CAREACT: AN ADAPTIVE AND CONTEXT AWARE FRAMEWORK FOR ANOMALY DETECTION IN AMBIENT ASSISTED LIVING

by

Mustafa Ozan Özen

B.S. Computer Engineering, Boğaziçi University, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2013

To Pınar Özcan,

For her endless support and love.

ACKNOWLEDGEMENTS

I'd like to thank Prof. Cem Ersoy for being a father figure for me far away from home and always being there whenever I truly needed. Without his guidance and support this long journey wouldn't even have started. I am truly grateful for everything he has done for me.

I specially offer my deepest thanks to Tim Van Kasteren who showed the light when I could see nothing but pitch black. He set this thesis on the right course.

I am truly grateful to Evren Önem for his friendship, guidance and covering my deficiencies throughout this work.

I am also truly grateful to Yunus Dönmez for all his effort on making my thesis right.

I would like to offer my thanks to my bosses Can Alhas and Müjdat Ayoğuz for their understanding throughout the thesis process.

I would like to thank to my family. They have always supported me whichever direction I wanted to with my life. They always offered their help without a shred of doubt and always made me feel secure and happy with myself.

And finally I'd like to thank my to my fiancee Pınar. She was always there whenever I felt like quitting and get me back to what I should be doing. She sacrificied her sleep, happiness and sometimes sanity to help me lift this heavy burden. It was her endless love and support that help me finish my thesis after long and stressful years.

This work is partially supported by Bogazici University Research Fund under the grant numbers 6056 and 6370.

ABSTRACT

CAREACT: AN ADAPTIVE AND CONTEXT AWARE FRAMEWORK FOR ANOMALY DETECTION IN AMBIENT ASSISTED LIVING

In this thesis, we analyzed current trends and challenges in ambient assisted living environments and deduced that there is a need for a monitoring system that incorporates existing telemonitoring concepts to activities of daily living recognition systems in a way that is easy to integrate, requires little or no maintenance after deployment and is easy to modify whenever needed. To address this need, we present CAREACT, an ambient assisted living system which monitors daily lives of inhabitants for anomaly detection in real-time with a set of features supported by state of the art literature, that have not been combined in a single system up to date: interoperability, rule based pattern detection, adaptivity and analyzability. CAREACT is built on complex event processing technology which has built-in support for rule based pattern detection and interoperability that works in real-time under heavy load. In this work, we also present a new methodology, called future events, that changes the paradigm of processing the time windows within patterns in a way that it takes constant time to process time windows regardless of their size. We implemented CAREACT and integrated it with one of the existing activities of daily living systems in our testbed environment to show that the proposed architecture is applicable and indeed it increases the efficiency of the anomaly detection process.

ÖZET

CAREACT: ÇEVRE DESTEKLİ YAŞAM ALANLARINDA NORMAL DIŞI DURUM YAKALAMA İÇİN UYARLANABİLİR VE BAĞLAM DUYARLI BIR ÇATKI

Bu çalışmada, ortam destekli yaşam çevrelerinde mevcut eğilimler ve zorluklar analiz edilmiş ve günlük yaşam algılama sistemlerini uzaktan izleme konusundaki kavramlarla uyumlu çalışacak, çabuk entegre edilebilir, bakımı ve geliştirilmesi kolay bir izleme sistemine ihtiyaç duyulduğu sonucuna varılmıştır. Bu gereksinimi karşılayabilmek adına geliştirdiğimiz, ortam sakinlerinin günlük yaşamlarını izleyip olağandışı gelişmeleri gerçek zamanlı olarak algılayabilen CAREACT sistemi bu güne kadar bir arada görülmemiş özellikleri bünyesinde barındırmaktadır. Bu özellikler arasında birlikte çalışabilirlik, kural tabanlı olay örgüsü algılama, analiz edilebilirlik ve uyarlanabilirlik bulunmaktadır. CAREACT üzerine inşa edildiği karmaşık olay işleme teknolojisinin kural tabanlı olay örgüsü algılama, gerçek zamanlı çalışma ve analiz edilebilirlik yeteneklerine tamamıyla sahip olup yoğun yük altında başarılı bir biçimde çalışabilmektedir. Bu tez çalışmasında ayrıca son derece yenilikçi bir yaklaşım olan "Gelecek Olaylar" kavramı üzerinde de durulmaktadır. "Gelecek Olaylar" yaklaşımı zaman kısıtı işleme modelini değiştirmektedir ve bu sayede ilgili kısıtları işleme süresi sabit zamanlı olarak gerçekleşmektedir. CAREACT sistemini geliştirip test merkezimizdeki mevcut izleme sistemlerinden birine entegre ederek, önerilen mimarinin uygulanabilir olduğunu ve olağandışı etkinlikleri algılama yetisini kayda değer bir şekilde artırdığını göstermiş bulunmaktayız.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

λ_i	Overall arrival rate to node i
r_i	Arrival rate for queue i
μ_i	Service rate of node i
ρ_j	Utilization of node j
p_{ij}	Probability of customer going to queue j from node i
$E[N_i]$	Expected number of customers on queue i
$E[R]$	Expected latency

LIST OF ACRONYMS/ABBREVIATIONS

AAL	Ambient assisted living
ADL	Activities of daily living
CEP	Complex event processing
RFID	Radio frequency identification
NFA	Nondeterministic finite automaton

1. INTRODUCTION

With the elderly population ratio increasing throughout the world, one of the main challenges of the developed countries is to reduce healthcare costs while maintaining the quality of the healthcare [1]. One of the paradigms to address this challenge is called ambient assisted living (AAL). This approach allows elderly to live independently at home longer and centralizes the process of caregiving by continuously monitoring the living environment with sensory systems, for the purpose of detecting and acting on exceptional cases via anomaly detection systems and finally assisting residents in their regular lives by recognizing Activities of Daily Living (ADL) [2].

Current AAL literature covers several enabling technologies including telemonitoring and ADL recognition. In telemonitoring, the main purpose is to monitor inhabitants of an AAL environment continuously by multi-modality of sensors [3, 4] and detect anomalies by finding patterns within this data [5–7] to provide services like heartrate monitoring and reminder generation. Sensor data is also processed by ADL recognition systems to create higher level information of human activities [8, 9], which enable broader range of applications like cognitive assistance and human environment interaction as well as anomaly detection. In ADL recognition context, the anomaly detection is either matching a pattern of activities or detecting a single activity like falling for elderly, which attract significant research effort at present [10, 11] while the former is often overlooked in ADL-centric work. Another aspect of anomaly detection, that is also overlooked in ADL-centric work and to some extent in telemonitoring, is the creation of automatic or semi-automatic reactions on matched patterns. While there is a large variation on the detection, reaction and learning capabilities of the existing monitoring systems, the current literature lacks a holistic study on reacting against detected patterns through a generic anomaly detection system which works on both sensor and activity data (See Figure 1.1). In the following paragraphs, we describe the current challenges for making AAL environments more efficient from the anomaly detection perspective and present a mapping between the challenges and the existing features

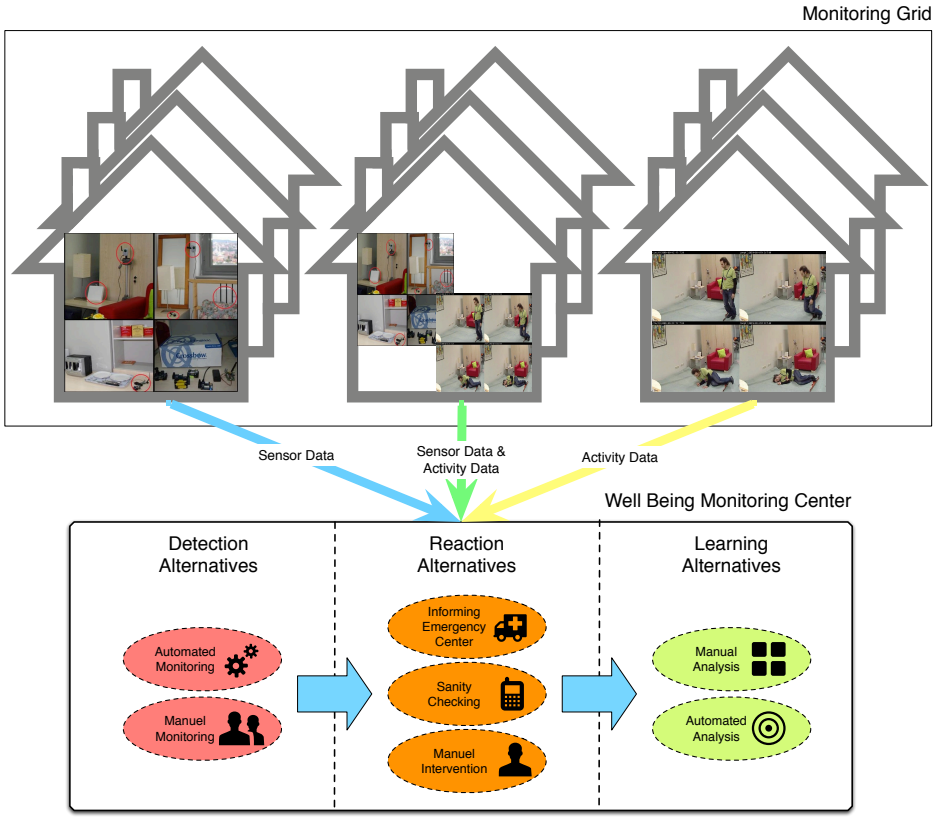


Figure 1.1. A ”full” monitoring system where a typical monitoring system consists of a subset of the modules depicted above.

in the literature.

The first challenge is becoming aware of the context. A very simple example for the importance of the context is lying on the bed at midnight vs lying on the kitchen floor at noon. Without the context information of the place and the time of the lying activity these two cases are indistinguishable. Furthermore, there might be more than one context in a smart home. It is inefficient to create anomaly detection systems for each and every of them while a centralized approach offers not only detecting anomalies that happen within one context but also detecting those that span over more than one context. An example of such an anomaly is taking a shower vs. taking a shower after firing the kitchen stove while preparing the breakfast. Again while the former scenario is perfectly normal, the latter might be a possible cognitive disorder of an Alzheimer patient and without the context information from the both contexts they are indistinguishable from one another. An anomaly detection

system should be able to fuse data from wide variety of computation subsystems including sensors, smartphone, ADL-recognition systems. An important consideration for this type of system is that it is expected to work in (soft) real-time to be effective [1] and usually having a higher quality of contextual information degrades system performance because it is typically achievable by increasing the number of sensory and/or ADL recognition systems. Moreover, increasing the number of the monitored AAL environments that is served by one such system will decrease the service costs of the system, which puts more burden on the performance requirements of the system.

The second challenge arises from the human factor in the design of the anomaly detection patterns. The quality of the anomaly detection pattern is prone to mistakes, misleading assumptions and so on [12]. While some of the problems created by such factors are identifiable before the deployment by manual validation, testing or simulation, some of the problems might be missed due to the infrequency of the problem or the differences between the testing environment and the actual environment. To identify these problems, the performance of the anomaly detection patterns needs to monitored in the AAL environment, the flawed patterns needs to be fixed and redeployed until they are performing reasonably well for their respective anomaly pattern.

The next challenge is handling the inter-personal and intra-personal variations while executing the same task. While some of the activities are almost universal like sleeping, some of the activities have high variation that might affect the anomaly detection process like taking medication via pills or injection. An anomaly detection system should provide the ability of tailoring the anomaly detection patterns to the monitored individuals. The differences are not limited to monitored individuals. There is also a high variation in the layouts of the apartments, in the deployed set of sensors and communication devices in the monitoring environments [13]. The anomaly detection patterns should be customizable depending on the capabilities of the smart home instance.

The final challenge is the extensibility of an anomaly detection system. The patterns in the anomaly detection system should not break when a new recognition capability is added to the monitoring environment and furthermore the new recognition capabilities should be used

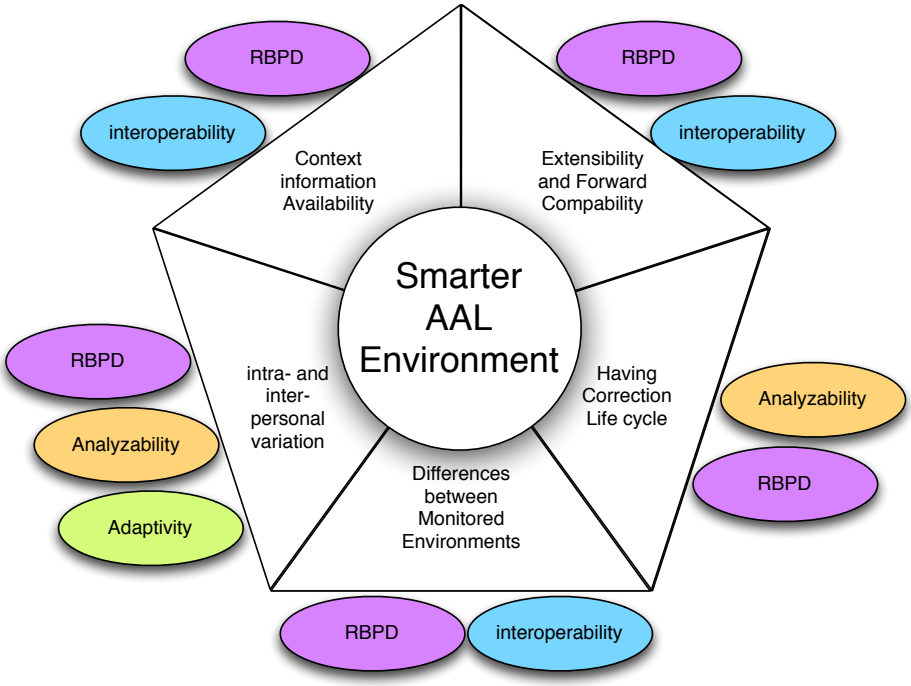


Figure 1.2. Dependencies of the current challenges to the features proposed by the existing literature: Rule Based Pattern Detection (RBPD), interoperability, adaptivity, analyzability

for updating the old ones as well as creating the new ones. Assume a new smart pill bottle is added to the monitoring environment that had pill forgetting detection pattern in place. The system should continue functioning without any changes and should allow improvements for the pill forgetting anomaly detection by incorporating the new capability.

We have identified four features that address the challenges described above in the current literature: rule based pattern detection [14], interoperability [7], adaptivity [10] and analyzability [15]. We have visualized the relationship of these challenges to the features in Figure 1.2. While partial combinations of these four features which address a subset of the mentioned challenges existed in the current literature, there is no single framework that supports all these features.

Interoperability support enables all modalities of sensory and ADL systems to coexist in the same environment. System experts, who design anomaly detection patterns, can fuse data from multiple modalities to express a unified anomaly detection pattern, which is an im-

provement over using a single modality. Rule based pattern detection gives system experts the ability to fuse the data from any of the existing subsystems and contextual information related to that subsystem in designing anomaly detection patterns. Furthermore, this feature together with interoperability also gives the system experts the flexibility of changing existing anomaly detection patterns to match the variance of the deployed subsystems in a given environment as well as creating new ones to extend existing environment with new subsystems. While rule based pattern detection helps system experts in designing anomaly detection patterns for different deployment environments, it offers very limited flexibility to handle parameters of these patterns, like threshold values, that vary from resident to resident and evolve through time. With such adaptivity, designing anomaly detection patterns that adjust to these varying parameters becomes possible. Additionally, one can define the concept of analyzability as keeping track of data generated by sensory and ADL systems as well as generated reactions. Such data offers a chance of analysis for the system experts to measure performance of existing anomaly detection patterns and to find new anomalies that exist in the context of the system.

In this work, we present CAREACT, a novel AAL framework for monitoring multiple number of AAL environments that supports all these four core features. CAREACT is a framework under which various sensory and ADL recognition systems coexist and the data of those systems are fused by a pattern detection engine to deduce anomalies and to automatically create appropriate reactions on such anomalies in real-time. To support all these features with a solid performance while processing massive amounts of data generated by the subsystems of the CAREACT system, we employ the Complex Event Processing (CEP) [16] architecture. The CEP architecture supports working in real-time under heavy load conditions while providing inherent support for interoperability and rule based pattern detection.

We also make a contribution to the CEP literature in our work by including a novel mechanism called Future Events, that addresses a well-known challenge within the CEP architectures. In AAL systems, it is typical to observe anomaly patterns spanning a time window higher than minutes. Nevertheless, current literature is limited in processing such time windows under heavy load due to buffering problems [17–21]. With Future Events, it

becomes possible to process time windows of higher orders.

