Activity Recognition Through Complex Event Processing: First Findings

Sylvain Hallé and Sébastien Gaboury
LIF, Université du Québec à Chicoutimi, Canada

Bruno Bouchard
LIARA, Université du Québec à Chicoutimi, Canada

Abstract
The activities of daily living of a patient in a smart home environment can be detected to a large extent by the real-time analysis of characteristics of the habitat’s electrical consumption. However, reasoning over the conduct of these activities occurs at a much higher level of abstraction than what the sensors generally produce. In this paper, we leverage the concept of Complex Event Processing (CEP), in which low-level data streams are progressively transformed into higher-level ones, to the task of activity recognition. We show how the use of an appropriate representation for each level of abstraction can greatly simplify the process. We also report on the use of an existing event stream processor to successfully implement the complete chain, from low-level sensor data up to a sequence of discrete and high-level actions.

Introduction
The world population over age 60, especially in Europe, the U.S. and Canada, is projected to increase from 11 percent today to 22 percent by 2050 (UN1, 2013). As the population ages, more people will need care; at the same time, resources, such as caregivers and funds, are diminishing critically, leading to the need to completely rethink the way we help people in loss of autonomy. This reality, combined with recent advances in the field of Artificial Intelligence (AI) has led to the emergence of a new research paradigm called Ambient Intelligence (AmI).

Ambient intelligence (Ramos, Augusto, and Shapiro, 2008) refers to a multidisciplinary approach which consists of enhancing an environment (room, building, car, etc.) with technology (e.g. infrared sensors, pressure mats, etc.), in order to build a system that makes decisions based on real-time information and historical data to benefit the users within this environment. The main application of this AmI concept involves the development of smart homes (Augusto and Nugent, 2006), which are houses equipped with ambient intelligent agents providing advanced, assistive services to a resident, in the execution of Activities of Daily Living (ADL). The first AI-related problem that needs to be solved is that of human activity recognition (Bouchard, Giroux, and Bouzouane, 2007). This challenge can be summarized by the following question: how do we take raw data from sensors, filter it, and then transform that into relevant information that can be associated with the basic ADLs? Observations from sensors are not only useful for recognizing the patient ongoing activities, but also for learning routines and creating a user profile detailing daily living activities that he is used to perform by analysing the history log.

In the literature, most of the proposed recognition approaches exploit probabilistic models, such as Hidden Markov Model (HMM) or Bayesian networks, which allow representing the uncertainty level in the recognition process (Roy et al., 2014). For creating the activities’ knowledge base and learning the usual routines, most the existing works exploited data mining algorithms to learn the models corresponding to the set of ADLs (Bouchard et al., 2015).

However, most of these approaches suffer from what could be called an impedance mismatch: to a large extent, the data captured by sensors installed in a smart home are low-level data elements such as voltage, power, signal strength or on/off triggers. Yet, the activities that need to be monitored and acted upon can only be recognized through the aggregation of such data elements over time —in other words, actual reasoning over a patient’s activities occurs at a much higher level of abstraction. This mismatch has mostly been addressed through ad hoc means in previous works, or discarded altogether and regarded as “implementation details”. Only a handful of works recognize the event-based nature of action recognition (Storf et al., 2009; Suresh et al., 2011).

In this paper, we leverage the concept of Complex Event Processing (CEP), in which low-level data streams are progressively transformed into higher-level ones, to the task of activity recognition. CEP has already found many applications, mostly in the fields of stock market data analysis, sensor processing and realtime databases. Its event-based nature makes it a fitting paradigm for the recognition of ADLs. First, we briefly recall how this task can be done through the real-time analysis of characteristics of the habitat’s electrical consumption. Then, we provide some background on CEP, and show how the use of an appropriate representation for each level of abstraction can greatly simplify the process. We also report on the use of an existing event stream processor to successfully implement the complete chain, from low-level sensor data up to a sequence of discrete and high-level actions. These early findings show the potential of the approach in a
real-world context, as the use of electrical appliances could be detected easily with ample computing power to spare.

