Oh, wait, reasoning was wrong! Let's replay

Martin Kodyš[1,2], Joaquim Bellmunt[1,3], and Mounir Mokhtari[1,3]

[1] IPAL Image & Pervasive Access Lab - UMI CNRS 2955, Singapore,
{martin.kodys, joaquim.bellmunt, mounir.mokhtari}@ipal.cnrs.fr
[2] Université Grenoble Alpes, Grenoble, France
[3] Institut Mines-Télécom, Paris, France

Abstract. Decision making by real time rule-based reasoners is prone to errors. Modern systems must implement decisions based on partial, statistical, or approximate information. If a part of the information is missing or corrupted, the reasoner might be misled into reaching wrong conclusions. Systems using monotonic static logic are less affected, as they must be more careful in their decisions by design. On the other hand, human reasoning would prompt for some conclusions even if the conclusions may be wrong, and later rectify them if a new fact was discovered. To mimic similar reasoning in order to achieve users' satisfaction, the use of non-monotonic reasoning is preferred as it allows us to easily model a situation whereby a missing piece of information completely changes the outcome of the reasoner. However, detecting reasoning incoherence is very challenging, and thus there is a need to formulate strategies to deal with reasoning defaulting. In this paper, we propose the use of a replay mode on a part of the dataset, including ground truth data. This would also address the problem of late data arrival. As a case study, this paper discusses the implementation of a decision rectification in our ontology-based tracking system that is currently deployed in real conditions.

Keywords: Ontology, Rule-Based System, Decision Making, Reasoning, Semantic Web, Internet of Things, Semantic Replay

1 Introduction

Ontology-based reasoners are a convenient way to create an executable description of a specific universe using declarative programming [1]. Three components are needed for such a system: *Ontology* sets the frame and describes the portion of world we decided to model, *rules* complete the static description with inferred information, and *queries* select useful knowledge for the targeted use-case. As a complement, the *Internet of Things* (IoT) describes a world where machines and physical objects are seamlessly integrated into the information network, and communicate together to exchange and to process data. Together, they may yield a number of real applications.

The powerful combination of IoT and Linked Data deviates pervasive computing away from predefined bindings and from static communication protocols. Semantic technologies are used to perform a context-aware service provision in smart environments [2]. Indeed, we referenced four main purposes of these technologies in our use-case: 1. a model of personalised assistance in smart spaces including non-predictable behaviours; 2. integration of all entities of the system, with an environment discovery and configuration mechanism, 3. collaboration between modules of the system based on a shared model and vocabulary, 4. reasoning to create the system's intelligence, based on the three previous points.

Besides, systems using this architecture rely on the information coming from outside, possibly sensor events and it is necessary to handle the uncertainty of the environment. Many of these systems are used to make a decision in real time and never get an opportunity to question this decision. The most common strategy in case of any delayed or missing information is simply to discard it. Obviously, this strategy disables a possibility of taking decisions back. As a result, it may be preferable to use repaired decision to make future suggestions or to enhance the uncertainty handling. We propose a methodology, design and implementation that enables the system to"re-think" and backtrack its inferences. In terms of validation, our approach has been implemented and tested in an IoT Ambient Assisted Living (AAL) platform currently deployed in 23 homes.

In this paper we present an implementation of a reasoning replay in an AAL platform in order to open a discussion on its potential as a new paradigm. In the following section, we compare our work to other projects and concepts. In the next section, we explain our implementation, followed by another section describing achieved results. Then, we discuss the relative significance of this approach, in addition to useful cases of its application. Finally, we conclude with the overall outcome.

2 Related Work

"Reasoning replay" is a term that emerged from our platform implementation: it is the mechanism of resubmitting the knowledge to the reasoner in order to include facts omitted because they were unavailable at the moment when the decision making process was performed. Some systems accomplish a similar task without using this term – either because of its triviality (in trace based systems) where the reasoning replay is the key component of the design and the main functionality; or because of little or no added value in real-time systems with critical decision process and dilemmas (e.g. in driver-less cars). Our type of application of activity tracking fits in-between: decision repair is technically possible, and can greatly benefit the user monitoring the activity in real time. In the domain of activity tracking and ontology-based systems, we can find several technological solutions neighbouring our domain.

