exactly as specified for the activity to be detected, any minor deviation results in the activity not being detected. To illustrate this point, the specification `computing` relies on the power consumption of the plug in the office. Had the tenant been charging his phone instead of computing, the recall would have suffered greatly. Another great example of this is the `shower_usage` specification, that is captured by inspecting the water usage of the shower. The time the tenant spends getting into the shower and out of the shower will not be considered, which greatly impacts recall. Table 2b shows how we can modify the specification `napping` to attempt to better capture the activity. In this case, using the additional light sensor to detect lights are off, helps us increase precision. The above issues are further compounded by the annotation being carried out by a person. The annotator can for example take a few seconds to annotate some events which could impact recall, especially for short intervals of activity. However, even with the inherent limitations of using knowledge-based approaches, our observed groups and results fall within the expected range, of knowledge-based approaches such as [25], and also have similar effectiveness as model-based SVM approaches such as [8].

## 4 Related Work

We present similar or useful techniques for detecting ADL activities in a smart apartment that use log analysis and complex event processing. Then, we present techniques from stream-based RV that can be extended for monitoring smart apartments.

**ADL detection using log analysis.** Detecting ADL can be performed using trace analysis tools. The approach in [25] defines parametric events using Model Checking Language (MCL) [28] based on the modal mu-calculus (inspired by temporal logic and regular expressions). Traces are read and transformed into actions, then actions are matched against the specifications to determine locations in the trace that match ADL. Five ADL (sleep, using toilets, cooking, showering, and washing dishes) are specified and checked in the same smart apartment as our work. While this technique is able to detect ADL activities, it amounts to checking traces offline, and a high level of post-processing is required to analyze the data.

**ADL detection using Complex Event Processing.** Reasoning at a much higher level of abstraction than sensor data, the approach in [21] attempts to detect ADL by an-

alyzing the electrical consumption in the household. To do so, it employs techniques from Complex Event Processing (CEP), in which data is fed as streams and processed using various functions to finally output a stream of data. In this work, the ADL detection is split into two phases, one which detects peaks and plateaus of the various electrical devices, and the second phase uses those to indicate whether or not an appliance is being used. This illustrates a transformation from low-level data (sensor signal) to a high-level abstraction (an appliance is being used). The use of CEP for detecting ADL is promising, as it allows for similar scalability and abstraction. However, CEP's model of named streams makes it hard to analyze the specification formally, making little distinction between specification and implementation of the monitoring logic.

**ADL detection using Runtime Verification.** Similarly to CEP but focusing on Boolean verdicts, various stream-based RV techniques have been elaborated such as LOLA [13] which are used to verify correctness properties for synchronous systems such as the PCI bus protocol and a memory controller. A more recent approach uses the Temporal Stream-Based Specification Language (TeSSLa) to verify embedded systems using FPGAs [14]. Stream-based RV is particularly fast and effective for verifying lengthy parametric traces. However, it is unclear how these approaches handle monitor synthesis for a large number of components and account for the hierarchy in the system.

**Discussion.** Stream-based systems such as stream-based RV and CEP are bottom-up. Data in streams is eventually aggregated into more complex information and relayed to a higher level. Decentralized specifications also support top-down approaches, which would increase the efficiency of monitoring large and hierarchical systems. To illustrate the point, consider the decentralized specification in Fig. 3b. In the automaton $\mathcal{A}_{\texttt{sc\_light}_i}$, the evaluation of the dependent monitor $\mathcal{A}_{\ell_i}$ only occurs when reaching $q_1$, so long as the automaton is in $q_0$, no interaction with the dependent monitor is necessary. This top-down feedback can be used to naturally optimize dependencies and increase efficiency. Because of the the oracle-based implementation of decentralized specifications, it is possible to integrate any monitoring reference that eventually returns a verdict. One could imagine integrating other stream-based monitors or even data-driven ADL detection approaches. The integration works both ways, as monitors can be considered a (blocking) stream of verdicts for the other techniques.