**Power-Based Action Recognition**

In a recent work, Belley et al. (2015) developed an algorithmic method for detecting actions in the home environment based on the concept of Non-Intrusive Appliance Load Monitoring (NIALM) (Zeifman and Roth, 2011). NIALM describes a process to detect changes of state in the voltage and the electric current supplying a house or a building, which directly influence the power difference. The main idea is to be able to identify the load signature of each appliance present in a home. Typically, the parameters considered are the voltage, the electric current and the power (active and reactive). These signatures can be recognized by monitoring in real time the changes of state in the voltage and the electric current supplying the house. In summary, the NIALM approach consists in associating a device with the proper load signature extracted from a single power meter installed at the main electrical panel.

In their works, Belley et al. (2015) focused on two main features related to power consumption, namely, the active and reactive power of each appliance that is being used within the house. They divided their algorithmic method in two distinct steps. The first step (learning) consists of establishing a representative load signature database based on three characteristics: active and reactive power variations during an on/off event (steady state), the line-to-neutral that supplies the appliance, and the presence (or absence) of a peak of active power when the device is turned on (transient state). They recorded each signature by turning on and off each appliance individually a sufficient number of times. For example, Figure 1 shows the components of the electrical signal measured in real time when a blender is being operated. One can easily see the spike, plateau and subsequent drop on the “S.WL1” line, which in this case represents the active power of Phase I of the habitat’s electrical box.

The second phase (recognition) consists of detecting in real time the changes in the on/off status of appliances by analyzing the variation in these three characteristics and to compare these changes to the signatures in the database. When there is a match, it means that an “on” of an “off” event/action has been detected on a specific appliance. In previous work, this recognition step was performed by producing a decision tree whose inputs are the components of the electrical signal at various moments.

However, it turns out that creating this tree is a tedious and error-prone task. For example, Figure 2 shows only a tiny portion of such a tree, successfully detecting the activity of about a dozen electrical appliances. One can see how the tree, apart from being barely legible, entangles decisions about all appliances at once, making it very hard to understand how each is actually detected.

Moreover, since the detection of the signal is based on the presence of spikes and plateaus of given heights, the processing cannot be done by passing each data sample to the decision tree in isolation. These samples have first to be buffered, aggregated over a sliding window of a given width, and preprocessed to detect any signal features, before being passed to the decision tree for the detection of an appliance. Therefore, the actual knowledge about the detection of these appliances resides not only in the tree, but is also intertwined across a bunch of hand-coded scripts. Although successfully showing the concept, it was suspected early on that proceeding in such an ad hoc way would prove deleterious on a larger scale.

**Event Stream Processing**

This is where the concept of Complex Event Processing (CEP) comes into play. CEP exists to help understand and control event-driven information systems (Luckham, 2005). These techniques are based around the concept of complex events, that is, in other words, events that could only happen because other events happened before it. The idea is that events are related because of the reason they happened, when they happened and the membership between them and other events. In a way, CEP distinguishes itself from “static”, database-related approaches by focusing on the dynamic nature of the data to process: events are created in an ordered fashion, propagated into streams, and must be processed, aggregated and filtered in near-realtime.

This basic definition has been implemented in multiple ways in the recent past. A variety of CEP software and theoretical frameworks have been developed, which all differ in...
a number of dimensions. For example, TelegraphCQ (Chandrasekaran et al., 2003) was built to fix the problem of continuous stream of data coming from networked environments; it shares similarities with the STREAM system (Arasu et al., 2004). The SASE language (Wu, Diao, and Rizvi, 2006) was brought as a solution to meet the needs of a range of RFID-enabled monitoring applications; the system does not, however, use a hierarchy of complex event types, meaning that it is impossible to create complex event types from other complex events.