Trace-driven systems provide an interesting insight into data processing. One of the engines allowing to build such systems is *kTBS*, *"a kernel for Trace-Based Systems"* [3]. It provides a formalisation of a model of a trace, and REST API for storing and querying the data. The data being treated as a trace introduces several interesting concepts. A trace is viewed as a container for observed elements. Instead of a time-stamp, it uses a broader concept of an *origin* that allows reasoning over co-occurrence of elements without time-stamp. Another interesting feature is the differentiation between *stored trace* and *computed trace*. This trace-centred approach seems to be compatible with our approaches and we need to investigate the possibility of merging.

Amongst applications built with kTBS, we can mention semantic wiki interaction tracking via a browser plugin described in [4]. Activity tracking is based directly on computer-collected trace, and they examine the on-line behaviour of the users.

Similar concepts to ours, and their recent evolution are presented in [5]. The authors offer a full plug & play solution for a "Smart home in a box", although they do not mention any semantic replay, error correction, or processing of delayed events. Another daily activity tracking application is described in [6]. The traces are post-processed and it does not seem to have any real-time monitoring use case.

Using a different perspective, field of stream reasoning provides a solid formalisation of reasoning over a possibly massive knowledge stream (e.g. sensor events). The last decade displayed intensive efforts summarised in [7], notably in querying, benchmarking, and incremental reasoning. Similar to our problem, in [8], the authors tackled the issue of updating the knowledge in a non-monotonic reasoner.

3 Implementation

In order to assess the suitability of our solution, in this section, we describe our system's context, characteristics, and relevant configuration.

3.1 UbiSmart Platform

The purpose of the platform is to collect *quality of life* indicators, using them to detect risky situations [9,10], long-term evolution [11], and to enhance the quality of life by an intervention. The main monitored target group is the ageing and frail population, although caregivers can benefit as well [12].

To achieve its goals, the platform *UbiSmart* makes use of standard commercialised sensors producing events, a gateway **UbiGate** that relays the structured event data to the main component – our **server** written in *Node.js*. Eventually, notifications are sent to other devices as part of **service provisioning**. The reasoning is situated in the server: Incoming events are stored in a database, translated in triples using Notation3 format, and queued to be processed within the reasoning cycle.

Reasoning cycle Whenever the server is ready to process a queued event, reasoner (EYE[4]) is executed with knowledge originating from three ontological layers (as illustrated in Figure 1 as content of knowledge base, and in Listing 1.1, Figure 2):

[4] *Euler Yet another proof Engine*, `https://github.com/josd/eye`

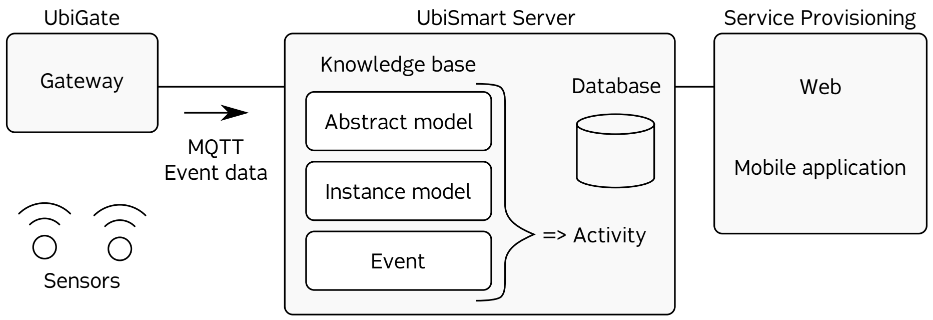


Fig. 1. UbiSmart framework in three sections. The UbiGate, the Server and the Service Provisioning.

– *Abstract model*: **Static ontology** describing the world that does not change (sensor type associated to its abilities and characteristics, type and description of detectable activities);
– *Instance model*: **Instantiated ontology** comprising of persisted information that was instantiated and has current information (e. g. sensor-room associations, durations of previously detected activity, . . .);
– *Action*: **Injected knowledge** produced by the sensors or a user interaction.

Using the described knowledge base, we apply rules that bring conclusions about what the user's current activity is, compute new durations, update the persisted information about this instance, and decide what notifications are to be sent. The conclusions are passed to software components performing decided actions. The reasoning cycle ends and waits till next event triggering a new one.