We implement CAREACT in our testbed laboratory environment [22,23] and show that it works in a real life setting. We also demonstrate that CAREACT supports both adaptivity and analyzability with experimental results.

The rest of the thesis is organized as follows. In Chapter 2, we discuss the state-of-the-art AAL systems and CEP engines in more detail. We present the architecture of CAREACT, details on the CAREACT engine and Future Events on Chapter 3. In Chapter 4, we present our implementation in a real life setting as well as the validation of adaptivity and analyzability of CAREACT. Finally, the conclusion, discussions and future directions are presented in Chapter 5.

2. Related Work

2.1. Ambient Assisted Living

There are many studies on telemonitoring in the context of AAL. We classified these works into four groups on their support for interoperability and rule based pattern detection:

- without interoperability or rule based pattern detection support
- with rule based pattern detection support, without interoperability support
- with interoperability support, without rule based pattern detection support
- with both interoperability and rule based pattern detection support

In the first group of studies, data integration model and rule based pattern detection decision capability are either omitted or the details are not supplied. Lubrin et al. [15] and Jiang et al. [24] propose a WSN based monitoring system where data is transferred to a central station and written to a database. However, they do not present a way to act on detected patterns. Zhang et al. [5] propose WSN based monitoring system where monitoring information is presented to doctors for reviewing to decide on alerts. One of main goals of monitoring in ambient assisted living environments is to detect emergencies and one of these emergencies is fall incidents. In [11], Doukas et al. propose a system that uses a multimodality of motion, audio and visual sensors on fall detection process. Naranjo et al. [10] propose a telemonitoring system that also focuses on fall detection. Their work supports adaptation by learning system parameters from working deployments and pushing these parameters to all deployments via firmware updates.

In the second group of studies rule based pattern detection is addressed but interoperability is not. Lu et al. [25] proposes a middleware for telemonitoring applications that have a standard interfaces in the sensor level up to the application layer. They propose two approaches for reporting in sensor level: periodic or event based. Pattern detection is left to developers to generate code customized to telemonitoring on top of their middleware where each new case has to be coded from scratch. Huo et al. [26], propose a system that monitors

elderly inhabitants in indoor and outdoor settings for activity and health state decisioning, for emergency decisioning and alert generation purposes. In this work, sensor data are transferred using publicly available protocols such as GSM and Wi-Fi but no details on data modeling are presented. They employ a decision making engine which does not have any implementation or specification details. Ungureanu et al. [27], focus on fetal monitoring on their work and they leave pattern detection process to telemedicine systems by integrating to those systems.

In the third group of studies, data integration models are addressed, however rule based pattern detection capabilities are either omitted or details are not presented. Corchado et al. [7] describes a telemonitoring system built on top of SLYPH, a framework for light physical devices on service oriented architecture. This enables them to standardize interfaces between heterogeneous sensors. In their architecture, human monitoring operators are the main decisioning units. When a sensor detects an anomaly like falling, video/voice link is set up between home and monitoring center and operators decide on the case. Liu et al. [4], proposes a healthcare system, HealthKiosk, where they address data integration between data centers and biomedical sensors. They propose a sensor proxy which collects sensor data and correlates it with semantic information, like social security number, through a standard interface called topic interface. Sensor proxy also employs event driven messaging engine and a micro processing engine which supports simple operations like aggregation and transformation. Resulting pre-aggregated data is transferred to KioskKontroller, where the data is monitored. Anomaly decision making is left to the human operators.

In the last group of studies, data integration models and rule based pattern detection capabilities are both addressed. Mouttham et al. [14] propose an event driven data integration model for telemonitoing purposes which employs a CEP based decision engine. With this architecture they are able to flexibly integrate all building blocks, achieve context awareness, real-time alert generations over application based architectures. They do not include an analyzabiltiy capability or adaptivity capability in their system. Storf et al. [13] also present an event driven approach, mainly for activity detection that has built-in anomaly detection and alert generation capabilities. Their system focuses on adaptivity and real-time detection of anomalies. They employ different types of approaches in activity recognition including

Table 2.1. Summary of existing telemonitoring systems from interoperability, rule based pattern detection, adaptivity and analyzability aspects

	Interoperability	Rule Based Pattern Detection	Adaptivity	Analyzability
Huo [26]	✗	✓*	✗	✗
Ungureanu [27]	✗	✓*	✗	✗
Lubrin [15]	✗	✗	✗	✓***
Jiang [24]	✗	✗	✗	✓***
Zhang [5]	✗	✗	✗	✗
Doukas [11]	✓	✗	✗	✗
Naranjo [10]	✗	✗	✓	✗
Lu [25]	✗	✓**	✗	✗
Corchado [7]	✓	✗	✗	✗
Liu [4]	✓	✗	✗	✗
Mouttham [14]	✓	✓	✗	✗
Storf [13]	✓	✓	✓*	✗
Catarinucci [28]	✓**	✓	✗	✓

CEP. They support adaptability by a trend calculator but do not specify the details, nor have an analyzability capability. In [28], Catarinucci propose a context-aware and rule based healthcare monitoring system for anomaly detection. Their system support interoperability which is limited to sensory systems, rule based pattern detection and analyzability.

In Table 2.1, we summarized the existing telemonitoring systems. While every feature is used by at least one existing telemonitoring system, there is no single telemonitoring system that supports all four features. Furthermore, the works marked with * do not present the details on how they support a given feature, the works marked with ** are either task or system specific and hence are not suitable for generic usage. The works marked with *** have a central database for analyzability support which can be thought as the simple version of a knowledge base.

There are other studies that are also related to the monitoring in AAL environments. In one of these studies, Szucs et al. [29] overview the importance of the data modeling in remote monitoring environments. They compare the problem specific data model to the

generic model and come to the conclusion that the former favors simplicity, integrity and the performance whereas the latter favors reusability, integration and flexibility.

Human tracking is an important aspect of the AAL monitoring environments. On this subject Ngai et al. [30] propose a prototype RFID based healthcare management system that is tested by potential users of the system and confirmed the practical viability of the system. On a related work, Toplan et al. [31], show that RFID based systems can track inhabitants of an ambient assisted living environment when privacy is a concern.

Another aspect of the monitoring in AAL environments is recognizing activities of daily living. To accomplish such goal, Quintas et al. propose an HMM based approach [32] to detect human behaviours in the context of smart homes. Their approach employs hierchical models to detect human behaviours from low level sensor data. Chen at al., on the other hand, propose a knowledge-driven approach [9], that is based on ontological modelling and semantic reasoning which is orthogonal to statistical modelling. Tracking undiscovered activities, is an important extension to activity recognition domain where current literature primarily focuses on identifying preselected activities. To this end, Rashidi et al. propose an automated approach [33], that track frequently repeated activities in daily living by statistically clustering sequences of sensor events, using a voting multi Hidden Markov Model(HMM) as the boosting mechanism.

2.2. Complex Event Processing Engines

CEP technology finds complex relations of events, time and historic/semantic information while employing an event driven integration model [16]. It has advantages over database driven pattern mining methodologies which has many conventional use cases in the literature [34]. The most important feature of CEP is that it performs well in real-time under heavy load. Furthermore, CEP on top of an event driven architecture allows various different sources of information on multiple modalities, including RFID readings, sensor readings, detected activities and time, to be integrated as event sources and to be fused in the same rule pattern. Such a feature hinders the creation of a more meaningful context information and hence more accurate reactions. There are numerous use cases of CEP in the context of

healthcare, like smart hospitals [35, 36], activity recognition [13] and e-health [14].

In order to present CEP literature in detail, initially we will present CEP terminology that will be used in the context of this thesis. An event, also called as primitive event, is an occurrence of change of data within a system. The event type is the semantic name of the change. A parameter of an event is a component in the structure of an event that specifies the properties of the change. A pattern is the correlation of events, time and semantic/historic information. These patterns might include value based or temporal constraints. Value based constraints are related to values of event parameters, whereas temporal constraints are related to occurrence/non-occurrence of events within certain time windows. Matched patterns are complex events. Decisions taken on these complex events are called actions. Finally, patterns together with action(s) are called a scenario.

In this section, we will mainly focus on the aspect of processing temporal constraints in CEP engines. Temporal constraints are implied by correlation operators. While there are two main approaches, query based and GUI based, for defining patterns, we will stick to naming convention of operators defined by the query based approach. These operators are SEQ, AND, OR and NEGATION. All of the state-of-the-art engines support time window based execution. While time windowed version of these operators might vary depending on the engines, we will use the same name for both windowed and non-windowed version of the same operator.

These operators create complex events within patterns. While primitive events have only one value as the timestamp, namely the occurrence timestamp, complex events have at least two time stamps associated with them, which are the start timestamp and the end timestamp. Start and end timestamps hold first and last matched primitive event times respectively. If the matched events are also complex events, the process of holding timestamp information becomes much more complicated [37]. In their work, White et al. tackle with the problem of event ordering. If events are successfully ordered, then time window processing and negation are almost simple boundary comparisons. However, they state that in order to successfully achieve real ordering of events one must store complete history of events.

State-of-the-art CEP engines have different solutions, petri nets, trees, NFAs to handle scenarios with temporal constraints but they all employ some kind of buffering of events. In their work [20], Adaikkalavan et al. propose the SnoopIB, which is a CEP engine that differentiates point based semantics, which assumes an event has only occurrence time, from interval based semantics, in which events have both a start time and end time, for temporal constraint management and shows how these different semantics can yield different results. They use event detection graph based execution where events are stored and different correlation operators are applied on them using interval based semantics.

Wu et al. present SASE [17], which is a nondeterministic finite automaton (NFA) based CEP engine that has five layers of processing in pattern matching: sequence scan and construction (SSC), selection, windows, negation and transformation. In SSC layer, an NFA based pattern matcher is applied to the event stream to find event sequences. These sequences are intermediate results that are filtered by consequent processing layers. Selection layer filters event sequences with parameter based constraints, whereas window layer applies its filtering using temporal constraints. Finally negation layer filters sequences that include negated events. Remaining sequences are converted to complex events by the transformation layer. They also present a result where throughput decreases linearly with increasing window size in temporal constraints.

In a more recent work, Demers et al. present Cayuga [21], a CEP engine that mainly targets real-time stock trading. In Cayuga pattern detection rules are represented with a query language, Cayuga Event Language, and transformed into an automaton at the time of execution. While there is no support for event negation, temporal constraints are resolved with an operator called FOLD. The underlying processing mechanism stores a new state machine instance for every event that do not change state of an existing state machine instance.

Another event processing language is EventScript [38]. The EventScript is an event processing language that has reaction capabilities on matched patterns. They use regular expressions as the main computation paradigm to be more programmer friendly for the pattern detection process. They do not present a way to tackle with the temporal constraints and event negation.

In a more specialized problem domain of managing RFID events, Welbourne et al. present Cascadia [39], which uses a probabilistic model for processing events to handle noisy data of RFID streams. They aim to decouple low-level RFID data from end user applications by enabling developers to specify high-level events. They do not present a way to handle temporal constraints or event negation in real-time.

On the same topic, Nie et al. [40], present a declerative Cleansing Language for Unreliable RFID Event Streams (CLUES) that enables RFID streams to be processed by CEP engines directly. They propose a complex event processing paradigm that extends NFA based CEP arhitectures. Like Cascadia, both temporal constraints and event negation are out of the scope of this work.

Agrawal et al. present a query based event stream processing language [41] that supports event negation. They focus on defining a query evaluation model, related NFA and analyzing its expressibility. They also do not present a way to process temporal constraints in event streams.

Mei and Madden propose the ZStream [18], a tree based cep engine that supports temporal constraints and event negation. In the tree, every leaf corresponds to events storage areas whereas internal nodes correspond to operations. Events are stored in leaf nodes if they satisfy value based constraints and are advanced to intermediate nodes if they satisfy operator relations with the other connected leaf nodes. Temporal constraints are applied at this stage by removing events that are outside the range of time windows. Such an evaluation is done in rounds to decrease memory consumption.

In their three related works [19, 42, 43], Liu et al concentrate on processing nested pattern queries. They support temporal constraints and event negation and search for patterns on stored events with a mechanism to purge events that are outdated due to temporal constraints. The effect of increasing time window sizes on the performance of the system is not presented in their work.

We present a summary of existing event processing techniques in Table 2.2. It should

Table 2.2. Summary of existing CEP systems from the perspective of temporal constraint and event negation support, the methodology to handle events and finally whether or not they store events.

	Temporal Constraint	Event Negation	Methodology	Buffering
Adaikkalavan [20]	✓	✓	Event Detection Graph	✓
Wu [17]	✓	✓	NFA	✓
Demers [21]	✓	✗	NFA	✓
Cohen [38]	✗	✗	Regular Expression	✗
Welbourne [39]	✗	✗	Probabilistic Event Extractor	✗
Nie [40]	✗	✗	Probabilistic Event Extractor	✗
Agrawal [41]	✗	✓	NFA	✗
Mei [18]	✓	✓	Tree-based	✓
Liu [19, 42, 43]	✓	✓	Nested Query Plan	✓

be noted that every engine that supports temporal constraints require events to be stored in some manner before patterns are matched. However, storing the entire event history costs unbounded memory, whereas processing this history on every relevant event costs unbounded CPU time. While unbounded CPU time is not acceptable for many application domains, the real value of complex events lies on their availability in (near) real-time [44]. There is a trade off between achieving full time window based execution or negation, which are actually temporal constraints, and working in (near) real-time. Because full time window based execution requires complete history of events to be stored and processed which contradicts with bounded execution time requirements of real-time processing.

In its essence, buffering creates two problems: memory management of finite but unbounded number of events to match a pattern and supporting (near) real-time execution. There are also side problems that might come up depending on the execution strategy on the CEP engine, namely, creating intermediate result sets that do not produce complex events. These intermediate result sets are the complex events that are matched within a pattern, that might or might not match the entire pattern. The problem with these intermediate result sets are that they cost CPU time to calculate, eliminate, garbage collect and memory to store.

The engines that support full a spectrum of temporal constraint management, perform

this task at the cost of additional CPU and memory usage. The negative consequence of these costs is that they grow larger as the time window of temporal constraint gets larger. Buffered events or aggregated intermediate results increase over time since they are not available for purging due to increasing window size. This type of execution might work in real-time under light load conditions or with relatively smaller time windows, typically in the order of milliseconds or seconds. But in the AAL domain, there are cases where we are interested in correlations of events that are minutes, hours or days apart. One of the challenges of using CEP engines in telemonitoring scenarios is the lack of a more light-weight mechanism that enables processing of simpler temporal constraints.

Other than temporal constraint approaches, Wu et al. [45] focus on reacting to matched patterns. Their proposal employ a semantic repository where events are enriched with semantic level information. This semantic repository is also used for changing semantic information while reacting to events. This concept is a key milestone for achieving self adaptivity on CEP engines.

In [12], Coffi and Marsala, present an adaptive framework for generic harmful situation detection to mitigate errors created by the evolution of monitored environment over time. They employ relational association rule mining and inductive logic programming together to extract new knowledge from monitored environment, which in turn is used to adapt system to the monitored environment. This study achieves adaptation to environmental changes over time.

3. CAREACT

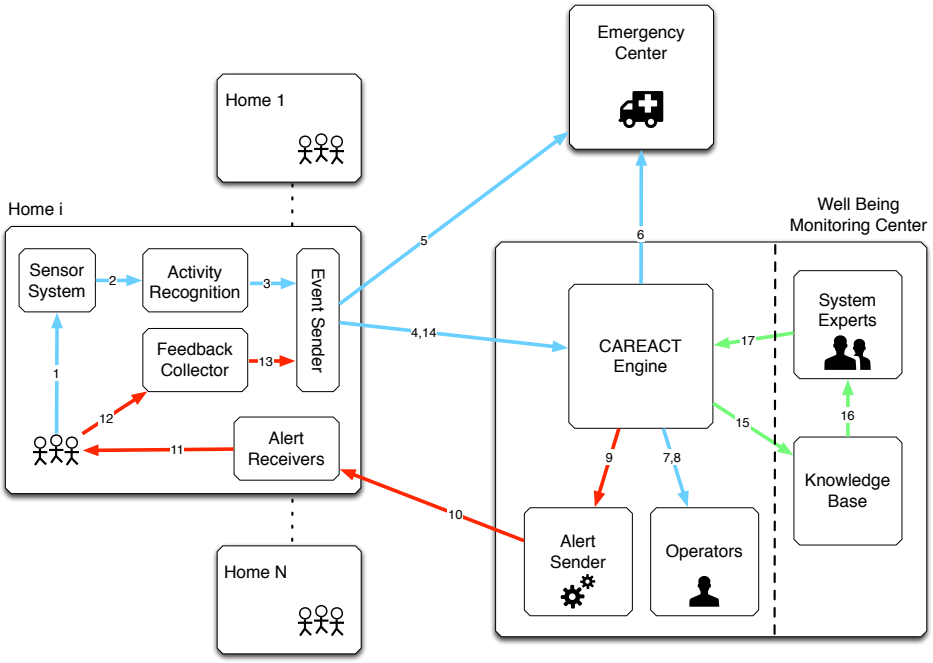
In this chapter, we present CAREACT, a novel anomaly detection and ambient assistance framework that incorporates telemonitoring concepts on multimodality of sensory and ADL recognition systems with the additional support for interoperability, rule based pattern detection, adaptivity and analyzability. Our main contribution arises from two facts where the former is that no existing system in the literature supports such features in an all-in-one manner, and the latter is that CAREACT applies telemonitoring concepts to multimodality of both sensory and ADL recognition systems.