Since most CEP engines tend to use a single processing thread, Siddhi (Suthothayan et al., 2011) focuses on the multithreading aspect of the CEP engine. We shall also mention the flow language Lustre (Halbwachs, Lagnier, and Ratel, 1992), the Cayuga system (Brenna et al., 2009), which uses a specific query language (Cayuga Event Language) close to SQL, and finally Borealis (Abadi et al., 2005), which provides a Graphical Query Editor that simplifies the composition of streaming queries using a boxes-and-arrows interface. Events in both Cayuga and Borealis are limited to tuples of values, while we shall see that our problem requires the handling of events of arbitrary type (sets, bags, tables, vectors, or single numerical values).

Our choice ultimately fell on BeepBeep, an event stream processor recently developed at LIF (Hallé, 2015) and available under an open source license.\(^1\) BeepBeep was chosen for its relative ease of use, and most importantly, its capacity to compose together processing units that use widely differing specification languages.

It shall be noted that BeepBeep is a generic event processor, not aimed at AI tasks in particular. However, we shall see how the composition of its basic “building blocks” is appropriate for the problem at hand, thereby showing the power of the approach. As we shall see, it is possible in BeepBeep to connect the output of a signal processor to the input of a Moore machine, and send the result to a monitor for Linear Temporal Logic, all in the same chain.

Contrarily to the majority of event processors, BeepBeep is based on a very simple formal model first introduced in Hallé and Varvaressos (2014). Let $T$ be an arbitrary set of elements. An *event trace of type* $T$ is a sequence $T = e_0, e_1, \ldots$ where $e_i \in T$ for all $i$. The set of all traces of type $T$ will be denoted $T^*$. In the following, event types will be denoted in double strike (e.g. $T, U, \ldots$) and can refer to any set of elements. Event types can be as simple as single characters or numbers, or as complex as matrices, XML documents or any other user-defined data structure.

A *processor* is then defined as a function taking zero or more input traces, and returning zero or more output traces. Borrowing terminology from the theory of relations (Quine, 1945), a processor accepting one trace as input will be called *monadic* (or unary), while one accepting two traces will be called *dyadic* (or binary). Since processors both have input and output, they must be qualified according to both—one may hence talk about a dyadic-input, monadic-output processor, or more succinctly a 2:1 processor.

---

\(^1\)http://lif-labs.github.io/beepbeep-3

**Step 1: Learning**

Leveraging these concepts, we shall show how the use of an event stream processor can greatly simplify the task of detecting a patient’s activities in a smart home environment from low-level sensor data. The first step of the process consists of using CEP for discovering the load signatures of each appliance.

Figure 3 describes the chain of basic event processors that are used to discover the peaks on the electrical signal. In this figure, events flow from the left to the right. The signal from the electrical box is sent to a first processor, which transforms raw readings into name-value tuples, one for each time point. Each tuple contains numerical values for various components of the electrical signal; for example, parameter $WL1$ measures the current active power of Phase 1.

The second processor picks one such parameter from the tuple, extracts its value, and discards the rest. The output trace from this processor is therefore a sequence of numbers. This sequence is then fed to the third processor, which detects sudden increases or decreases in a numerical signal. For each input event, the processor outputs the height of the peak, or the value 0 if this event is not a peak. Since an event needs to be out of the window to determine that it is a peak, the emission of output events is delayed with respect to the consumption of input events.

The next step in the processing takes care of removing some of the noise in the signal. Typical appliances consume at least 100 W and generate a starting peak much higher than that. Therefore, to avoid false positives due to noise, any peak lower than 100 W should be flattened to zero.

In order to do so, the output from the peak detector is replicated in two traces. The first one (top) is sent to a simple comparator, which compares the input value with the constant trace 100, and returns either true or false. This result is the first input of the *dispatcher* processor, represented in Figure 3 by traffic lights. The second input of the dispatcher is the output of the peak detector itself, while its third input, in this case, is the constant trace 0. The dispatcher’s task is simple: given a triplet of events $(e_1, e_2, e_3)$, (one from each of its inputs), output $e_2$ if $e_1$ is true, and output $e_3$ otherwise. In the present case, this has indeed for effect of replacing all events of the peak detector lower than 100 W to 0.