Reasoning context In this paper, we operate with the term "reasoning context". It is simply a set of knowledge elements that are made available from previous reasoning cycles and carry an important message about the events. Reasoning context is volatile because it is not persisted and theoretically can be re-created. Keeping track of the past is useful when we want to reason over past usage of objects. However, restarting the platform have for consequence a loss of this data, as the data is stored only in the RAM.

Origin of semantic components Ontology in our system has been created manually for UbiSmart [13], as well as the rules (example in Listing 1.2). They represent a common sense about observations (presence detection in a space, object manipulation) and implied conclusions based on common sense (the person is in the kitchen, preparing some food, receiving a visitor)

Listing 1.1. Selected triples from a snapshot of knowledge base at the end of a reasoning cycle. We can recognise the origin of the knowledge from 3 different levels of input (mentioned in *Reasoning cycle*). Each section in the listing needs the knowledge from previous one. Real-time injected knowledge is featuring the "*clock event*" in the

reasoning. The part of the output labeled as “conclusions” refers to the triples that are related to our use-case of activity tracking being the highest level that is presented to the end user. *Note:* Anonymous nodes (e.g. `_:b1`) that appear in *static ontology* part are generated by inference as we are interested in properties of a state and not in an identitifier for each state.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix qol: <http://www.ubismart.org/n3/qol-model#>.
@prefix hom: <http://www.ubismart.org/n3/home#>.

# OUTPUT/INPUT originating from Static ontology:
hom:house a qol:House.
hom:johndoe a qol:Resident;
  qol:residentIn hom:house.
hom:bed qol:locatedIn hom:house.
hom:nap a qol:Activity.
hom:livingroom a qol:Livingroom.
hom:SleepMatSensor rdfs:subClassOf qol:Sensor.
hom:SleepMatSensor qol:hasPossibleState _:b1.
hom:SleepMatSensor qol:hasPossibleState _:b2.
_:b2 qol:hasValue "bed_empty".
_:b1 qol:hasValue "bed_occupied".
_:b1 qol:indicateUse "true"^^xsd:boolean.
_:b2 qol:indicateUse "false"^^xsd:boolean.

# OUTPUT/INPUT originating from Persisted knowledge:
hom:sensor_sleepmac_b8 rdf:type hom:SleepMatSensor.
hom:sensor_sleepmac_b8 qol:attachedTo hom:bed.

# OUTPUT/INPUT originating from Real-time injected knowledge:
hom:clock qol:hasValue "2017-07-20T17:41:02+08:00"^^xsd:dateTime.
hom:sensor_sleepmac_b8 qol:hasCurrentState _:b1.

# OUTPUT Computed knowledge
hom:bed qol:lastUsed "2017-07-20T17:31:02+08:00"^^xsd:dateTime.
hom:sensor_sleepmac_b8 qol:lastUpdate "2017-07-20T17:41:02+08:00"^^xsd:dateTime.
hom:clock qol:lastUpdate "2017-07-20T17:41:02+08:00"^^xsd:dateTime.
hom:johndoe qol:detectedIn hom:livingroom.
hom:johndoe qol:inRoomFor "6213.0"^^xsd:decimal.
hom:livingRoom qol:motionMeasured "0"^^xsd:integer.

# OUTPUT Computed knowledge Conclusions important for the end user: activity tracking
hom:johndoe qol:believedToDo hom:nap.
hom:johndoe qol:doesActivitySince "2017-07-20T17:31:02+08:00"^^xsd:dateTime.
```

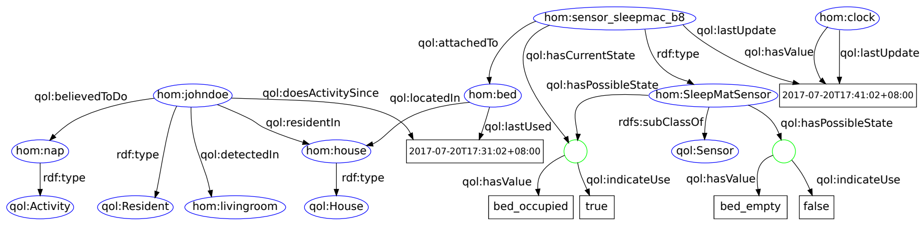


Fig. 2. Visualisation of the knowledge base in Listing 1.1.

Listing 1.2. Example of activity detection rule: *"nap" activity gets 4 more points if the user is detected in a livingroom for 600 or more seconds and motion was measured less than twice (within last 300 seconds).* Generally, the rule-based scoring system (not detailed in this paper), selects the activity having the highest score to become the object of a Computed knowledge *Conclusions* triple `hom:johndoe qol:believedToDo []` If there are too many activities with a similar score, then no activity is inferred.