We will describe the architecture of CAREACT in Section 3.1 and the details of the CAREACT Engine and its anomaly detection methodology in Section 3.2. Future Events, our proposed mechanism to handle huge time windows, is presented in Section 3.2.1. Section 3.3 is dedicated to how CAREACT supports interoperability, rule based pattern detection, adaptivity and analyzability.

3.1. Architecture of CAREACT

In this section, we present the architecture and the logical data flow indexed with their transition sequence numbers in the CAREACT system (See Figure 3.1). In this architecture, a sensor network constantly monitors daily lives of inhabitants (i) and transfers this raw sensor data to ADL systems (ii). The details of how the data is collected and transferred to ADL systems depend on the design of the integrated sensory systems and are out of the scope of this work. Subsequently; activities including sitting, standing and cooking and abnormal activities like falling are transformed into events (iii) and sent to the CAREACT engine (iv). In case of communication failures, events that describe an emergency by itself, like falling, are sent to an emergency center (v).

After successful delivery of events from a monitored home to the CAREACT engine (iv), events are processed to find anomaly patterns and to create reactions for these patterns. There are three types of external alert actions in our system. Most urgent and low false-



1	Sensor Network monitors daily lives of caretakers	10	Alerts are passed to Alert Receivers.
2	Raw sensor data is passed to Activity Recognition module to be processed	11	Inhabitants are alerted via mobile phone, custom mobile devices, ambient peripherals
3	Activities, which are calculated from raw sensor data are passed to Event Sender module	12	Inhabitants can create feedbacks to sanity checks and reminders
4	Activities and feedbacks are transformed into events and sent to CAREACT engine to be processed in scenarios	13	Feedback Collector collects feedback information and passes it to Event Sender to be transformed into feedback events
5	If an emergency event can not be sent directly to the CAREACT engine, it is sent to Emergency Center as an Alert	14	Feedback events are sent to CAREACT engine to be used in adjusting inhabitants semantic data
6	Type 3 Alerts, which have high priority or result of operator feedback are sent to Emergency Center	15	All of the events, feedback events and actions are stored into global information store
7	Type 2 Alerts, which require human intervention are sent to operators	16	Global Information store can be used for data mining by System Experts to aid them on their scenarios design process
8	Operators decide the level of importance of alert and send their feedback back to the CAREACT engine	17	Better fitting or new scenarios are created with analytic results and deployed to engine
9	Type 1 Alerts, which include reminders, sanity checks or Operator feedbacks, are passed to Alert Sender		

Figure 3.1. The architecture of the CAREACT system and transition sequence table for the logical data flow.

positive rated alerts are Level 3 alerts and these are directly sent to emergency centers without any human intervention (vi). Level 2 alerts are where human intervention is required to double check the anomaly (vii). Operators decide on the type of these alerts, namely false-positive, Level 3 or Level 1. The decision is also passed as an event to the CAREACT engine (viii). Internal scenarios are created to handle this type of feedback events to update semantic information store and/or generate Level 3 and Level 1 alert actions.

Level 1 actions are reminders and sanity checks. These type of actions generate an alert to be sent back to the monitored inhabitant (ix). Typically a cell phone or a medical device receives these alerts and notify the inhabitant (x). In case of sanity checks, the inhabitant is asked for his/her response on the occured event (xi) and if exists (xii), inhabitant's feedback is sent back to the CAREACT engine again as an event (xiii,xiv). Both the presence and the absence of the feedback event are processed in the CAREACT engine in internal scenarios to update the semantic information store.

Throughout execution, events, feedback events and actions are logged into the knowledge base after the identity information is anonymized (xv). This knowledge base is crucial in the sense that system experts can evaluate the performance of the existing anomaly detection patterns (xvi) in order to generate novel and better fitting rules via data mining activities (xvii).

3.2. CAREACT Engine

The CAREACT engine is a nondeterministic finite automaton (NFA) based CEP engine that supports actions. Each scenario is a definition of a state machine where state transitions are triggered with incoming events, controlled with value based constraints and may result in actions. Events are drawn from a queue to be executed against every state machine definition. A new state machine instance is created for an inhabitant from the state machine definition whenever neccessary. If the type of the event does not exist in any of the state transitions of the state machine instance, that event is skipped for that specific instance. Otherwise, value based constraints of the outgoing transitions from the current state are validated and if all validations of a transition are satisfied, then actions related to that

transition are executed and the state information is updated. The action interface allows the CAREACT engine to interact with both systems outside the engine and components inside the engine. Examples of interacting with systems outside the CAREACT engine may include calling a web service, running a relational operation on a database and creating a record in a file system. An example of an internal action is creating a future event.

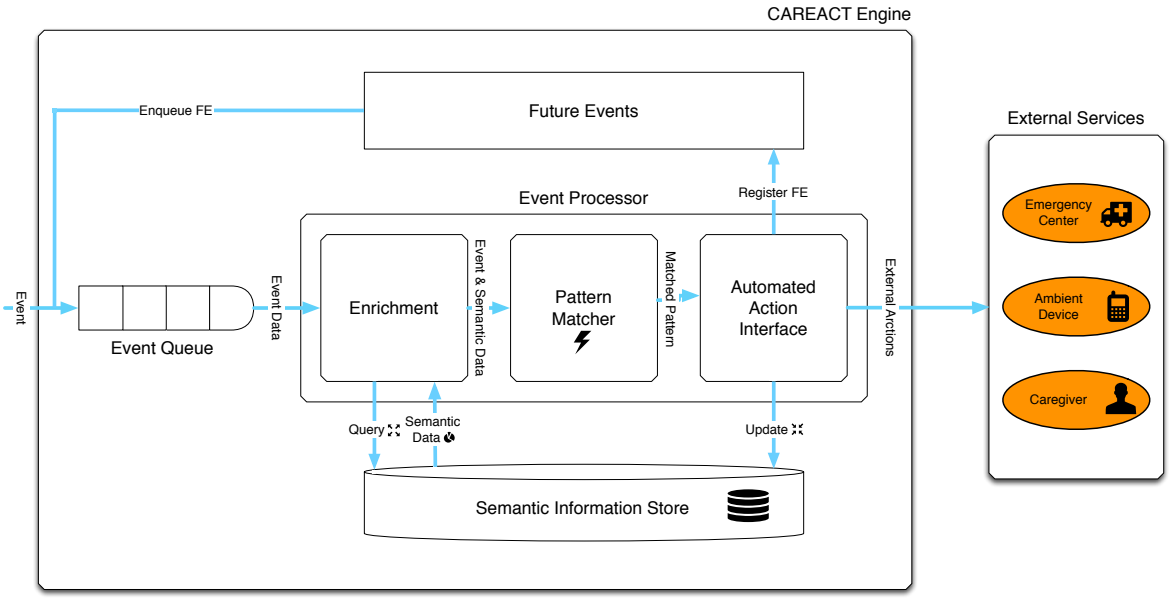


Figure 3.2. The architecture of the CAREACT engine.

At this point, we will discuss the important design decision for the event processing mechanism of the CAREACT engine and then we will continue describing the architecture of the CAREACT engine. With current model, the only means of processing an event is executing the event on the corresponding state machine instance or creating a new state machine instance if a corresponding one does not exist. There are no higher levels of processing mechanisms or no intermediate results of any sort as opposed to the state of the art CEP engines. Such an approach results in constant processing time for every event for every scenario regardless of the complexity of the scenario or the history events executed earlier. While this is clearly an advantage over the state of the art cep engines, such a performance gain comes with a tradeoff. The approach increases memory consumption due to storing a state machine instance for every inhabitant if they ever produce an event. As the second trade off, this approach is not able to cope with the time based constraints without any additional layer of processing. In a sense, this design separates time based constraints from the processing

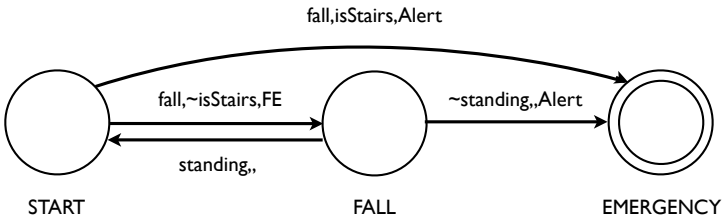
mechanism to provide constant time event processing performance. In this work we provide a new mechanism to support time based constraint, called future events, that does not increase complexity of processing events depending on the value of the time based constraint. The methodology of handling time based constraints will be discussed shortly whereas the implementation of future events will be discussed in detail in the next section.

The CAREACT engine utilizes a semantic information store which is proposed by Wang et al. [36]. There are two interfaces of this semantic repository: Query, Update. Update interface is implemented as an internal action in the CAREACT engine. Query interface is integrated to be used in the validation of the value based constraints. Semantic data are labeled within the engine. Whenever that label is used in a value based constraint, the engine uses query interface to read the value of semantic data. Then the value based constraint is executed with the queried value (Figure 3.2).

The CAREACT engine supports event negation and time window based execution with a combination of future events and explicit scenario design. In Figure 3.3, implementation of the scenario "Generate a Level 3 Alert if an inhabitant falls in stairs or falls elsewhere and does not get back in the next inhabitant-specific time threshold" is presented. The scenario starts with matching an instance of event type *fall*. If an event of the event type *fall* has happened in stairs, the transition *(fall,isStairs,Alert)* will be executed, a Level 3 alert will be generated and state information of this inhabitant will be updated as *EMERGENCY*. If the *fall* event is not in the stairs then the transition *(fall,~isStairs, FE)* and as a result, a future event action *FE* will be executed.

These relations are three tuples, such as the transition relation of *(fall,~isStairs,FE)*. The first component of the three tuple is the name of the event type to match, the second and the third components are the name of the validation and the name of the action respectively. Validation and action parameters are specified in the charts in Figure 3.3. '~' is used for marking a negation in validation.

Validation is composed of three components called *left value*, *compare operator* and *right value*. In this instance, *left value* is location parameter of the event type *fall* denoted by



Validation Definitions

isStairs	left value	operator	right value
	fall.location	in	stairs

Action Definitions

FE	Action Type	Future Event	
	Parameters	Event Name	~standing
		Offset Time	inhabitantTime
Alert	Action Type	Level 3 Alert	
	Parameters	Address	inhabitantAddress

Figure 3.3. Event Negation with the CAREACT engine.

fall.location whereas *right value* is the string "stairs" which is deduced from the integrated ADL system. Validation *isStairs* is defined as location parameter of the event *fall* to be in "stairs".

Action is composed of an *action type* and variable number of *action parameters*. In this instance, *action type* is *Future Event* and parameters of the future event are *~standing* and *inhabitantTime* which is a label for inhabitant specific getting back time. These parameters define the name and offset time of the event that will be scheduled respectively. Future time of the event is calculated by adding an offset time to the current time relative to the time of the executed action. Therefore, the resulting action is the creation of a future event that will be scheduled after inhabitant specific time has elapsed after the execution of action with the name *~standing*.

Subsequently in the execution of the transition, the state of the inhabitant is updated to *FALL*. The created future event action enqueues an instance of event *~standing* to be executed

after the duration of *inhabitantTime* into event queues by future event scheduler. Within that time window, if an instance of event type *standing* is matched, the state machine will return to state *START*. If the event ~*standing* is executed while the state is *FALL*, ~*standing* is matched and the state information is updated as *EMERGENCY* after a Level 3 alert is generated.

3.2.1. Future Events

To address the challenge of processing temporal constraints with large time window sizes, we present a new mechanism called Future Events. Future events use the action infrastructure of a CEP engine to mark a future point in time, which we refer as the 'future event time'. When the 'future event time' arrives, a special type of event is created and fed to the event queues of the CEP engine. The type and parameters of this event are specified during the action configuration. When the action is triggered, an event instance is registered in the future event store. We designed this mechanism to support only one event with the same type and parameters. Multiple events with the same parameters and type are overwritten with a new future event time. We achieve this by storing events in key-value based hash maps where a unique key is created from parameters and the type of the event to be created. This design decision allows the CEP engine to bound the number of future events. After being created, events are registered to future event registrar. This component is responsible for managing uniqueness and staleness of events. The last component in the architecture is the future event scheduler which creates events and enqueues them to event queues once the future event time has been reached.

There are two main approaches for handling events that are registered to the future event scheduler. The first one is maintaining a fully ordered list of events. Then the scheduler can simply wait until the smallest future event time. When the time arrives, the event is dequeued from the list and the corresponding event is enqueued into the event queues of the CEP engine. Registering a new event corresponds to searching the right spot with binary search and inserting the event in the correct queue position. The other approach is putting all events into a pool and maintaining the smallest future event time. When the smallest future event time arrives, the events scheduled to that time are removed from the pool and are enqueued into queues of the CEP engine. Afterwards, the smallest future event time

is recalculated. In other words, the event with the smallest future time is always searched. Registering an event is equivalent to adding one more item to the pool.

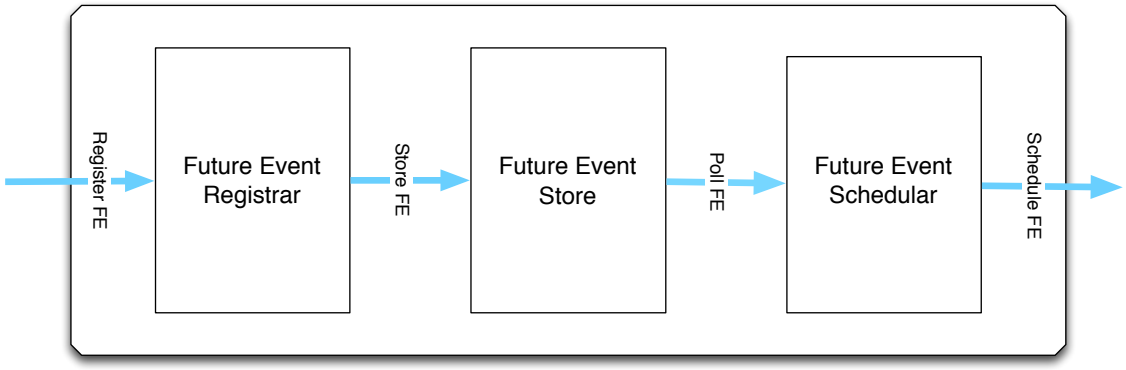


Figure 3.4. The components of the Future Events.

In our implementation, we have chosen a strategy in between these two approaches. We defined time slices of fixed size and registered events to those time slices. We maintain an ordered list of time slices. Within time slices, we search for the event with the smallest future time. There are two reasons for selecting such strategy. The first one is for enabling the CEP engine to relax the time that future event is created. By this relaxation, events from the same time slice are created in the same execution cycle rather than waiting for the exact time each event needs to be created. This is especially useful for scenarios where the time constraints are on a higher order (e.g., days or longer) than the time slices. Such a feature gives rise to the relief that the exact time within a time slice does not affect the logic of the scenario. The other reason lies in the subtleties of wait mechanisms. There are two types of wait mechanisms available. High resolution CPU intensive waits (i.e., busy waits) and lower resolution non CPU intensive waits (i.e., sleep waits). This strategy allows the option of high resolution waits within time slices for scenarios with hard real-time constraints.