The resulting trace requires one further cleanup task. Again due to the nature of the electrical signal, two successive peak events may sometimes be reported for the same sudden increase. The last processor takes care of keeping only the first one. This *yield* processor behaves like the dispatcher, but with the additional guarantee that the second input will be selected at most once in every $n$ successive events. In the present context, this has for effect of eliminating “ghost” peaks in the signal.

Given a feed from an electrical signal, this complete chain of processors produces an output trace of numerical events; most of them should be null, and a few others should indicate the occurrence of an abrupt increase or decrease in the values of the input signal, along with the magnitude of that change. Moreover, the position of these events, relative to the original signal, also indicates the exact moment this change was detected. As an example, Figure 1 shows the realtime
value of three components of the electrical signal, to which the output of the peak detector was superimposed. One can see that the detector behaves as we want, reporting exactly two changes of the appropriate magnitude at the right time.

This approach presents several advantages over the original method. First, the processing of the raw electrical signal is expressed as the composition of simple building blocks. Apart from providing an appealing graphical representation, these processors also take care of many tedious tasks one would otherwise have to code by hand: management of event windows, buffers, type conversions. Second, since the process is decomposed into simple operations, modifying that chain becomes an easy task. For example, averaging every two successive events, in order to smoothen the input signal, simply amounts to inserting that operation at the appropriate location and re-piping the rest of the chain accordingly. Adding the same operation in a monolithic block of custom code is comparatively much more challenging.

Finally, and perhaps most importantly, the chain of processors elegantly represents the process of taking a low-level stream of input data, and progressively lift it to a higher level of abstraction. What began as a sequence of raw meter readings ends up as a trigger for the higher-level concept “drastic change of magnitude x”.

Figure 3: The piping of processors for discovering peaks on the original electrical signal. Elements in pink indicate parameters that can be adjusted, changing the behaviour of the pipe.

As a matter of fact, BeepBeep provides constructs to encapsulate the whole chain of processors as a newly-defined entity, taking as input one trace of raw readings and four parameters. This is shown in Figure 4. The four pink elements of Figure 3, representing each a parameter of one of the processors in the chain, become the four parameters of the box. The first defines the component of the electrical signal one wishes to analyze; the second is the width of the window used for the peak detector; the third specifies the minimal magnitude of change required to report something, while the final argument can be used to adjust the dampening in the case of duplicate peaks.

Some appliances exhibit activity on more than one signal component. This is the case of e.g. the oven, which, in our smart home environment, produces variations in active and reactive power on phases 1 and 2; however, these variations always occur at the same moment in time for all components; the approach described here can hence be easily extended to that case by merging the peaks and plateaus of multiple components into a single, compound event.

Step 2: Detection

The previous step allows us to easily measure signal peaks and plateaus for each appliance. As was explained earlier, these variations are fairly constant over multiple uses of the same appliance, and can therefore be used as its “signature” on the electrical signal. As an example, Table 1 shows average values for various appliances; each value is a rounded average over 10 different event traces. It shall be noted that for some appliances, the peak is equal to the subsequent plateau; this indicates that the power raises immediately to a stable regime, without going through an initial burst of consumption (as is the case for e.g. the blender).

The second step is to lift peak and drop events to a yet higher level of abstraction, and to report actual appliances
Empirical Testing

Detecting the activity of a single appliance can be grouped into a single appliance monitor, containing the complete chain from selecting the appropriate signal component, detecting peaks and plateaus, thresholding, filtering and damping the resulting signal, and piping it to an appropriately-configured Moore machine. The end result is a monitor whose input are raw electrical readings, and whose output are on/off events for that appliance. Then, detecting the use of multiple appliances can be done by simply duplicating the original electrical readings to each monitor and merging their output into a single trace, as is shown in Figure 6. The result is a high-level trace of interleaved events for each appliance.