```
@prefix qol: <http://www.ubismart.org/n3/qol-model#>.
@prefix hom: <http://www.ubismart.org/n3/home#>.
@prefix math: <http://www.w3.org/2000/10/swap/math#>.
{?u qol:detectedIn ?r. ?r a qol:Livingroom. ?u qol:inRoomFor ?d. ?d math:notLessThan 600. ?r
     qol:motionMeasured ?m. ?m math:lessThan 2} => {hom:nap :getScore 4}.
```

3.2 Requirements

Some assumptions are necessary in order to apply enhancements.

- **Decision is an inference conclusion.** The most obvious requirement is that the *decision* is a subset of inferred knowledge, e.g. all objects of a triple `:Reasoner :concludes ?conclusion.`
- **Independent cycles of iterations.** The system works in cycles, a cycle is composed of one call or of a sequence of calls to a reasoner. The number of iterations within a cycle is supposed to guarantee the stability of conclusions. It depends on the complexity and the order of rules. Usually, the output of previous iteration is used as the input of the following one.
- **Time independence.** The system must not depend on the processor's clock. All time-related computation must take inputs from a virtual clock that emulates the processing in the past. Time-related context must be entirely provided from the input parameters.
- **Full context available.** Not only all of the events but also the reasoning context must be fully available for each cycle, otherwise, only full replay is possible. Usually, a database may contain significant snapshots is needed, otherwise, replay would be needed from the beginning of the last restart.

3.3 Initial Situation

In this section, we detail some aspects of our system that is necessary to understand our implementation.

No repair strategy Our system performs a decision making process to infer the activities of a person. Inputs are events from various sources (remote sensor events, user interaction, and open data). Figure 3 illustrates the inputs, **events**, and main outputs, **activities**. In our real deployment, recurrent problems impacted the output: events were blocked on the gateway and could not reach our reasoner due to a connectivity problem, gateway malfunction, or temporary unavailability of the server intercepting the data. The system was not able to deal with delayed data. The information that eventually reached the server or could be extracted from backed up local copies of the data was not included in reasoning. The deployed strategy was to discard (ignore) delayed events.

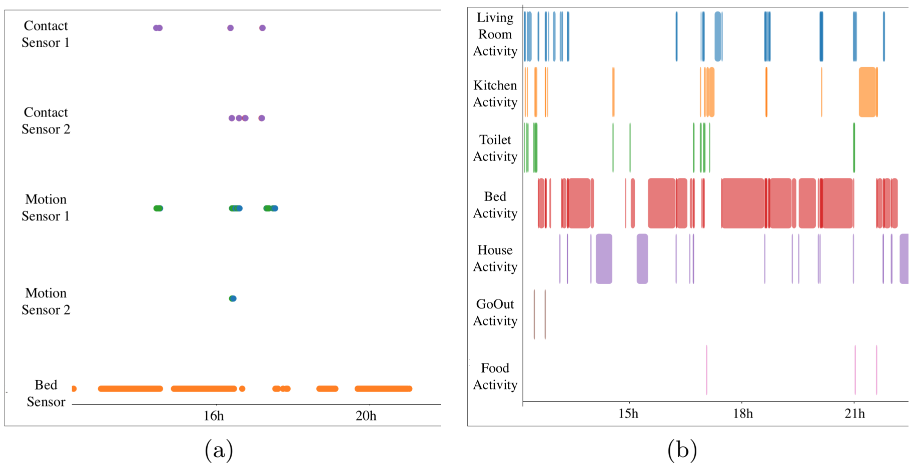


Fig. 3. Data visualisation in UbiSmart platform: (a) sensor events plotted in time, same colour denotes the same kind of event; (b) activities inferred from sensor events in (a)

Time-awareness and "clock event" Our reasoning needs to take into account the time flow. To achieve it, we use the fact that events trigger a new reasoning cycle and carry the information about the time. To take care of the cases when the delay between the events is too long, we set a timer. If no event arrives, we issue a special volatile *"clock event"*. As any other event, it resets the timer and launches a new reasoning cycle. This way, we guarantee a minimum time resolution for the time dependent activity detection (if the person is in a room for more than 30 minutes and last sensor event happened 20 minutes ago, we can infer that the person might have fallen). In addition, within the cycle, we are able to compute durations and trigger time-related actions (e.g. notifications). Setting the timer delay to shorter period gives higher time resolution but also may create a processing overhead. Unlike "real" events, *"clock events"* are not persisted. A representation of a *"clock event"* in the knowledge base is shown in Listing 1.3.

Listing 1.3. A *"clock event"* representation in knowledge base