## 5 Conclusion

Monitoring a smart apartment presents RV with interesting new problems as it requires a scalable approach that is compositional, dynamic, and able to handle a multitude of devices. This is due to the hierarchical structure imposed by either limited communication capabilities of devices across geographical areas or the dependencies between various specifications. Attempting to solve such problems with centralized specifications is met with several obstacles at the level of monitor synthesis techniques (as we are presented with large formulae), and also at the level of monitoring as one needs to model interdependencies between formulae and re-use the sub-specifications used to build more complex specifications. We illustrate how decentralized specifications tackle such systems by explicitly modeling of interdependencies between specifications. Furthermore, we illustrate monitoring specifications that detect ADL in addition to system properties and even more specifications defined over both types of specifications.

## References


1. Aimal, S., Parveez, K., Saba, A., Batool, S., Arshad, H., Javaid, N.: Energy optimization techniques for demand-side management in smart homes. In: Advances in Intelligent Networking and Collaborative Systems, The 9th International Conference on Intelligent Networking and Collaborative Systems, INCoS-2017. Lecture Notes on Data Engineering and Communications Technologies, vol. 8, pp. 515–524. Springer (2017)
2. Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, Santa Barbara, CA, USA, July 10 - 14, 2017. ACM (2017)
3. Bartocci, E., Falcone, Y. (eds.): Lectures on Runtime Verification - Introductory and Advanced Topics, Lecture Notes in Computer Science, vol. 10457. Springer (2018), https://doi.org/10.1007/978-3-319-75632-5
4. Bartocci, E., Falcone, Y., Bonakdarpour, B., Colombo, C., Decker, N., Havelund, K., Joshi, Y., Klaedtke, F., Milewicz, R., Reger, G., Rosu, G., Signoles, J., Thoma, D., Zalinescu, E., Zhang, Y.: First international competition on runtime verification: rules, benchmarks, tools, and final results of crv 2014. International Journal on Software Tools for Technology Transfer (Apr 2017)
5. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification - Introductory and Advanced Topics, Lecture Notes in Computer Science, vol. 10457, pp. 1–33. Springer (2018), https://doi.org/10.1007/978-3-319-75632-5_1
6. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. 20(4), 14 (2011)
7. Brdiczka, O., Crowley, J.L., Reignier, P.: Learning situation models in a smart home. IEEE Trans. Systems, Man, and Cybernetics, Part B 39(1), 56–63 (2009)
8. Chen, B., Fan, Z., Cao, F.: Activity recognition based on streaming sensor data for assisted living in smart homes. In: 2015 International Conference on Intelligent Environments, IE 2015. pp. 124–127. IEEE (2015)
9. Chen, L., Hoey, J., Nugent, C.D., Cook, D.J., Yu, Z.: Sensor-based activity recognition. IEEE Trans. Systems, Man, and Cybernetics, Part C 42(6), 790–808 (2012)
10. Cotard, S., Faucou, S., Béchennec, J., Queudet, A., Trinquet, Y.: A data flow monitoring service based on runtime verification for AUTOSAR. In: 14th IEEE International Conference on High Performance Computing and Communication & 9th IEEE International Conference on Embedded Software and Systems, HPCC-ICESS 2012. pp. 1508–1515. IEEE Computer Society (2012)
11. Crowley, J.L., Coutaz, J.: An ecological view of smart home technologies. In: De Ruyter, B., Kameas, A., Chatzimisios, P., Mavrommati, I. (eds.) Ambient Intelligence. pp. 1–16. Springer International Publishing, Cham (2015)
12. Cumin, J., Lefebvre, G., Ramparany, F., Crowley, J.L.: A dataset of routine daily activities in an instrumented home. In: Ubiquitous Computing and Ambient Intelligence - 11th International Conference, UCAmI 2017, Proceedings. Lecture Notes in Computer Science, vol. 10586, pp. 413–425. Springer (2017)
13. D'Angelo, B., Sankaranarayanan, S., Sánchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: LOLA: runtime monitoring of synchronous systems. In: 12th International Symposium on Temporal Representation and Reasoning (TIME 2005). pp. 166–174. IEEE Computer Society (2005)
14. Decker, N., Dreyer, B., Gottschling, P., Hochberger, C., Lange, A., Leucker, M., Scheffel, T., Wegener, S., Weiss, A.: Online analysis of debug trace data for embedded systems. In: 2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018. pp. 851–856. IEEE (2018)