Integration of Future Events with the CAREACT engine has two interfaces. The first one is the registration of future events. This interface is integrated as an action to the CAREACT engine. The second one is the scheduling of future events. This interface is integrated with the event queue of the CAREACT engine. Whenever a new event is scheduled from the Future Event Scheduler, it is directly enqueued in the CAREACT engine event queue. (See

Figure 3.2).

3.3. Features supported by CAREACT

In this section, we will describe how the CAREACT supports the features that are introduced in Section 1, namely: interoperability, rule based decision making, adaptivity and analyzability, in more detail. We also discuss, factors that affect overall system performance in related sections.

3.3.1. Interoperability

The architecture of CAREACT is built on the CEP technology over an event driven architecture. This allows every submodule of the sensory system or the activity recognition system to be integrated to our CAREACT architecture by implementing just one interface: Event interface. Thus every submodule that implements an event interface are able to co-exist in this framework and may be able to increase their accuracy by using the data generated by other submodules.

In this type of interoperability where every submodule formats its data and sends it to a central decisioning module, concurrency is an important factor that affects the overall performance in monitoring systems. It might take different amounts of time to form an event out of a real world phenomenon for different systems since each system might use different mechanisms to observe, transmit and process real world data. Such a difference in the event creation time means that the order of events in the real world can be different from the order of events received by the CEP engine. On the other hand, performance of the anomaly detection patterns that use more than one event source in the decision making process are vulnerable to such factors because the CAREACT engine assumes in order delivery of events for sequence calculation. One solution to avoid this problem is to buffer events before using in the decision making process within the CEP engine, effectively resulting in a latency that is equal to the maximum latency of all submodules.

3.3.2. Rule Based Pattern Detection

The CAREACT engine is a rule based engine that supports creation of new rules or modification of existing rules through a user interface. All rules are designed as state machines and deployed to the CAREACT engine where they start executing until a specified end date is reached or indefinitely. An important aspect of this type of architecture is that it supports forward compatibility. Integration of a new submodule does not affect the existing rules and new rules can be generated for a new submodule after it is integrated.

Interoperability and rule based pattern detection collaboratively help us increase the system efficiency by extending current system through adding other subsystems and changing the patterns that detect emergencies without modifying the behavior of the fall detection system. For instance, if we integrate an RFID powered location service, we are able to use locations of the inhabitants in anomaly detection pattern and skip the waiting time before informing the emergency center if the location of fall is stairs.

3.3.3. Adaptivity

Our system employs two types of adaptivity: adaptation to differences between inhabitants and adaptation to changes through time. To achieve adaptivity, we store inhabitant specific data in the semantic information store and use this data in the enrichment process. This effectively results in executing the same rule but with different values.

We update adaptive data using feedback loops. A feedback event is an event generated as a response to sanity checks by inhabitants. Existence/non-existence, timing or parameter values of feedback events are used to adjust adaptive data via algorithms including binary search, running median calculation and direct assignment.

Consider the scenario, where an inhabitant needs to take a pill nightly at 8 pm, and the CAREACT system tries to remind her to take her pills in case she forgets about it. For every individual, the amount of time to make sure that the individual remember her medication may be highly variable. The CAREACT system can adapt to this type of variance by adjusting

the value of the threshold time with user feedback, and hence a single anomaly detection pattern is applicable to every inhabitant in the monitoring environment.

3.3.4. Analyzability

In CAREACT, events, feedback events and actions are logged into the knowledge base with their relationship with each other. This information is the primary source for possible future analysis. While basic reports, like performance of an existing anomaly detection patterns, are built-in within the knowledge base, the information present in the knowledge base can be used for different purposes like finding new anomaly patterns.

To give an example, consider the drug reminder scenario again. This scenario might still perform poorly, due to full stomach requirement of some pills. Namely, the CAREACT system might send the reminder to an inhabitant who is having a meal and did not forget about the medication. This type of false alarm is in fact visible in the knowledge base and can be fetched by querying the percentage of false alarms that do not have a preceding *eating* event. If a system expert finds out that this is the case, then he/she can change the anomaly detection pattern that detects forgetting medication, by adding an *eating* event as a prerequisite.

A very important factor that affects analyzability is privacy and security. Because of their importance, we discuss them in a separate section.

3.3.5. Privacy and Security

In a monitoring system where virtually everything is logged to increase the overall system performance, privacy is an important factor. While anonymity and usefulness almost contradict with each other [46], in our design, we employ two mechanisms that increase the level of privacy.

The first mechanism is logging inhabitant data after anonymization and the second one is the separation of powers. The knowledge base is only open to system experts for their

analysis and the data residing here are totally anonymous. On the other hand, the operational data are not anonymous and are not logged. An instance of this data is only accessible by operators where a Level 2 emergency has occurred. Operator has access to the operational data of that specific emergency to make a decision. Hence, the open data is accessible only case by case and when the case occurs effectively hiding the rest of the user data.

3.4. Architectural Benefits

The architecture of the CAREACT has several distinguished benefits which will be described in this section with solid examples where possible.

First benefit of using the CAREACT system is that it centralizes the anomaly detection logic. This means that whenever a sensory or ADL recognition system is designed, the designer do not have to add any logic that attempts to detect anomalies. They just need to create relevant events for the environment changes and the recognized activities respectively. The system experts will use the CAREACT engine to create anomaly detection patterns by using the activity events and sensor reading events as the building blocks. This type of architecture creates a layered approach to the ambient assisted living, where each submodule is expected to provide the functionality it excels. For instance, in this layered approach, an ADL recognition system that focuses on detecting activities in the kitchen is not responsible for detecting a cognitive disorder of an Alzheimer patient, in case she fires up the kitchen stove during breakfast but leaves the kitchen to take a shower. In the layered approach which is dictated by the CAREACT system, the role of a kitchen ADL recognition system is defined as reporting the recognized activities as events. The CAREACT engine fuses this information with other subsystems to detect anomalies in a context-aware manner which is not possible to accomplish by any submodule itself. Centralizing the logic both increases the efficiency of the overall system and decreases the amount of code that must be written by any submodule.

Even for a centralized anomaly detection process, designing detection patterns that perfectly describe a real world anomaly is not a trivial task and is prone to semantical errors. Some of these errors are usually observable after they are deployed in an AAL environment.

Thus the nature of designing anomaly detection patterns is usually includes an exploration process where anomaly detection patterns are monitored and improved in iterations. For identified errors, the anomaly detection pattern is improved by removing rules that produce false alerts and adding rules for handling previously undetected anomalies of the same type.

The exploration process of finding better fitting anomaly detection patterns against real world anomaly patterns is supported by the CAREACT system which is another benefit of using the proposed system. Rule based pattern detection enables system experts to improve their patterns by adding new rules, removing or changing existing rules within anomaly detection patterns, effectively resulting in better tuned anomaly detection patterns for real life anomaly patterns. Analyzability helps system experts to monitor the affect of the changes they made in the existing anomaly detection pattern in terms of the false alerts it creates or the anomalies it is not able to detect. These two features complement each other for a cycle of exploration iteration which can be repeated until the system experts are satisfied with the accuracy of the anomaly detection pattern.

Furthermore, the rule based pattern detection mitigates problems created by the differences between the AAL environments and between different societies. There are inherent differences in the way people from different cultures behave in executing the same task, like preparing the breakfast, which might be important for detecting a cognitive disorder. In the CAREACT system, patterns can be tailored to fit the targeted inhabitant's behaviors in the anomaly detection by updating the rules within the patterns. Consider a scenario where forgetting to take a scheduled pill is the anomaly which triggers a reminder action. When this scenario is deployed to an environment that hosts patients who needs to take medication via injection, it will create false alerts indefinitely due to not detecting a pill intake. Such a scenario can easily be updated by changing the rule that detects taking pills with using a syringe before being deployed into the second environment. Having such a flexibility is an important benefit of the CAREACT system, which improves the overall system applicability.

This flexibility also plays a role in adapting the existing rules to new environments that do not have the same set of subsystems. Going back to the reminding medication scenarios and assume existence of a smart pill bottle that ensures a drug has been taken from it, such

an information can be used as the taking medication event for the CAREACT engine. If this type of a subsystem does not exist, taking medication event can be replaced by the sequence of events of opening the drawer, turning the faucet on, turning the faucet off and closing the drawer in existence of the binary sensors that feed the respective information. This sequence assumes that the pills were in the drawer and the inhabitant opened the faucet to drink water while taking the pill whereas the same sequence might be the result of tooth brushing. Hence, this approach is expected to create higher false alert rates than having a smart pill bottle, but still provides significant information in case the inhabitant does not open the drawer. The CAREACT system provides the flexible infrastructure if a system expert decides to use medication reminder scenario on the environments that do not have smart pill bottles.

Another benefit related to having a rule based pattern detection support is that when the system is extended by adding a new submodule, it is possible to update existing anomaly detection patterns to use the information supplied by the new submodule resulting in better anomaly detection patterns in terms of recognizing the real world anomalies. An example of such an improvement is using location of the fall in deducing emergencies after adding location services in an AAL environment that has a fall detection submodule in place. Integration of a new submodule to the CAREACT system is easy because it means implementing only one interface, namely the event interface, rather than integrating it to every other subsystem within the AAL environment. To sum up, it is easy to integrate a new submodule to the CAREACT system, as well as using the data from such a submodule in anomaly detection pattern design which means that the CAREACT system has the benefit of being extendable.

Yet another benefit of using the CAREACT system is that it is able to adapt itself. This ability is used to handle changes between different inhabitants and changes that happen over time. The CAREACT system has the ability to capture feedback events which are created as a result of the sanity checks and the reminders produced as a part of the anomaly detection pattern. These feedback events are used to change the values of parameters that are used in the anomaly detection pattern effectively adapting it to the time and the inhabitant. An example of such an adaptive scenario is to remind inhabitants with chronic diseases who forget to take their medicine. The amount of time to make sure that the inhabitant indeed forgot taking her medicine is different for every inhabitant. The CAREACT system adapts

the parameter that corresponds to this real life value for every inhabitant. This way the CAREACT system is able to decrease the false alerts and increase the quality of the AAL experience of the inhabitants.

Since the CAREACT engine is built on the CEP architecture, it has a higher performance than traditional rule based approaches. Such a performance is important in reacting to emergencies in real-time under heavy load. The massive amount of data that can be produced by sensory systems puts strict performance requirements for monitoring systems in AAL context. It is particularly important to work in (near) real-time, because there are emergencies in the context of the AAL environment in which seconds earlier detection might save an inhabitants life. An example of such an anomaly is the heart attack. The CAREACT system works in real-time under heavy load because it employs the CEP architecture where an event is processed in the order of milliseconds which is negligible compared to time frames of emergencies that occur in the healthcare domain [1].

The final benefit of using the CAREACT system is that it creates a common language between users of the system. Since the patterns are designed as finite state machines via a graphical user interface (See Figure 3.5), designing patterns is easier than writing the respective code and also patterns designed by others are easy to analyze. People with simple understanding of the finite state machine, or people who can understand UML flow charts can describe, analyze and change anomaly detection patterns within the CAREACT system. The graphical representation of an anomaly detection pattern is a common language which makes communication between medical experts from different domains, system operators and system experts easier. This communication minimizes the risk of designing anomaly detection pattern that detects a wrong real life phenomenon.

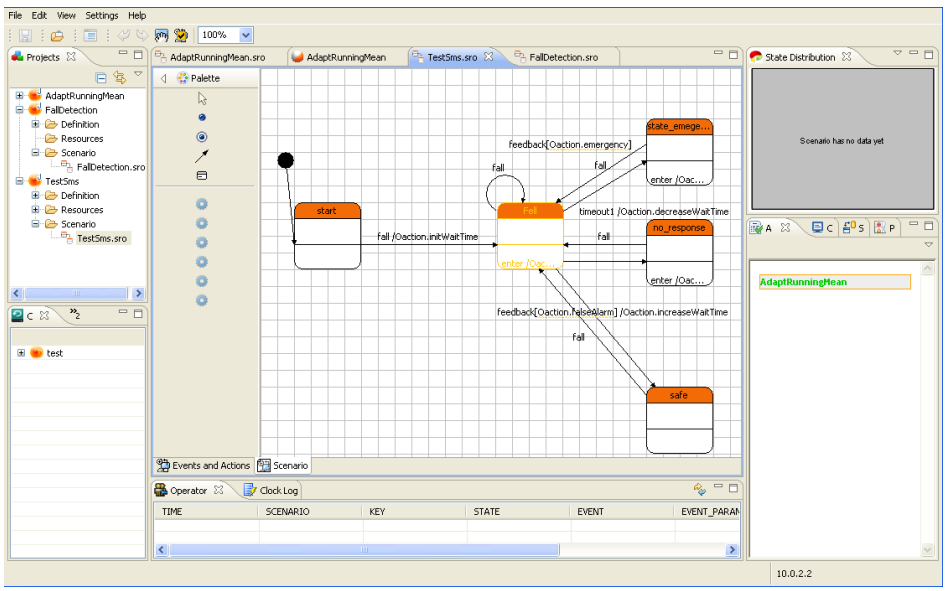


Figure 3.5. Graphical user interface of the CAREACT engine.

4. EXPERIMENTS AND RESULTS

We implemented our proposed architecture in Bogazici Univesity, Kandilli Campus testbed as a part of WeCare framework [47]. This chapter describes the results and the setup used for our implementation.

To validate our system, we conducted two types of experiments. In the first group, we worked on artificially created data where the proposed architecture is evaluated from adaptivity and analyzability perspectives. In the second group, we made a real life implementation by integrating a fall detection algorithm to the CAREACT to showcase applicability of the overall architecture.

4.1. Demonstration of Adaptivity

To validate adaptivity, we conducted three experiments. In each of the experiments, the CAREACT system adapts the value of a parameter to match the generated value of the same parameter which can correspond to a threshold in a real life scenario, like the maximum allowable inactivity period before creating an alarm for an elderly. When an event including such a parameter is generated, the CAREACT engine creates a sanity check. Positive feedback events, like *I am OK*, increase the value of parameter whereas negative feedback events, like *I don't feel good*, decreases the value of the same parameter.

In the first experiment, the threshold data are generated for 1,000 inhabitants from a Gaussian distribution with the mean and the variance equal to 10 and 1 respectively. The system is initialized to learn just one parameter and the starting value is set to the mean of the normal distribution, 10, for each inhabitant. In each iteration, the step size which is used to increase or the decrease the value of the parameter is halved and the starting step size value is set to the variance of the Gaussian distribution. With this schema, the CAREACT engine actually makes a binary search for generated threshold value of every inhabitant.

In the second experiment, 500 values are generated from two Gaussian distributions

each with parameters (mean:10, variance:1), (mean:5, variance:2) to create a mixture of Gaussian distributions. The system is initialized with a starting value of 7.5 (mean of means) and the step size of 1.5 (mean of variances).

In the last experiment, it was assumed that parameters of distributions from the second experiment are known at the time of designing the adjusting scenario. This experiment aims to evaluate if and by how much the CAREACT architecture can make use of a-priori data about an adaptive parameter. In this scenario, half of the inhabitants have a starting value of 10 with a step size of 1 whereas the other half has a starting value of 5 with a step size of 2.

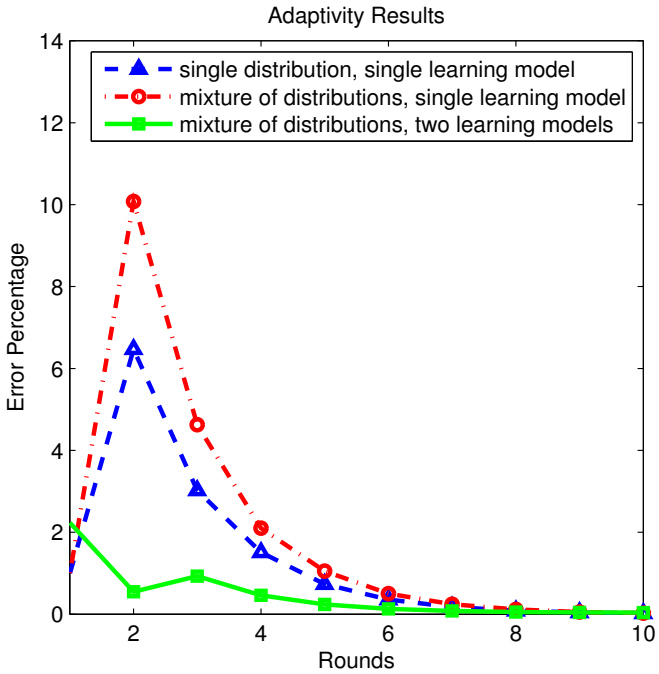


Figure 4.1. Error percentage vs Number of iterations.