```
hom:clock qol:hasValue "2017-07-22T18:36:06+08:00"^^xsd:dateTime.
```

In terms of *kTBS*, recorded events are *stored trace* and the union of recorded events and *"clock events"* is computed trace of first order. The activities inferred are computed trace of second order.

3.4 Modifications

Fundamentally, requirements (Section 3.2) were respected, only time-independence had to be implemented. Initially, system was time-dependent and every event got evaluated at its arrival time using the server machine's system time to provide *current time*. This time measure was used to compute the duration of activities, quantity of movement of the person and other indicators.

Replacing the system time by the time included in every *"clock event"* with an optional delay of arbitrary one second (in order to get non-zero duration activities) fulfilled the requirement of time-independence.

Automatic *clock event* has to be disabled during the replay. As every regular event starts a timer for an automatic *clock event*, we added a replay tag on all events prepared for replay. This property is used to discriminate and omit the replayed events, so that they are not duplicated in the event database and they do not trigger the automatic clock. If a new event (not a part of replay) enters the queue, the timer for clock events will naturally resume.

4 Results

Our implementation yielded following results and observations that prove the feasibility of the approach, and underline some issues and open problems.

Parameters For the actual replay test, we used a dataset containing 22 034 events covering 84 days of event data. After completing the dataset with 105 138 generated clock events indicating the points in time when the evaluation should take place, we obtained the total of 127 172 events to be replayed. The *"clock events"* were generated in the same way as the system would generate them in real-time mode: if the gap between two consequent events is more than 30 seconds, a *"clock event"* is inserted every 30 seconds in-between. Scheduling of clock events is illustrated in Listing 1.4.

Listing 1.4. Example of scheduled events (in JSON format) with `replay` tag, showing 30-second delay of the inserted clock event.