15. El-Hokayem, A., Falcone, Y.: THEMIS Smart Home Artifact Repository, https://gitlab.inria.fr/monitoring/themis-rv18smarthome
16. El-Hokayem, A., Falcone, Y.: Monitoring decentralized specifications. In: Antoine El-Hokayem and Yliès Falcone [2], pp. 125–135
17. El-Hokayem, A., Falcone, Y.: THEMIS: a tool for decentralized monitoring algorithms. In: Antoine El-Hokayem and Yliès Falcone [2], pp. 372–375
18. El-Hokayem, A., Falcone, Y.: Bringing runtime verification home - A case study on the hierarchical monitoring of smart homes. CoRR abs/1808.05487 (2018), http://arxiv.org/abs/1808.05487
19. El-Hokayem, A., Falcone, Y.: Bringing Runtime Verification Home - A case study on the Hierarchical Monitoring of Smart Homes. CoRR abs/1808.05487 (2018)
20. Falcone, Y., Havelund, K., Reger, G.: A tutorial on runtime verification. In: Engineering Dependable Software Systems, NATO science for peace and security series, d: information and communication security, vol. 34, pp. 141–175. ios press (2013)
21. Hallé, S., Gaboury, S., Bouchard, B.: Activity recognition through complex event processing: First findings. In: Artificial Intelligence Applied to Assistive Technologies and Smart Environments, Papers from the 2016 AAAI Workshop. AAAI Workshops, vol. WS-16-01. AAAI Press (2016)
22. Institute for Software Engineering and Programming Languages: LamaConv - Logics and Automata Converter Library, http://www.isp.uni-luebeck.de/lamaconv
23. van Kasteren, T., Englebienne, G., Kröse, B.J.A.: Transferring knowledge of activity recognition across sensor networks. In: Pervasive Computing, 8th International Conference, Pervasive 2010. Proceedings. Lecture Notes in Computer Science, vol. 6030, pp. 283–300. Springer (2010)
24. Katz, S.: Assessing self-maintenance: Activities of daily living, mobility, and instrumental activities of daily living. Journal of the American Geriatrics Society 31(12), 721–727 (1983)
25. Lago, P., Lang, F., Roncancio, C., Jiménez-Guarín, C., Mateescu, R., Bonnefond, N.: The ContextAct@A4H real-life dataset of daily-living activities - activity recognition using model checking. In: Modeling and Using Context - 10th International and Interdisciplinary Conference, CONTEXT 2017, Proceedings. Lecture Notes in Computer Science, vol. 10257, pp. 175–188. Springer (2017)
26. Leucker, M., Schmitz, M., à Tellinghusen, D.: Runtime verification for interconnected medical devices. In: Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications - 7th International Symposium, ISoLA 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9953, pp. 380–387 (2016)
27. Majumder, S., Aghayi, E., Noferesti, M., Memarzadeh-Tehran, H., Mondal, T., Pang, Z., Deen, M.J.: Smart homes for elderly healthcare - recent advances and research challenges. Sensors 17(11), 2496 (2017)
28. Mateescu, R., Thivolle, D.: A model checking language for concurrent value-passing systems. In: FM 2008: Formal Methods, 15th International Symposium on Formal Methods, Proceedings. Lecture Notes in Computer Science, vol. 5014, pp. 148–164. Springer (2008)
29. Tapia, E.M., Intille, S.S., Larson, K.: Activity recognition in the home using simple and ubiquitous sensors. In: Pervasive Computing, Second International Conference, PERVASIVE 2004, Vienna, Austria, April 21-23, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3001, pp. 158–175. Springer (2004)
30. Thapliyal, H., Nath, R.K., Mohanty, S.P.: Smart home environment for mild cognitive impairment population: Solutions to improve care and quality of life. IEEE Consumer Electronics Magazine 7(1), 68–76 (2018)