The results are presented in Figure 4.1. In all experiments, the system adjusted the values of parameters to their generated values in a convergent manner. In Experiment 1, at average, 5 iterations were necessary to decrease the percentage of difference between generated and adjusted values of parameters under 1%. In Experiment 2, where we used parameters of a single Gaussian distribution to adapt to a mixture of Gaussian distributions, the errors were higher and it took 5 iterations on average to decrease the percentage error under 1%. In the third experiment, we were able to observe that the more a-priori knowledge

the system had, the sooner it converged. This time, we used the parameters of two Gaussian distributions to adapt to the same type of data from Experiment 2 and it took 3 iterations to decrease the error percentage under 1%.

Experiments showed that our proposed architecture is able to adapt its parameters. Furthermore, a-priori knowledge of the distribution of the environment parameters can be used to design better adapting scenarios.

4.2. Demonstration of Analyzability

To demonstrate the analyzability capabilities of the CAREACT system, we worked with a real life like scenario on artificially created data. In the scenario, when an inhabitant forgets to take his/her medication, a reminder is sent. If the inhabitant sends a *Later* feedback event, the reminder is marked as a false alarm. If an inhabitant sends *Done* feedback event, the reminder is marked as a successful reminder and the time between the actual medication time and reminder time is logged into knowledge base as the reporting latency.

In this scenario, two adaptive parameters are used. The first one is the *mean_timeout* (mean timeout before sending a reminder). Lower timeout values increase the false alarm rate while decreasing the reporting delay, whereas higher timeout values decrease the false alarm rate while increasing the reporting delay. The second parameter is *stdev_timeout* (standard deviation of the timeout). The value of this parameter is multiplied by a factor (F) and then added to the *mean_timeout* to calculate the *timeout* (timeout before sending parameter).

We used three algorithms in three sets of experiments to adapt two parameters. The first two algorithms, additive increase multiplicative decrease (AIMD) and multiplicative decrease linear increase (MDLI) only adapt the *timeout* and require only binary(positive/negative) feedback events for adaptation process. The third algorithm, running standard deviation (RSD), adapts both parameters but requires absolute values of the measured timeout in feedback events.

We generated data with the following characteristics. Every inhabitant completely for-

gets taking medication with 0.25 probability. The mean time to remember taking medication is a random variable with a mean of 60 minutes and a variance of 60 minutes for each inhabitant. Actual time to remember taking medication is another random variable with the mean and the variance equal to the mean time to remember taking medication.

Table 4.1. Performance results of different adaptation algorithms for same scenario

Algorithms	**False Alert Rate**	**Reporting latency (min)**
No Adaptivity	50%	60
AIMD	24.16%	110.22
MDLI	46.30%	56.03
RSD, F= 3	5.23%	160.03

The results from the knowledge base are presented in Table 4.1. Results show that different algorithms can be employed to fine tune the false alert rate and the latency of the overall system. For scenarios where handling a false alarm is costly, like calling an ambulance, algorithms with low reporting latency can be employed at the cost of an increased overall latency. For scenarios where handling a false alarm is cheap or latency is critical, a low latency adaptation algorithm can be selected at the cost of an increased false alarm rate.

This is the main point of having an analyzability module in our proposed architecture. System experts can analyze these data to find the best fitting adaptation algorithms for their needs. Thus allowing better overall efficiency.

4.3. Demonstrative Implementation

We implemented a real life use case of CAREACT by integrating it to a fall detection system [47] in our laboratory environment. The existing fall detection system works on the smartphone platform to monitor daily lives of inhabitants for fall detection purposes. When the system detects a fall, it provides the ability of confirming the fall or marking it as a false alarm to the inhabitant via the user interface. If the final decision is not a false alarm, the system connects to the preconfigured emergency center.

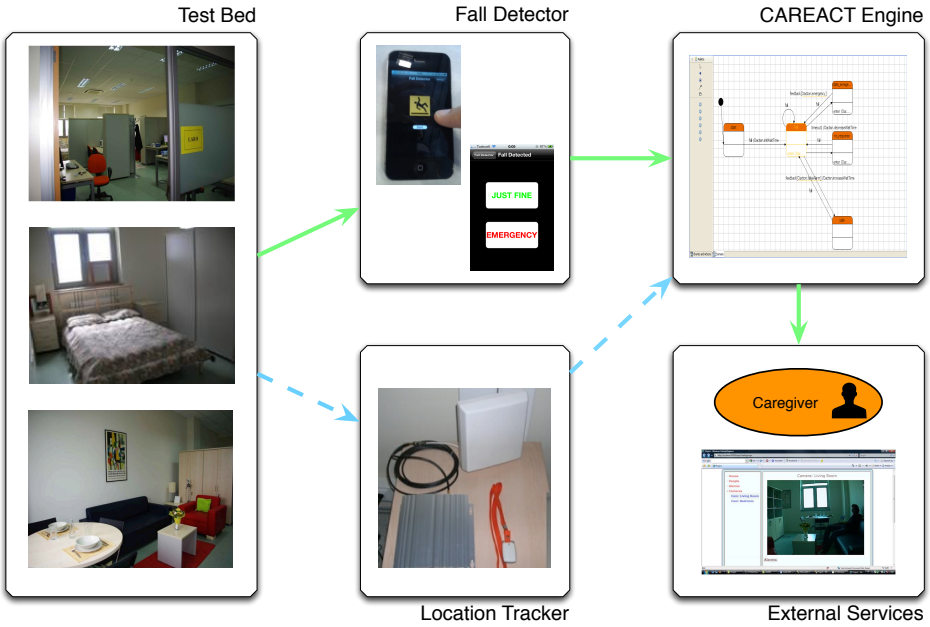


Figure 4.2. In our laboratory environment, we integrated our proposed architecture with an existing fall detection algorithm [47].

In our implementation, we improved this use case by integrating the fall detection system to CAREACT and designed a scenario that can detect false alerts after a fall is detected. This scenario introduces a waiting time threshold for collecting feedback events from the inhabitants. If there is no feedback within such threshold after a fall is detected, it is deduced as the inhabitant is unconscious and the CAREACT system connects to emergency center. If the inhabitant gets up within the specified threshold, the system treats the fall as a false alert.

The integration required two easy modifications on the existing solution. As the first modification, the fall detection system sends the *fall* event directly to the CAREACT system rather than asking for the inhabitant feedback. As the second modification if the CAREACT system sends a sanity check event, asking for the inhabitants feedback, it is represented as user interface choices to the inhabitant.

When a *fall* event is received by the CAREACT system, the inhabitant's personal information is accessed from semantic information store which include the average get up time after falls and the age of the inhabitant which are used as parameters on executing the

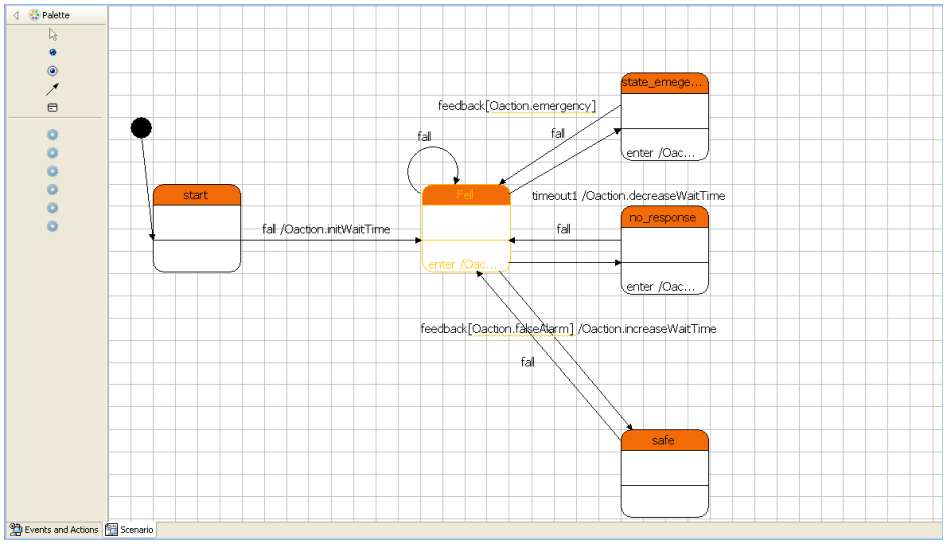


Figure 4.3. Screenshot of the scenario definition screen of the CAREACT engine.

described logic. (See Figure 4.3 for the designed scenario).

4.4. Performance Evaluation of CAREACT Engine

We evaluate the performance of the CAREACT engine to assess the appropriateness of CAREACT engine for typical use cases of the proposed framework which includes monitoring daily activities monitoring and anomaly detection. Since typical use cases include monitoring environments from a hospital or a multitude of homes to a single home and monitoring capabilities that produce a couple of events like per hour to several events per second we selected values from different orders.

We identified five parameters that might have an effect on the performance of the CAREACT engine, namely, the number of the monitored inhabitant, the number of events per inhabitant per second, the number of concurrent scenarios, the number of event types used and finally the number actions per event. We conducted two sets of experiments, in the first set, we identified which parameters have an impact on the performance whereas in the second set we measured the effect of the each parameter that has an impact in detail. We measured three metrics to evaluate the performance of the system, the maximum throughput, the memory consumption and the latency.

Table 4.2. Parameter set of the first experiment set.

Parameter	Description	Values
num_inhabitants	Number of the monitored inhabitants	1,100,10000
num_events_per_inhabitant	Number of events per inhabitant per second	0.01,1,100
num_concurrent_scenarios	Number of concurrent Scenarios	1,2,10
num_event_types	Number of event types used	1,2,10
ave_num_actions_per_event	Number of actions per event	0,0.5,1

All the experiments were performed on amazon elastic cloud computing standard large instance running Oracle SE 1.6.0_33 Java virtual machine (JVM) on Ubuntu 12.04 operating system. We set the JVM maximum and minimum heap size to 2 GB, so that the virtual memory allocation activity had no influence on the results. To test the system, we implemented an event generator that creates stream of events according to a Poisson process. In each of the experiments we sent 6 batches of 100,000 events to the CAREACT system. First batch makes sure that all the startup initializations are finished and do not have an effect on the steady state runtime behavior of the CAREACT system. The following batches are the repetitions of the same experiment and are started after all the events from previous batch has been completely processed. The scenarios used in the experiments are identical and designed with a self transition loop in the state machine so that every processed event results in a transition in every scenario. The number of event types are the number of self transition loops in a given scenario. The number of actions are controlled with the value based parameters. For every event, we sent a dummy parameter whose value is uniformly distributed between 1 and 10. If this value is smaller than a threshold for an event, then an action is executed. The value of this threshold controls the number of the actions per event parameter. The action itself is a number crunching operation which takes a couple of microseconds. The selection of this CPU bound action is for practical purposes although sleeping the thread is a better fit for simulating the workload of typical operations like sending a packet over network, writing to a database or to a file. While Java supports nanosecond sleeping, the JVM only guarantee that the sleep operation will take at least specified amount of time which makes overall action execution time vary wildly when a lower priority thread, mainly garbage collector

thread, requests for CPU cycles. We selected the CPU bound action to mitigate the problems caused by unguaranteed wake up time of thread sleeps. We present the values we used in the first set of experiments in The Table 4.2. For this set, we did not experiment on the cartesian product of the possible values of the each parameter. We varied one parameter while using a fixed value for the rest of the parameters, which are underlined in the table.

Table 4.3. Parameter set of the first experiment set.

Parameter	**Throughput & Latency**	**Memory Consumption**
num_inhabitants	Partial effect	Affects
num_events_per_inhabitant	Partial effect	No effect
num_concurrent_scenarios	Affects	Affects
num_event_types	No effect	No effect
ave_num_actions_per_event	Affects	No effect

There are a number of observations from this set of experiments. The parameters can be classified into two categories, those that affect the latency and the throughput together and those that affect the memory consumption. Because of this fact, we examine the affect of the parameters in two perspectives, from the memory consumption perspective and from the latency and the throughput perspective. We summarized the relationship of the parameters and the performance metrics in Table 4.3.

From memory consumption perspective, only two parameters have an impact, *num_inhabitants* and *num_concurrent_scenarios*. The memory consumption of the CAREACT engine increases linearly with the increasing number of the monitored inhabitants. The increase is ~240 KB for 10,000 inhabitants. This is an expected outcome since the CAREACT engine creates a new state machine instance for every inhabitant which has a fixed memory cost. The effect of increasing the number of scenarios is similar to increasing the number of inhabitants. The memory consumption increases linearly with increasing the concurrent number of scenarios. This behavior is also an outcome of creating a new state machine instance for every inhabitant. For every scenario, every inhabitant have one state machine which means a linear increase in the memory consumption for increasing either the number of monitored

inhabitants and the number of concurrent scenarios.

From the throughput and the latency perspective, the initial experiments show that increasing *num_concurrent_scenarios* increases the latency and decreases the maximum throughput whereas *num_event_types* do not have a measurable impact on the these two performance metrics. This is a natural outcome of the state machine based execution of the system which is explained in Section 3.2. A new event type creates exactly one more if statement for the state it transits from, which is insignificant compared to the other processing components of the CAREACT system. Number of actions per event also affects the performance of the CAREACT system. When one increases the number of actions per event, the latency increases whereas the throughput decreases. The number of the monitored inhabitants and the number of events per inhabitant parameters do not have a direct effect as long as the total number of the events sent to the CAREACT engine remains the same. These two parameters together form the number of the events sent per second and before proceeding to the detailed experiments, we isolated this parameter so that we can understand which values make sense for *num_sent_events_per_second* in the detailed experiments.

We observe that the effect of increasing *num_sent_events_per_second* increases the throughput up to a threshold, which is the maximum throughput of the engine that is equal to ~25000 events/second for 5 concurrent scenarios. After this threshold, further increasing the *num_sent_events_per_second* does not increase the throughput because the CPU usage hits 100%. This is the limit for throughput and the rest of the sent events are backlog events and are put to queue. Since there is no event dropping in the CAREACT engine, these events are sooner or later processed unless the engine crashes due to queued events consuming all the available memory. In the detailed experiments we want to observe latency and maximum throughput while varying other parameters. But the current experiment setup which includes sending fixed number of events at a rate higher than maximum throughput of the CAREACT engine, which is called bursting in this context, skews the latency results due to the latency of the backlog events. To solve this problem, we decided to divide the experiments into two sub categories. In the first category, which we call steady state experiments, we adjust the *num_sent_events_per_second* parameter so that there is 1 event on average in the event queue. This way our observations will show the effect of changing different pa-

rameters of the system on the maximum throughput and on the latency without the skewing effect of the bursting. In the second category, which we call bursting experiments, we send a bulk of events as rapid as possible and calculate the additional latency introduced so that we can analyze the applicability of the CAREACT framework for environments that has bursty time spans. The impact of the number of events sent in one burst on the performance of the CAREACT engine are investigated in the second set of experiments.

Table 4.4. Parameter set of the *steady state experiments*.

Parameter	**Values**
event_rate	Adaptive
num_concurrent_scenarios	5, 10, 15, 20, 25
ave_num_actions_per_event	0, 0.2, 0.4, 0.6, 0.8, 1

The values that are used for the steady state experiments are presented in the Table 4.4. Unlike the first set of experiments, in the steady state experiments we conduct an experiment for every possible combination of the values of the parameters. For this experiment we have implemented a new event simulator which takes the number of average queue size of the previous repetition as an input. It starts with 1,000 events/second and increases load by with step size of 1,000 events/second. if the average queue size within 0.95 and 1.05 it repeats the experiment with the event rate until the there are 5 consecutive repetitions with average queue size within 0.95 and 1.05. Otherwise if the average queue size is below lower threshold it increases the load by the step size else it halves the load and starts decreasing the load. This time same rules apply except if the load is less than the 0.95 it halves the step size and starts incrementing again. With this schema event simulator find the event rate that creates average queue size around 1 and with that event rate makes 5 repetitions for every possible combination of the parameter set.