```
{
      "date": "2017-07-20T21:31:23.000Z",
      "house": 101,
      "sensor": "sleepMac_b8",
      "value": "Bed_Occupied",
      "id": 708409,
      "createdAt": "2017-07-20T21:31:23.000Z",
      "updatedAt": "2017-07-20T21:31:23.000Z",
      "replay": true
},
{
      "event": "clock",
      "date": "2017-07-20T21:31:53.000Z",
      "replay": true
}, ...
```

This *"clock event"* timer parameter was chosen by tests with different values: with 10 seconds, the number of events was too high and with 10 minutes (600 seconds), the resolution of resulting reasoning was too low to activate rules that are capturing activities shorter than 10 minutes. As the event data came from the real deployment, we already had a dataset to compare with – the real-time inference. The parameters differences are summed up in Table 1.

Table 1. Parameters for real-time run and for replay.

Parameter	Real-time reasoning	Replay
"Clock event" timer max. spacing between events	10 minutes	30 seconds
Delay between event time and current time variable	delay between event time-stamp and hardware clock at processing time	1 second
Activity change	nap	occupied

Execution The process was executed on x86_64 architecture virtual host with 2 400 MHz CPU. It took 78 hours and 58 minutes to complete (using 63 hours and 12 minutes of processor time) in our UbiSmart platform built with *Node.js*.

Observations Resulting activities have been compared to a running instance in real time. The differences were observed and are illustrated in Figure 4. The expected differences that the replay displays are as follows:

1. higher activity granularity (darker colour indicates more short activities),
2. new activities previously undetected (*nap* between 00:00 and 03:00),
3. overflowing activities (18:00 to 20:00 *nap*, where it was not supposed to be detected).

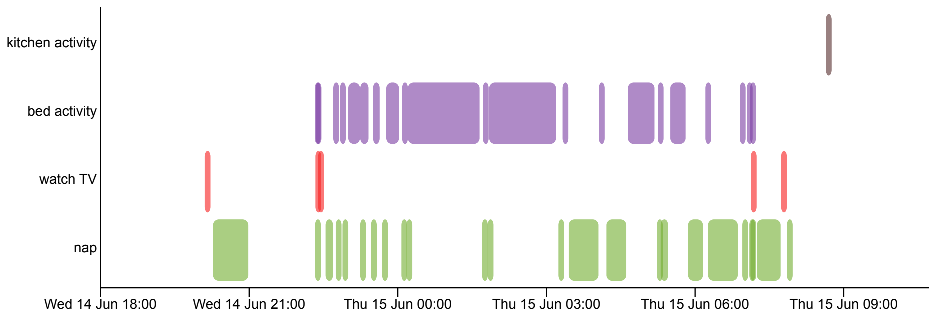
5 Discussion

The results suggest several questions around the proposed approach and its possible future.

5.1 Shall We Replay?

As it can be clearly seen in the execution details in previous section, the replay is very *resource intensive*. Therefore, strategies for partial replay have to be considered for more effective resource use. For example, store checkpoints of current state regularly and resume at a checkpoint to make it more efficient. However, depending on the ontology, the decision taken in the past can have impact deeply in the future. The question of handling such cases remains open.

We can distinguish two main motivations for such a replay: **enhance the resulting dataset** or **enhance the decision process**. For instance, in the first case, the priority is to *provide the best results* even if they come later; the second case might consider *delaying certain decisions* or make them more fuzzy if a certain type of data tends to arrive late. These two points of view are mutually compatible, and can lead to a richer event handling and greater user satisfaction.



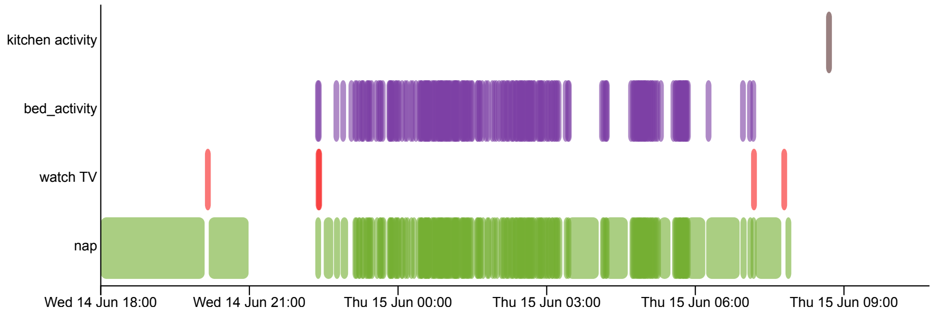


Fig. 4. Samples of inferred activities from the same set of events. Upper graphics is real-time inference, lower image is replay with shorter timer duration. Minimum width of an activity representation is set to 5 pixels so that short events are visible. If events are too close they visually overlap and appear in darker colour.

5.2 Result Management

Replay implementation also opens a question about what to do with the old decisions. Shall we just replace them? If we keep them, how many layers of replay?

For our system, we decided to keep the older decisions in order to: keep track of the state of the system in the past (we have to know what an observer might have seen when looking at the system). This can also be used for a differential analysis. We label previous decisions as "disabled by [a reference to a new decision]". As for our use-case, the optimal solution is to keep the oldest decisions and the newest one. It allows us to explain why the notification was sent without confusing the end user (caregiver) with too many versions of what the activity might have been.

5.3 Impact on Our Platform

In our case, the replay implementation helped to point out the difference between an uninterrupted operation of the platform during the replay and the case when the platform is interrupted and restarted. When the platform is restarted, we may lose some context information. Replay allows us to test these differences.

Another observation exposes some issues with previous implementation and brings a solution. It depended on the time when the event was effectively evaluated within the reasoner. In the case of processor overload, e.g. unrelated heavy processing on the server, the events queued and were processed much later. We were computing the duration of an activity as the time difference between the current machine time, and the first in an uninterrupted series of the same event. As a consequence, the activities appeared to have a longer duration than they actually had. Another reason of discrepancies was the misalignment of the clocks of event producer and consumer where we could get negative duration of an inferred activity.