As was described in Step 2 above, the output from BeepBeep is a sequence of “appliance X on/off” events. In all cases, BeepBeep was able to correctly identify the appliance being turned on or off at the appropriate moment. On modest hardware, the event traces were processed at a rate of roughly 18 kHz, more than 300 times the actual sampling rate of the habitat’s sensors. This indicates that the processing of low-level sensor data into increasingly higher levels of abstraction does not impose an undue burden in terms of computing load, and that even commodity workstations have ample processing power to tackle this task in such a manner.

Conclusion

In summary, this paper has shown how the detection of activities of daily living in a smart home environment can be done efficiently through the realtime analysis of electrical load signatures, which in turn, can be elegantly framed into a problem of Complex Event Processing. The whole process, of collecting electrical sensor data, filtering and applying signature recognition to them, has been reduced to the composition of half a dozen basic processors, piped in a chain transforming low-level data points into high-level on/off events. Besides from providing a simple and formal representation of the whole process, we have shown experimentally on a real-world CEP event stream engine how this can effectively be done by hand.

Table 1: Average peaks, plateaus and drops, as experimentally measured for a few home appliances.

<table>
<thead>
<tr>
<th>Appliance</th>
<th>Peak (W)</th>
<th>Plateau (W)</th>
<th>Drop (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blender</td>
<td>$\Delta P_1: 300$</td>
<td>$\Delta P_2: 200$</td>
<td>$\Delta P_3: -258$</td>
</tr>
<tr>
<td>Coffee pot</td>
<td>$\Delta P_1: 939$</td>
<td>$\Delta P_2: 939$</td>
<td>$\Delta P_3: -674$</td>
</tr>
<tr>
<td>Kettle</td>
<td>$\Delta P_1: 1,427$</td>
<td>$\Delta P_2: 1,427$</td>
<td>$\Delta P_3: -1,094$</td>
</tr>
<tr>
<td>Toaster</td>
<td>$\Delta P_1: 831$</td>
<td>$\Delta P_2: 831$</td>
<td>$\Delta P_3: -787$</td>
</tr>
</tbody>
</table>

Figure 5: The Moore machine for detecting on/off events for a single appliance.

Figure 6: Watching multiple appliances is a matter of duplicating the original electrical readings to each monitor and merging their output into a single trace.

2Asus T200TA with 2GB of RAM, quad-core 1.46 GHz CPU
applied to the live recognition of actual electrical appliances.
These promising first results open the way to various paths
of future work. First, the learning and detection of load signa-
tures could be streamlined into a library easily parameteriz-
able for each appliance to detect. Second, the same approach
could be applied to other types of data, such as RFID sensors.
Finally, the output from such a detection process can itself
be taken as the input of a yet higher level of processing, this
time operating on patterns of appliance events in order
to detect anomalies or build a profile of frequent behaviours.

References
Abadi, D. J.; Ahmad, Y.; Balazinska, M.; Çetintemel, U.;
Cherniack, M.; Hwang, J.-H.; Lindner, W.; Maskey, A.; Rasin,
The design of the borealis stream processing engine. In CIDR,
18(9):785–793.

Arasu, A.; Babcock, B.; Babu, S.; Cieslewicz, J.; Datar, M.;
Stream: The stanford data stream management system. Tech-

Augusto, J. C., and Nugent, C. D., eds. 2006. Designing
Smart Homes. The Role of Artificial Intelligence, volume
4008 of Lecture Notes in Computer Science. Springer.

Belley, C.; Gaboury, S.; Bouchard, B.; and Bouzouane, A.
2015. Nonintrusive system for assistance and guidance in
smart homes based on electrical devices identification. Expert

To appear.