The impact of *num_concurrent_scenarios* and *num_actions_per_event* on throughput under steady state is presented in Figure 4.4. As can be seen from the figure as the value of either increases the throughput under steady state decreases and the relationship is inversely proportional. Another observation we make from the figure is that the effect of

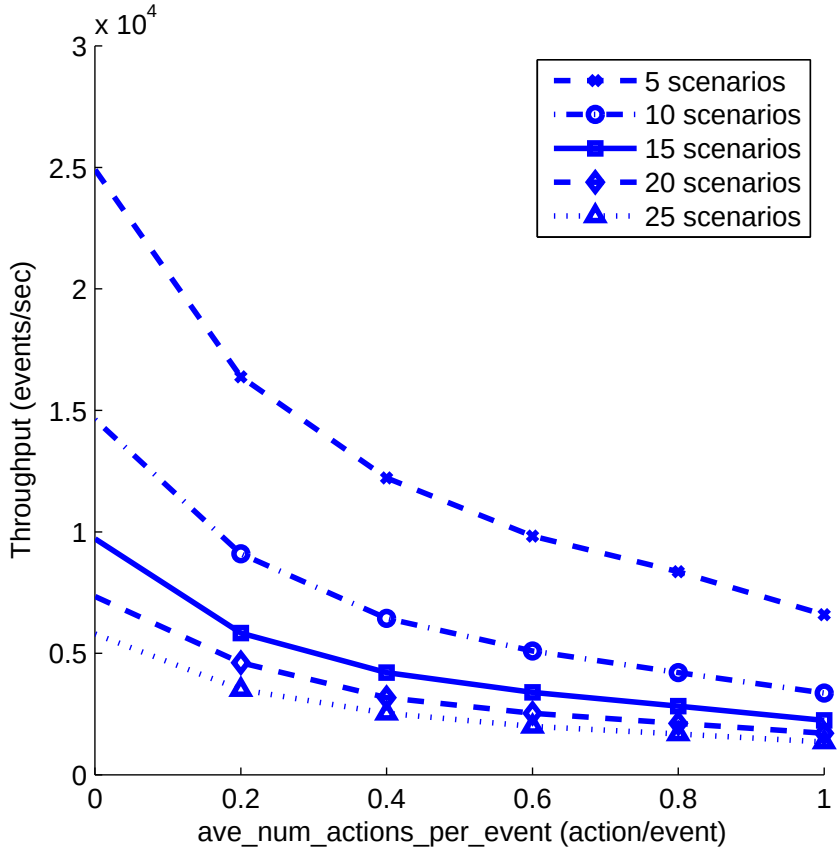


Figure 4.4. The effect of increasing ave_num_actions_per_event on throughput for different number of concurrent scenarios.

num_concurrent_scenarios is higher than the effect of *num_actions_per_event*. We conclude that deciding on the action is harder than actually running the action in the CAREACT engine. The impact of same parameters on latency is presented in Figure 4.5. In terms of latency, the effect of increasing the*num_actions_per_event* increases the latency linearly. This is an expected outcome because as the *num_actions_per_event* increases the time spent on executing the action linearly increases. The effect of increasing the *num_concurrent_scenarios* is linear as well. This is an expected outcome since the execution of an event on concurrent scenarios is sequential and the scenarios are identical. An event is executed in every scenario one by one and the latencies from each scenario add up. In a real life setting it is possible to have anomaly detection patterns with different input events and different action per event values because there will be scenarios that will focus on different aspects of daily

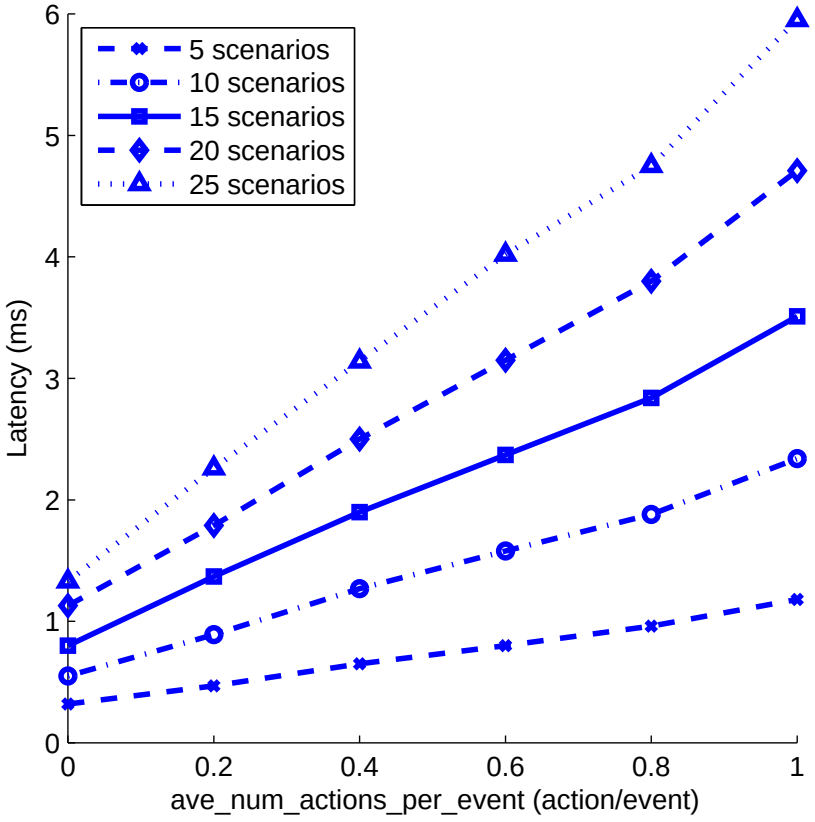


Figure 4.5. The effect of increasing rate on latency for different number of concurrent scenarios.

living which translates to using different event sources. This implies that the performance impact of adding an anomaly detection pattern will not be linear, it will depend on the event sources and the actions per event of designed anomaly detection patterns. An important observation from the steady state latency results is that the computation delay introduced by the CAREACT engine is in the order of milliseconds which is comparable or less other delay sources in an AAL environment like network communications and activity recognition and it is negligible in the context of ambient assisted living.

In the bursting experiments, we send *burst_load* of 1,000, 10,000 and 100,000 events to the CAREACT engine with the same set of values for *num_concurrent_scenarios* and *num_actions_per_event* parameters and we measure maximum latency of all the events. Ex-

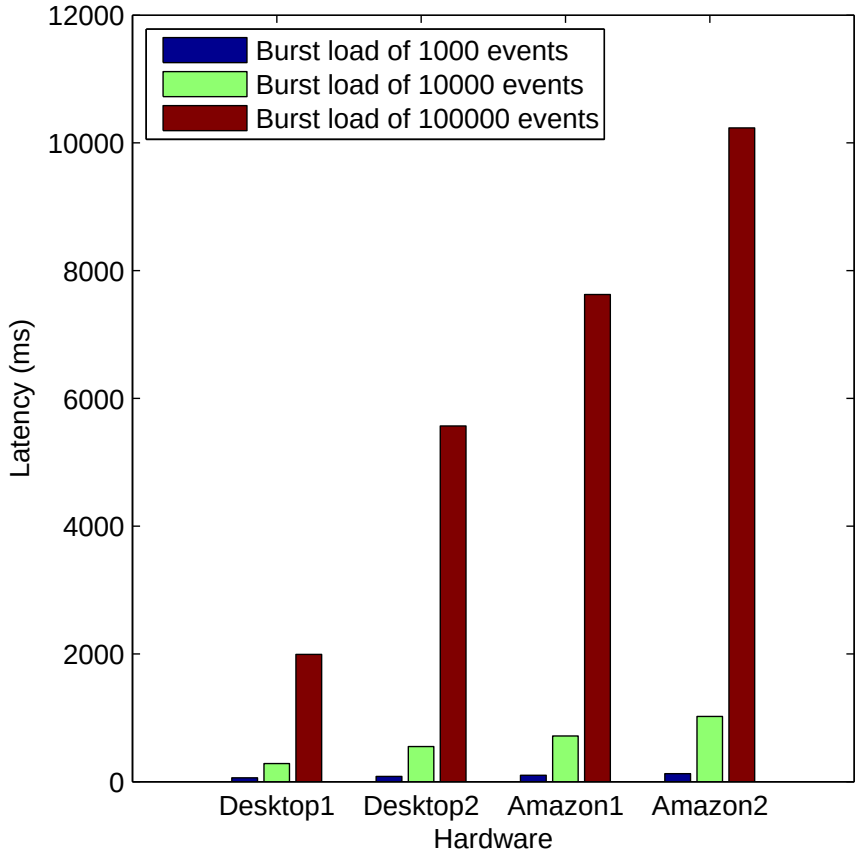


Figure 4.6. The effect of burst loads on different hardwares.

periments show that the relationship of the *burst_load* and the latency is linear and we observe that for 100,000 events it is possible to observe aroud 40 seconds maximum latency under *num_concurrent_scenarios*=25 and *num_actions_per_event*=1 for our previously described test hardware. While the 40 seconds latency might be reasonable for some if not most of the anomaly detection scenarios in the context of AAL we expand our experiments to different hardwares so analyze if it is possible to improve results by changing hardware. We used 4 types of machines, 2 desktops and 2 on demand instances of different sizes from Amazon Web Services. The first desktop has 4 Intel 2.8 GHz i7 cores whereas the second one has 2 Intel 2.23 GHz i5 cores which we named Desktop1 and Desktop2 respectively. The amazon instances are c1.medium which has 5 cores and m1.large which has 4 cores. The clock speeds are undocumented but same for all the Amazon Web Services instances. We named the c1.medium instance as Amazon1 and the m1.large instance as Amazon2. The

results for *num_concurrent_scenarios*=5 and *num_actions_per_event*=1 are presented in Figure 4.4. We observe that both clock speed and the number of cores have a positive impact on the maximum latency by comparing Desktop1 to Desktop2. Desktop1 has twice the cores and higher clock speed than the Desktop2 and the latency for the Desktop2 is almost 3 times the latency of Desktop1. The effect of changing only the number of cores is visible from the results of Amazon1 and Amazon2. The additional core to existing 4 decreases the maximum latency by 20% from $\tilde{1}0$ seconds to $\tilde{8}$ seconds for 100,000 events. We conclude that increasing clock speed decreases latency by decreasing the execution time of an event whereas increasing the number of cores decreases the latency by higher level of parallelization. Hence depending on the work load and the latency requirements of the CAREACT scenarios it is possible to select a hardware by sizing the computational power vertically by increasing the clock size or horizontally by adding new cores from the benchmark we present in Figure 4.4. We also share the bursting throughputs of these machines in the Table 4.5.

Table 4.5. Bursting throughput for different hardwares.

Hardware	**Bursting Throughput (events/second)**
Desktop1	26,542
Desktop2	9,062
Amazon1	6,573
Amazon2	4,744

To sum up, the CAREACT engine is able to process thousands of events per second on tens of concurrent scenarios while executing as high as 1 action per event with commodity hardware. Under steady state the latency is in the order of milliseconds and which is comparable or less other delay sources in an AAL environment like network communications and activity recognition and it is negligible in the context of ambient assisted living. When loaded with event rate over the maximum capacity of the CAREACT engine over short bursts of time the latency may go up to 40 seconds which may still be acceptable for most of the targeted scenarios for the CAREACT framework. Furthermore, for scenarios with stricter requirements both horizontal and vertical sizing is possible to decrease the latencies for burst loads. We conclude that the CAREACT engine is a good fit for the CAREACT

framework which targets scenarios ranging from long term monitoring activities of daily living to anomaly detection.

4.5. Performance Evaluation of Future Events

In this section we evaluate the performance of future events and compare the experimental results with theoretical analysis. The central part of the analysis is the size of the time window size in temporal constraints because as it grows the throughput decreases for the state of the art literature [17] based on the reasons we explained in Section 2.2. To such end, in our experiments, we varied the size of the time windows from 1 second to 30 seconds and measured throughput, memory consumption and latency. We have fixed the number of concurrent scenarios to 5, the number of actions per event to 0.1 and the rate to 1.

From throughput perspective, we have observered that increasing the size of the time window does not have an effect. The throughput throughout the experiments was almost constant at ~2510 events/second. There is no CEP engine in literature that does not have a decreasing performance when the time window sizes in the pattern matching sequences increase. These results show that the future events change the paradigm of processing the time windows within patterns in a way that it does not depend on the size of the time window.

From memory consumption point of view, increasing the time window size did not have an effect. The future events mechanism consumed additional memory of ~200 KB for every experiment. The reason is that after the system starts running in steady state, there is exactly one future event for every monitored inhabitant, which does not depend on the size of the time windows on temporal constraints.

From latency perspective, increasing the size of the time windows decreases the latency because the number of events fired within the CAREACT engine decreases thus adding additional workload. The number of fired events increase when they are not postponed by an event that schedules a future event. When the time window size increases, the probability of not receiving a postponing event decreases, in our setup they are reversely proportional. We present the results of the experiments together with the analytical counterpart in Figure 4.8.

We used queueing theory, in particular Jackson's Theorem for open queueing networks, to analyze the overall performance of the future event mechanism. From a queueing perspective the system is presented in Figure 4.7.

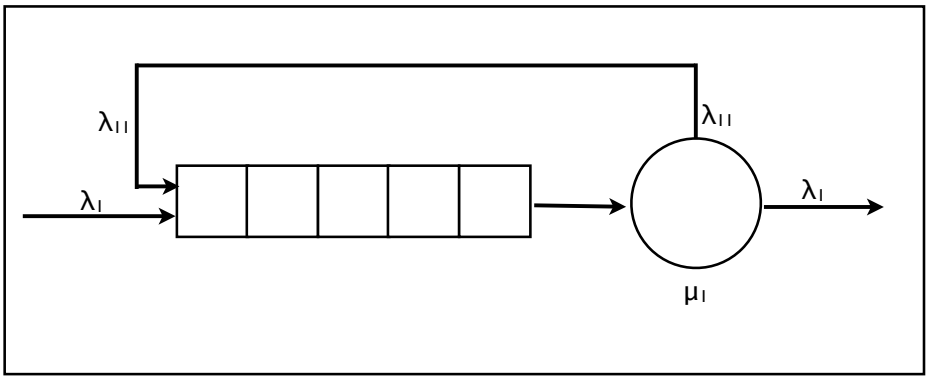


Figure 4.7. Modelling of a future events in the CAREACT engine.

Jackson's Theorem provides a general product-form solution for both feed forward and feedback open queuing networks and the CAREACT engine is a feedback open queueing network, where the future events are the feedback mechanism. There are 4 assumption for Jackson's theorem which are:

- the network is composed of *K* first come first serve (FCFS), single-server queues
- the arrival processes for the *K* queues are Poisson at rate $r_1, r_2...r_K$
- the service times of customers at jth queue are exponentially distributed with mean $1/\mu_j$ and they are mutually independent and independent of the arrival processes
- once a customer is served at queue *i*, it joins each queue *j* with probability P_{ij} or leave the system with probability $1 - \sum_{j=1}^{K} P_{ij}$ where P_{ij} is the routing probability from node *i* to node *j*.

In the CAREACT, the system is composed of 1 FCFS single server and in the experiments we have used a Poisson process for sending events. The events either create a future event with the probability P_{11} or exit the system. The only assumption that does not perfectly hold for the CAREACT engine is the third assumption. The service time of each event has two components, the time to process event in the state machine and the time to execute

an action if an action is triggered as a result of the state machine execution. The first component has constant complexity whereas the second is distributed exponentially. Since the more dominant componant of the two is action execution time, we simplified the complexity of real distribution with by assuming it is an exponantial distribution with the same mean.

Under these assumption, Jackson's theorem states that the number of customers in the queue *j* is

$$E[N_j] = \frac{\rho_j}{1 - \rho_j} \tag{4.1}$$

where

$$\rho_j = \frac{\lambda_j}{\mu_j} \tag{4.2}$$

and

$$\lambda_j = r_j + \sum_{i=1}^{K} \lambda_i \cdot P_{ij} \tag{4.3}$$

Applying Little's law results in average response time, in our case average latency.

$$E[R] = \frac{1}{\lambda} \cdot \sum_{j} E[N_j] \tag{4.4}$$

In the case of the CAREACT engine, we have just one queue and the probability of the feedback, P_{11} depends on the future event time, *t* with

$$p_{11} = \frac{1}{t+1} \tag{4.5}$$

substituting Equation 4.5 in Equation 4.3 and solving for one queue system yields

$$\lambda_1 = \lambda + \lambda_1 \cdot \frac{1}{t+1} \tag{4.6}$$

solving for λ_1

$$\lambda_1 = \lambda \cdot \frac{t+1}{t} \tag{4.7}$$

substituting Equation 4.1, Equation 4.2 and Equation 4.7 in Equation 4.4 results in

$$E[R] = \frac{1}{\lambda} \cdot \frac{\frac{\lambda \cdot (t+1)/t}{\mu_1}}{1 - \frac{\lambda \cdot (t+1)/t}{\mu_1}} \tag{4.8}$$

$$E[R] = \frac{1}{\frac{t}{t+1} \cdot \mu_1 - \lambda} \tag{4.9}$$

The comparison of the experimental results with the analytical results calculated from Equation 4.9 by substituting 4,800 for μ_1 and 2510 for λ, which are measured values for experiments are presented in Figure 4.8.