Particularity of the execution was its non-uniformity in time. In the beginning, the execution pace was quick and at around two cycles per second. In the middle, the pace slowed down to as low as 2 minutes per cycle. At the end, the execution resumed a quick pace at one cycle per second.

5.4 Possible Development

For future development, we consider these topics: "**dynamic replay**", when an event arrives late, roll back the events and replay; "**real resume**", context conservation during of platform restart; "**progress monitoring**" to show how the replay performs as part of other user interface replay management; "**augmented visualisation**" when the platform is confronted with a replayed information, serve it to the end user (caregiver).

6 Conclusion

In this paper, we explained our motivations for a semantic replay. We implemented and tested it in an existing *Ambient Assisted Living* platform. We demonstrated that activity tracking is a domain where this approach has some potential, in particular, because of the availability of unused data. We believe that the replay paradigm can bring useful insights for other systems as well, in the case of real time user-oriented systems, such as care-giving. Additional motivation is driven by users' needs to understand why a false alarm was triggered while being able to see a correction.

Even if it is clear that discarding events is the simplest strategy, the benefit of replay approach is highly case-dependent. Therefore, we believe that a discussion about this topic can be very enriching.

As any other feature, it makes sense to implement it only in certain conditions. In our case, the recalculation cost is high but its benefits are higher, and we believe the process can be enhanced and be made more efficient in case of partial replay.

References

1. Uschold, M., Gruninger, M.: Ontologies: Principles, methods and applications. The knowledge engineering review **11**(2) (1996) 93–136
2. Bellmunt, J., Mokhtari, M., Abdulzarak, B., Aloulou, H.: Agile framework for rapid deployment in ambient assisted living environments. In: Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services. iiWAS '16, New York, NY, USA, ACM (2016) 410–413
3. Champin, P.A., Prié, Y., Aubert, O., Conil, F., Cram, D.: kTBS: Kernel for Trace-Based Systems (2011) A reference implementation of the notion of Trace Based Management System. Allows to store and compute modeled traces, and access them through a RESTful interface.
4. Le, A., Lefèvre, M., Cordier, A., Skaf-Molli, H.: Collecting interaction traces in distributed semantic wikis. In Camacho, D., Akerkar, R., Rodríguez-Moreno, M.D., eds.: 3rd International Conference on Web Intelligence, Mining and Semantics, WIMS '13, Madrid, Spain, June 12-14, 2013, ACM (2013) 21
5. Hu, Y., Tilke, D., Adams, T., Crandall, A.S., Cook, D.J., Schmitter-Edgecombe, M.: Smart home in a box: usability study for a large scale self-installation of smart home technologies. Journal of Reliable Intelligent Environments **2**(2) (2016) 93–106
6. Cook, D.J.: Learning Setting-Generalized Activity Models for Smart Spaces. IEEE intelligent systems **2010**(99) (September 2010) 1
7. Dell'Aglio, D., Della Valle, E., Harmelen, F., Bernstein, A.: Stream reasoning: A survey and outlook: A summary of ten years of research and a vision for the next decade. (07 2017) 1–24
8. Beck, H., Dao-Tran, M., Eiter, T.: Answer update for rule-based stream reasoning. In: Proceedings of the 24th International Conference on Artificial Intelligence. IJCAI'15, AAAI Press (2015) 2741–2747
9. Bellmunt, J., Mokhtari, M., Abdulzarak, B., Aloulou, H., Kodyš, M.: Experimental frailty model towards an adaptable service delivery for aging people. In: Engineering of Complex Computer Systems (ICECCS), 2016 21st International Conference on, IEEE (2016) 227–230
10. Sadek, I., Bellmunt, J., Kodyš, M., Abdulrazak, B., Mokhtari, M.: Novel unobtrusive approach for sleep monitoring using fiber optics in an ambient assisted living platform (2017)
11. Kaddachi, F., Aloulou, H., Abdulrazak, B., Bellmunt, J., Endelin, R., Mokhtari, M., Fraisse, P.: Technological approach for behavior change detection toward better adaptation of services for elderly people. In: HEALTHINF. (2017) 96–105
12. Mokhtari, M., Endelin, R., Aloulou, H., Tiberghien, T.: Measuring the impact of icts on the quality of life of ageing people with mild dementia. In: Smart Homes and Health Telematics. Springer (2015) 103–109
13. Aloulou, H., Mokhtari, M., Tiberghien, T., Endelin, R., Biswas, J.: Uncertainty handling in semantic reasoning for accurate context understanding. Knowledge-Based Systems **77** (2015) 16 – 28