Bouchard, B.; Giroux, S.; and Bouzouane, A. 2007. A
keyhole plan recognition model for Alzheimer’s patients:

Distributed event stream processing with non-deterministic
finite automata. In Gokhale, A. S., and Schmidt, D. C., eds.,
DEBS. ACM.

Chandrasekaran, S.; Cooper, O.; Deshpande, A.; Franklin,
M. J.; Hellerstein, J. M.; Hong, W.; Krishnamurthy, S.; Madd-
en, S.; Raman, V.; Reiss, F.; and Shah, M. A. 2003. Tele-
graphCQ: Continuous dataflow processing for an uncertain
world. In CIDR.

and verifying real-time systems by means of the synchronous
18(9):785–793.

Hallé, S., and Varvaressos, S. 2014. A formalization of
complex event stream processing. In Reichert, M.; Rinderle-
Ma, S.; and Grossmann, G., eds., 18th IEEE International
Enterprise Distributed Object Computing Conference, EDOC
2014. Ulm, Germany, September 1-5, 2014, 2–11. IEEE
Computer Society.

Hallé, S. 2015. A declarative language interpreter for CEP.
In Mayer, W.; Hallé, S.; and Grossmann, G., eds., 19th IEEE
International Enterprise Distributed Object Computing Con-
ference, EDOC 2015, Adelaide, Australia, September 22-24,

Luckham, D. C. 2005. The power of events – An introduction
to complex event processing in distributed enterprise systems.
ACM.

Maitre, J.; Glon, G.; Gaboury, S.; Bouchard, B.; and
Bouzouane, A. 2015. Efficient appliances recognition in
smart homes based on active and reactive power, fast Fourier
transform and decision trees. In AAAI, 1–6.

Moore, E. F. 1956. Gedanken-experiments on sequential
machines. Automata Studies, Annals of Mathematical Studies

Quine, W. V. O. 1945. On the logic of quantification. The

Ramos, C.; Augusto, J. C.; and Shapiro, D. 2008. Ambient
intelligence - the next step for artificial intelligence. IEEE
Intelligent Systems 23(2):15–18.

Roy, C. P.; Bouchard, B.; Bouzouane, A.; and Giroux, S.
2014. Ambient activity recognition in smart environments for
cognitive assistance. Int. J. of Robotics Applications and

Storf, H.; Kleinberger, T.; Becker, M.; Schmitt, M.; Bomarius,
F.; and Prueckner, S. 2009. An event-driven approach to ac-
tivity recognition in ambient assisted living. In Tscheligi, M.;
de Ruyter, B. E. R.; Markopoulos, P.; Wichert, R.; Mirlacher,
T.; Meschtscherjakov, A.; and Reitberger, W., eds., Ambient
Intelligence, European Conference, AmI 2009, Salzburg, Aus-
tria, November 18-21, 2009. Proceedings, volume 5859 of
Lecture Notes in Computer Science, 123–132. Springer.

Suhothayan, S.; Gajasinghe, K.; Loku Narangoda, I.; Chatu-
second look at complex event processing architectures. In
Proceedings of the 2011 ACM Workshop on Gateway Com-
puting Environments, GCE ’11, 43–50. New York, NY, USA:
ACM.

Suresh, V.; Ezhilchelvan, P. D.; Watson, P.; Pham, C.; Jack-
son, D.; and Olivier, P. 2011. Distributed event processing for
Zdonik, S. B.; and Vincent, P., eds., Proceedings of the Fifth
ACM International Conference on Distributed Event-Based
Systems, DEBS 2011, New York, NY, USA, July 11-15, 2011,
371–372. ACM.

Nations.

Wu, E.; Diao, Y.; and Rizvi, S. 2006. High-performance
complex event processing over streams. In Chaudhuri, S.;
Hristidis, V.; and Polyzotis, N., eds., SIGMOD Conference,
407–418. ACM.

Electronics 57(1):76–84.