These results show that they have similar trends but with different characteristics. The analytic results are much more smooth and start from a higher latency value for smaller time windows and end up in lower latency values compared to the experimental results. The difference in smoothness is partly caused by the measurements and partly caused by the garbage collections, both of which are inherent to java programming language. The difference in the trends are caused by the assumptions made on analytical modelling. Given every assumption holds for our system with the exception of the third assumption, we conclude that the difference between trends caused by the difference between the actual distribution and the assumed exponantial distribution for processing an event.

4.6. Methods for solving interoperability related latency

It is common for an activity recognition system to report the detected activity certain time after it happened in the monitored environment. The reason is that the statistical models that recognize activities usually use sensor readings after the event happened in addition to the readings before the event in calculation. Consequently, there is a gap between the

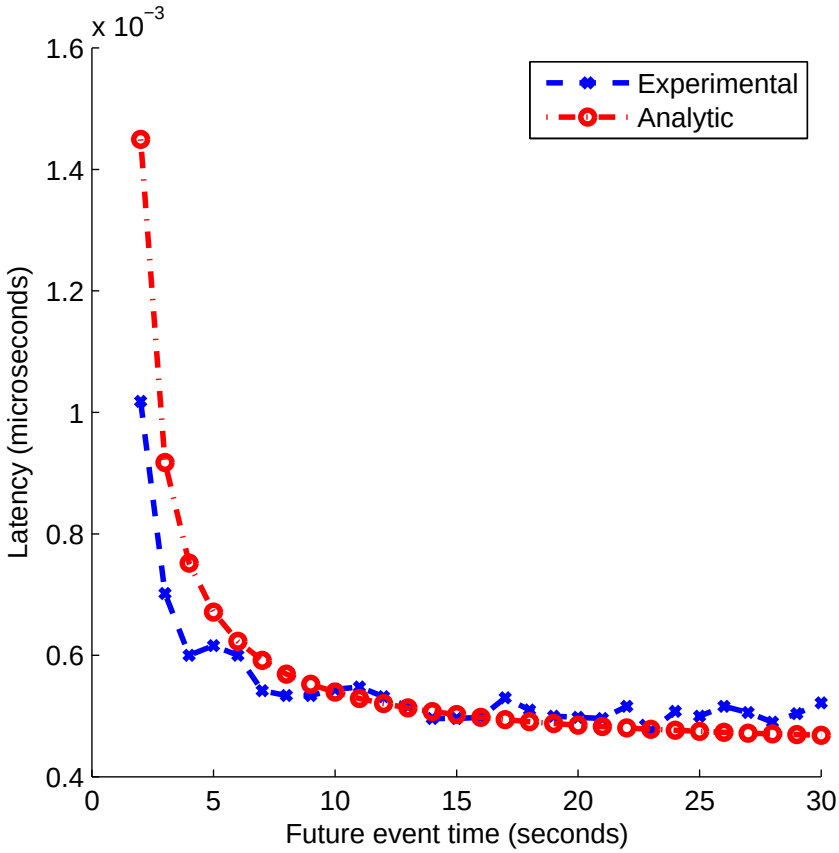


Figure 4.8. Experimental vs analytical results for the effect of the size of time windows on latency.

time event happened in real world and the time it is recognized which is called the *recognition latency* in this context. For scenarios where events having different latencies are used together in sequences, having different latencies may reduce the accuracy of the pattern detection. For instance, if a scenario uses has a pattern of event A followed by event B where the *recognition latency* of event A is higher than that of event B, real world occurences of event A followed by event B may be missed in cases where the *recognition latency* of event A is larger than the amount of time between the two events. This phenomenon is described in the top two rows of Figure 4.9.

In scenarios where events with different latencies are used together, the real world sequence and the sequence perceived by the CAREACT engine may differ, which decreases

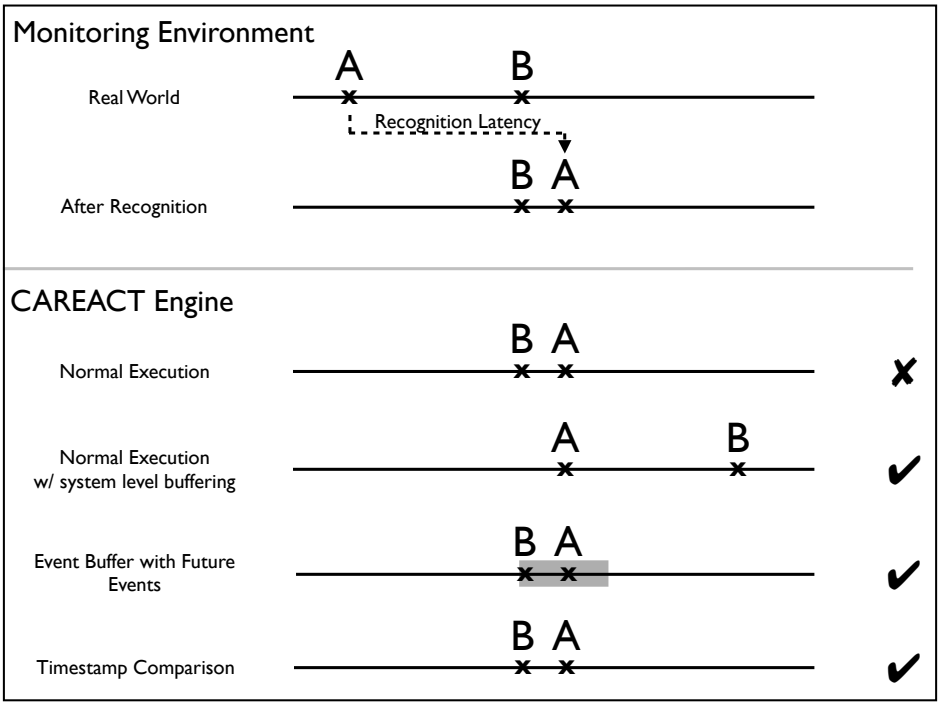


Figure 4.9. Methods of handling latency of different subsystems.

the pattern detection accuracy. One way to deal with this type of problem is buffering events up until a threshold, shown as system level buffering in Figure 4.9, that is equal to the *maximum_recognition_latency*. This way, events that are out of sequence are sorted before feeding them into the CAREACT engine and the real world ordering is restored. The first problem with this approach is that it requires the *maximum_recognition_latency* to be known. The second problem is that this approach adds a fixed latency to every scenario in the CAREACT engine. Another approach that employs the future events is shown as Event Buffer with Future Events in Figure 4.9. In this approach if the second event in a sequence is matched a future event is set. If the first event arrives within this time, the sequence is matched successfully. In other words, this approach allows out of order events to be processed in the CAREACT engine with the correction support in certain scenarios that employ events with latency. This approach does not inject any additional latency. The third approach employs a semantic information store which also allows out of order events to be processed in the CAREACT engine. In this approach, whenever the second event is matched in a pattern, its timestamp is stored in the semantic information store. If the first event is matched afterwards, its timestamp is compared against the second event that is already matched. The

pattern is matched if first event's timestamp is earlier than the second one. The challenge of this method is that it requires every subsystem to provide the real world timestamp of the events they produce.

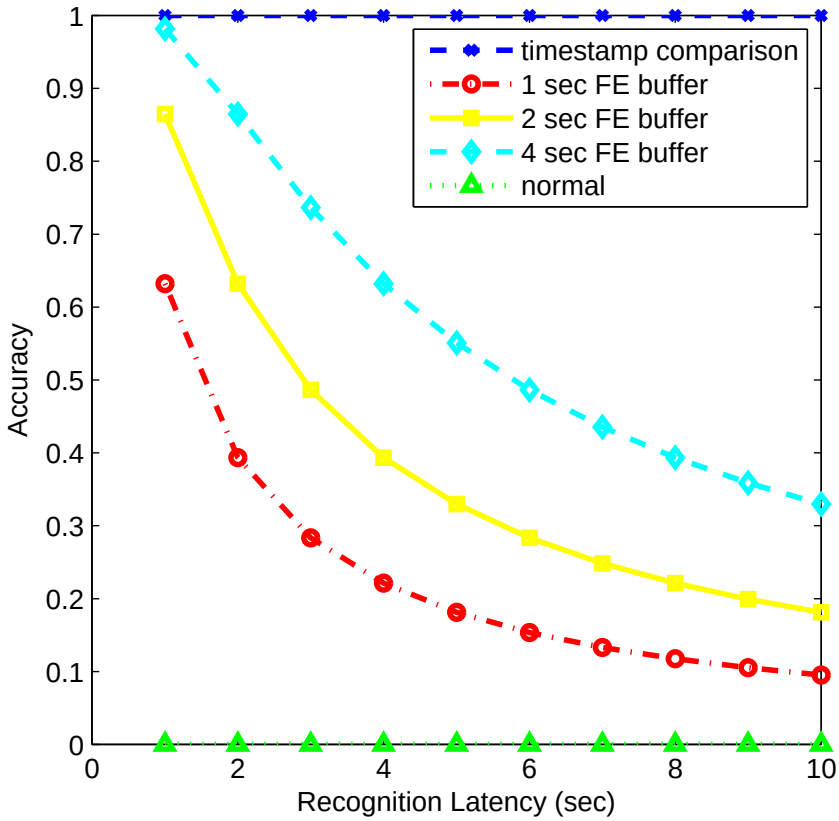


Figure 4.10. Accuracy of different methods on dealing with the recognition latency.

To evaluate the performance of each methodology, we setup the following experiment. For each inhabitant, we produce event pairs of A and B with very close timestamps of B following A and send these events in the reverse order and after waiting for an exponentially distributed random amount of time. The mean of exponential distribution function is named as the *recognition latency*. If the scenario is able to detect the pattern of A followed by B, we count it as a hit, otherwise it is counted as a miss. We calculated three metrics, accuracy, reporting latency and throughput in terms of the number of inhabitants monitored where we defined the accuracy as the ratio of hit count to the overall count.

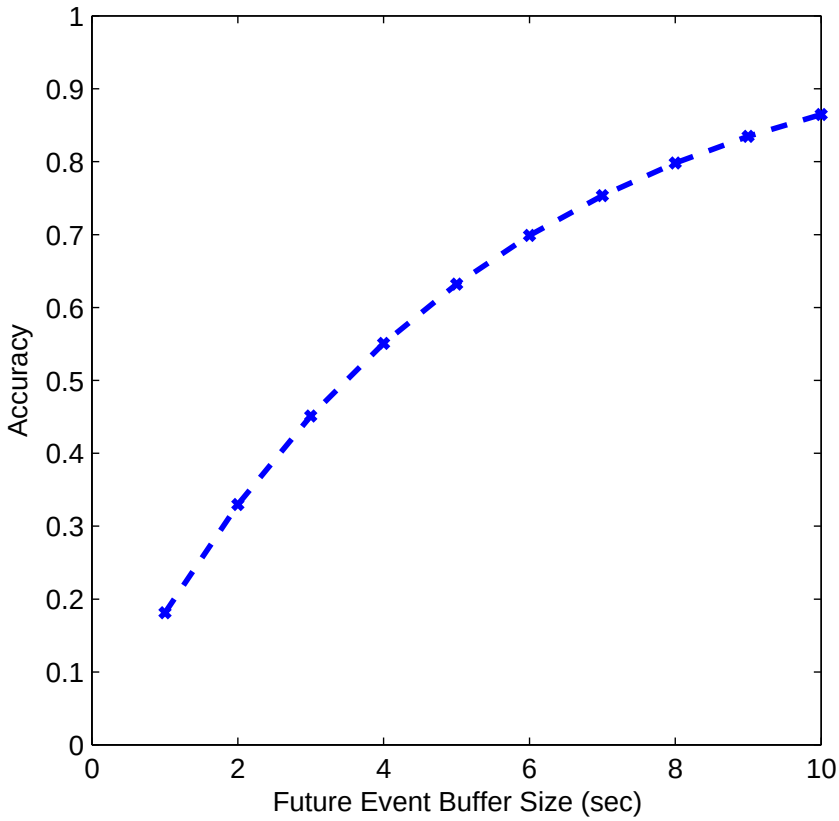


Figure 4.11. Impact of the *future event time* on the accuracy in an environment with 5 sec *Recognition Latency*.

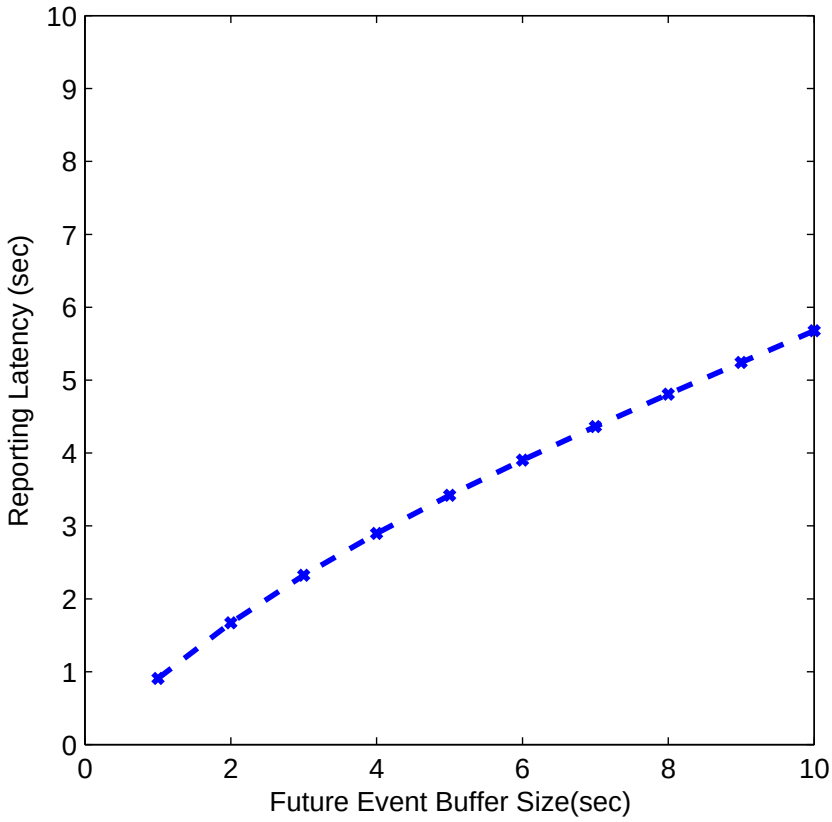


Figure 4.12. Reporting delay of different methods on dealing with the recognition latency.

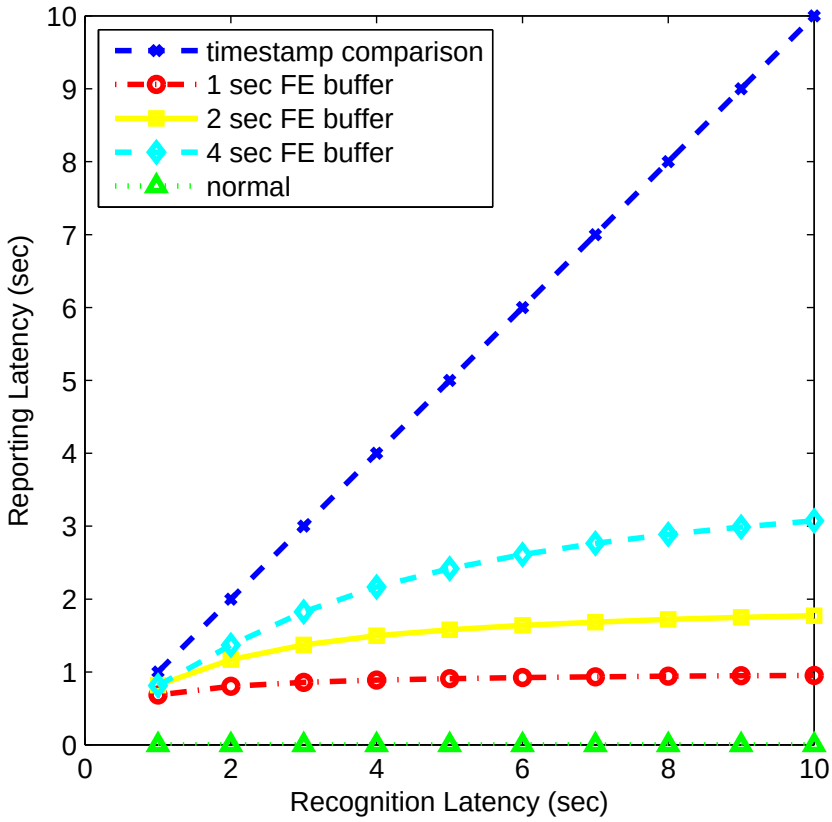


Figure 4.13. Impact of the *future event time* on the *Reporting Latency* in an environment with 5 sec *Recognition Latency*.

The experiments (See Figure 4.10) show that *no buffering*, which is normal execution for the CAREACT engine is not able to detect any pattern whereas *Timestamp based comparison* is able to detect every sequence. For *Future Event Buffer* methodology, we observe that increasing the *future event time* increases the accuracy of pattern detection (See Figure 4.11).

From the reporting latency perspective, *Timestamp based comparison* method always wait for the trailing B to arrive and hence its reporting latency increases linearly as the recognition latency increases. *Future Event Buffer* method waits the full *future event time* before failing to detect a pattern and detect the patterns with buffer period as they arrive. As the recognition latency increases, the *Future Event Buffer* method misses more patterns and the value of the reporting latency gets closer to *future event time*. As the *future event time* increases the *Reporting Latency* increases (See Figure 4.13).

From the maximum throughput perspective the worst performing method is the *Timestamp based comparison* method since in this method the timestamps are saved in every event of type A and read in every event of type B. (See). The maximum throughput of the *Future Event Buffer* method depends on the *future event time* and is presented in Table 4.6.

Table 4.6. Performance evaluation of different methods for handling the recognition latency.

Method	**Througput**
No Buffering	~4,800 events/sec
Timestamp Comparison	~1,160 events/sec
1 sec FE buffer	~2,410 events/sec
2 sec FE buffer	~3,230 events/sec
4 sec FE buffer	~3,890 events/sec

5. CONCLUSION

In this thesis, we analyzed the current literature on both telemonitoring and ambient assisted living from the perspective of detecting anomalies in activities of daily living. The studies on the former focuses on the anomaly detection and remote monitoring whereas the latter focuses on detecting activities from the raw sensor data. Both of the branches lack a holistic solution that effectively detects anomalies in the context of AAL by taking both the raw sensor data and the activities of daily living with the ability to automatically or semi autmatically act on found anomalies.

Consequently, we proposed a novel ambient assisted monitoring framework, the *CARE-ACT*, that has the capability to process both the sensor data and the activity data with the support for four core features that increase performance of overall anomaly detection process in a smart home environment: interoperability, rule support, adaptivity and analyzability. While all these features exist in different studies, to best of our knowledge they have not been combined in a single framework. Furthermore, the proposed architecture runs in real time and is able to act on detected anomalies both automatically and in real time. Our proposed architecture is built on CEP technology because of its high performance, inherent interoperability and rule support. We also addressed a challenge that arose while adapting CEP technology for detecting activities of daily living which includes temporal relationships in the order of minutes, hours or higher that is orders of magnitudes higher than the current CEP literature is able to deal with. Rest assured, the current state of the art CEP engines has more than linear computation and memory complexity dealing with the higher order temporal relationships whereas our proposed mechanism, *Future Events*, has O(1) computation and memory complexity.

We have implemented the proposed architecture as a framework, and realized sample scenarios on top this framework to validate its adaptivity and reporting capabilities. Conducted experiments with sample scenarios on generated data showed that our proposed framework is able to achieve adaptation and present easy to analyze reports. While the adaptivity and analyzability of the results highly suspect to the design of the anomaly detection

scenarios we were able to achieve a decrease in false alert rate from 48% to 5% with adaption on simple scenarios. These figures clearly show that the system gives adaptation tool at the anomaly designers disposal for detecting anomalies. These figures were extracted directly from the sample analysis reports of the scenarios where the desginer could clearly analyze the affect of changing a design element or a parameter within anomaly detection scenario which is a showcase of analyzability aspect of the CAREACT framework.

We devised experiments to measure the raw performance of the CAREACT engine to identify whether or not the engine itself create a bottleneck within the system. We deduced that latency in the order of milliseconds would not be the dominant latency of this end to end system where network communication and execution of modelling algorithms typically take orders of magnitude longer to finish compared to cpu bound execution of the CAREACT engine. We also modelled the system using analytical tools and compared it to experimental results which confirmed that there is a reasonable model for approxiamating the behaviour of the CAREACT engine. We have also integrated it to a working fall detection mobile application to validate its real life applicability. After the integration we were able to observe to entire CAREACT cycle with mobile device as both the sensor, activity recognition system, event sender and alert receiver. Thus while it seemed like the entire framework was too complicated and had too many components, the integration showed that one mobile client and one server was all that is necessary to create an automated and context aware anomaly detection system.

Our future work will focus on ease of scenario development by supplying reusable building blocks and tools to create them. Ultimately, we aim to create a platform as a service where various rule based pattern detection requirements, including but not limited to anomaly detection, are handled by a centralized system that is built on top of the CAREACT architecture.

REFERENCES

1. Alemdar, H. and C. Ersoy, "Wireless Sensor Networks for Healthcare: A Survey", *Computer Networks*, Vol. 54, No. 15, pp. 2688–2710, 2010.

2. Chan, M., D. Estève, C. Escriba and E. Campo, "A Review of Amart Homes - Present State and Future Challenges", *Computer Methods and Programs in Biomedicine*, Vol. 91, No. 1, pp. 55–81, 2008.

3. Zhao, Z. and L. Cui, "EasiMed: A Remote Healthcare Solution", *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, pp. 2145–2148, IEEE, 2006.

4. Liu, C., J. Wen, Q. Yu, B. Yang and W. Wang, "HealthKiosk: A family-based Connected Healthcare System for Long-term Monitoring", *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pp. 241–246, IEEE, 2011.

5. Zhang, P. and M. Chen, "A Remote Health Care System Based on Wireless Sensor Networks", *Computer Supported Cooperative Work in Design, 2008. CSCWD 2008. 12th International Conference on*, pp. 1102–1106, IEEE, 2008.

6. Zhang, Z., U. Kapoor, M. Narayanan, N. Lovell and S. Redmond, "Design of an Unobtrusive Wireless Sensor Network for Nighttime Falls Detection", *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE*, pp. 5275–5278, IEEE, 2011.

7. Corchado, J., J. Bajo, D. Tapia and A. Abraham, "Using Heterogeneous Wireless Sensor Networks in a Telemonitoring System for Healthcare", *Information Technology in Biomedicine, IEEE Transactions on*, Vol. 14, No. 2, pp. 234–240, 2010.

8. van Kasteren, T. L. M., G. Englebienne and B. J. A. Kröse, "Activity Recognition Using Semi-Markov Models on Real World Smart Home Datasets", *J. Ambient Intell. Smart*

Environ., Vol. 2, No. 3, pp. 311–325, Aug. 2010.

9. Chen, L., C. Nugent and H. Wang, "A Knowledge-driven Approach to Activity Recognition in Smart Homes", *Knowledge and Data Engineering, IEEE Transactions on*, , No. 99, pp. 1–1, 2011.

10. Naranjo, D., L. Roa, J. Reina-Tosina and M. Estudillo-Valderrama, "Personalization and Adaptation to the Medium and Context in a Fall Detection System", *Information Technology in Biomedicine, IEEE Transactions on*, , No. 99, pp. 1–1, 2012.

11. Maglogiannis, I. and C. Doukas, "Emergency Fall Incidents Detection in Assisted Living Environments Utilizing Motion, Sound and Visual Perceptual Components", *Information Technology in Biomedicine, IEEE Transactions on*, , No. 99, pp. 1–1, 2011.

12. Coffi, J., C. Marsala and N. Museux, "Adaptive Complex Event Processing for Harmful Situation Detection", *Evolving Systems*, pp. 1–11.

13. Storf, H., T. Kleinberger, M. Becker, M. Schmitt, F. Bomarius and S. Prueckner, "An Event-driven Approach to Activity Recognition in Ambient Assisted Living", *Ambient Intelligence*, pp. 123–132, 2009.

14. Mouttham, A., L. Peyton, B. Eze and A. Saddik, "Event-driven Data Integration for Personal Health Monitoring", *Journal of Emerging Technologies in Web Intelligence*, Vol. 1, No. 2, pp. 110–118, 2009.

15. Lubrin, E., E. Lawrence and K. Navarro, "Wireless Remote Healthcare Monitoring with Motes", *Mobile Business, 2005. ICMB 2005. International Conference on*, pp. 235–241, IEEE, 2005.

16. Fülöp, L. J., G. Tóth, R. Rácz, J. Pánczél, T. Gergely, A. Beszédes and L. Farkas, "Survey on Complex Event Processing and Predictive Analytics", *Nokia Siemens Networks*, 2010.

17. Wu, E., Y. Diao and S. Rizvi, "High-performance Complex Event Processing over Streams", *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 407–418, ACM, 2006.

18. Mei, Y. and S. Madden, "Zstream: a Cost-based Query Processor for Adaptively Detecting Composite Events", *Proceedings of the 35th SIGMOD international conference on Management of data*, pp. 193–206, ACM, 2009.

19. Liu, M., E. Rundensteiner, D. Dougherty, C. Gupta, S. Wang, I. Ari and A. Mehta, "High-performance Nested CEP Query Processing over Event Streams", *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pp. 123–134, IEEE, 2011.

20. Adaikkalavan, R. and S. Chakravarthy, "SnoopIB: Interval-based Event Specification and Detection for Active Databases", *Data & Knowledge Engineering*, Vol. 59, No. 1, pp. 139–165, 2006.

21. Demers, A., J. Gehrke, B. Panda, M. Riedewald, V. Sharma, W. White *et al.*, "Cayuga: A General Purpose Event Monitoring System", *Proc. CIDR*, pp. 412–422, 2007.

22. Alemdar, H., Y. Kara, M. Ozen, G. Yavuz, O. Incel, L. Akarun and C. Ersoy, "A Robust Multimodal Fall Detection Method for Ambient Assisted Living Applications", *Signal Processing and Communications Applications Conference (SIU), 2010 IEEE 18th*, pp. 204–207, IEEE, 2010.

23. Alemdar, H., G. Yavuz, M. Özen, Y. Kara, Ö. Incel, L. Akarun and C. Ersoy, "Multi-modal Fall Detection Within the WeCare Framework", *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pp. 436–437, ACM, 2010.

24. Jiang, S., Y. Cao, S. Iyengar, P. Kuryloski, R. Jafari, Y. Xue, R. Bajcsy and S. Wicker, "CareNet: an Integrated Wireless Sensor Networking Environment for Remote Healthcare", *Proceedings of the ICST 3rd international conference on Body area networks*, p. 9, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunica-

tions Engineering), 2008.

25. Lu, H. and J. Chen, "Design of Middleware for Tele-homecare Systems", *Wireless Communications and Mobile Computing*, Vol. 9, No. 12, pp. 1553–1564, 2009.

26. Huo, H., Y. Xu, H. Yan, S. Mubeen and H. Zhang, "An Elderly Health Care System Using Wireless Sensor Networks at Home", *Sensor Technologies and Applications, 2009. SENSORCOMM'09. Third International Conference on*, pp. 158–163, IEEE, 2009.

27. Ungureanu, G., I. Gussi, W. Wolf, D. Taralunga, S. Pasca and R. Strungaru, "Prenatal Telemedicine–Advances in Fetal Monitoring", .

28. Catarinucci, L., R. Colella, A. Esposito, L. Tarricone and M. Zappatore, "RFID Sensor-Tags Feeding a Context-Aware Rule-Based Healthcare Monitoring System", *Journal of Medical Systems*, pp. 1–15, 2011.

29. Szűcs, V., Á. Végh, M. Kasza and V. Bilicki, "Evaluating Data Modeling Aspects for Home Telemonitoring", *PATTERNS 2011, The Third International Conferences on Pervasive Patterns and Applications*, pp. 78–83, 2011.

30. Ngai, E., J. Poon, F. Suk and C. Ng, "Design of an RFID-based Healthcare Management System Using an Information System Design Theory", *Information Systems Frontiers*, Vol. 11, No. 4, pp. 405–417, 2009.

31. Toplan, E. and C. Ersoy, "RFID Based Indoor Location Determination for Elderly Tracking", *Signal Processing and Communications Applications Conference (SIU), 2012 20th*, pp. 1–4, IEEE, 2012.

32. Quintas, J., K. Khoshhal, H. Aliakbarpour, M. Hofmann and J. Dias, "Using Concurrent Hidden Markov Models to Analyse Human Behaviours in a Smart Home Environment", *International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2011), University of Delft, Netherlands (citation for C11)*, 2011.

33. Rashidi, P., D. Cook, L. Holder and M. Schmitter-Edgecombe, "Discovering Activities to Recognize and Track in a Smart Environment", *Knowledge and Data Engineering, IEEE Transactions on*, , No. 99, pp. 1–1, 2011.

34. Hinze, A., K. Sachs and A. Buchmann, "Event-based Applications and Enabling Technologies", *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, p. 1, ACM, 2009.

35. Yao, W., C. Chu and Z. Li, "Leveraging Complex Event Processing for Smart Hospitals Using RFID", *Journal of Network and Computer Applications*, Vol. 34, No. 3, pp. 799–810, 2011.

36. Wang, D., E. Rundensteiner, R. Ellison and H. Wang, "Active Complex Event Processing infrastructure: Monitoring and reacting to event streams", *Data Engineering Workshops (ICDEW), 2011 IEEE 27th International Conference on*, pp. 249 –254, april 2011.

37. White, W., M. Riedewald, J. Gehrke and A. Demers, "What is Next in Event Processing?", *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 263–272, ACM, 2007.

38. Cohen, N. and K. Kalleberg, "EventScript: an Event-processing Language Based on Regular Expressions with Actions", *ACM SIGPLAN Notices*, Vol. 43, pp. 111–120, ACM, 2008.

39. Welbourne, E., N. Khoussainova, J. Letchner, Y. Li, M. Balazinska, G. Borriello and D. Suciu, "Cascadia: a System for Specifying, Detecting, and Managing RFID Events", *Proceeding of the 6th international conference on Mobile systems, applications, and services*, pp. 281–294, ACM, 2008.

40. Nie, Y., Z. Li and Q. Chen, "Complex Event Processing over Unreliable RFID Data Streams", *Web Technologies and Applications*, pp. 278–289, 2011.

41. Agrawal, J., Y. Diao, D. Gyllstrom and N. Immerman, "Efficient Pattern Matching Over

Event Streams", *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 147–160, ACM, 2008.

42. Liu, M., M. Ray, E. Rundensteiner, D. Dougherty, C. Gupta, S. Wang, I. Ari and A. Mehta, "Processing Nested Complex Sequence Pattern Queries over Event Streams", *Proceedings of the Seventh International Workshop on Data Management for Sensor Networks*, pp. 14–19, ACM, 2010.

43. Liu, M., E. Rundensteiner, D. Dougherty, C. Gupta, S. Wang, I. Ari and A. Mehta, "Neel: The Nested Complex Event Language for Real-time Event Analytics", *Enabling Real-Time Business Intelligence*, pp. 116–132, 2011.

44. Schmidt, K., D. Anicic and R. Stühmer, "Event-driven Reactivity: A Survey and Requirements Analysis", *3rd International Workshop on Semantic Business Process Management*, pp. 72–86, 2008.

45. Wang, D., E. Rundensteiner and R. Ellison III, "Active Complex Event Processing over Event Streams", *Proceedings of the VLDB Endowment*, Vol. 4, No. 10, pp. 634–645, 2011.

46. Ko, J., C. Lu, M. Srivastava, J. Stankovic, A. Terzis and M. Welsh, "Wireless Sensor Networks for Healthcare", *Proceedings of the IEEE*, Vol. 98, No. 11, pp. 1947–1960, 2010.

47. Yavuz, G., M. Kocak, G. Ergun, H. Alemdar, H. Yalcin, O. D. Incel and C. Ersoy, "A Smartphone Based Fall Detector with Online Location Support", *International Workshop on Sensing for App Phones; Zurich, Switzerland*, pp. 31–35, 2010.