Context-awareness in Task Automation Services by Distributed Event Processing

Miguel Coronado , Ralf Bruns , Jürgen Dunkel , and Sebastian Stipković

Abstract. Everybody has to coordinate several tasks everyday, usually in a manual manner. Recently, the concept of Task Automation Services has been introduced to automate and personalize the task coordination problem. Several user centered platforms and applications have arisen in the last years, that let their users configure their very own automations based on third party services. In this paper, we propose a new system architecture for Task Automation Services in a heterogeneous mobile, smart devices, and cloud services environment. Our architecture is based on the novel idea to employ distributed Complex Event Processing to implement innovative mixed execution profiles. The major advantage of the approach is its ability to incorporate context-awareness and real-time coordination in Task Automation Services.

Keywords: Distributed Task Automation Services, Complex Event Processing, Personalized Services, Context-Awareness, Mobile Services

1 Introduction

Nowadays, most users of smartphones, smart devices, social platforms, and cloud services use these emerging technologies to coordinate their private and business tasks (and, of course, for other purposes). However, the numerous coordination tasks are still performed *manually* to provide different technical platforms and human participants with new information or to trigger appropriate actions. To overcome with this cumbersome and time-consuming procedure, Task Automation Services (TAS) platforms have been introduced recently.

Task Automation Services allow the users to automate their tasks by defining simple rules instead of performing manually all the required steps of a task. If these automation rules are matched by events that are emitted by smartphones or by services (such as Twitter or Dropbox), they trigger a desired reaction. For instance, some users may want to "post in Twitter their Facebook status as

soon as they publish it". Others may also need to "update their Twitter profile picture any time they change their Facebook's".

Currently, several TAS platforms are available to provide this type of functionalities. We can distinguish two types of TAS's depending on the platform they are running on:

- *Web-based TAS's* such as Ifttt[3], Zapier[4] and Elastic.io[5] are deployed as cloud services. They collect personal events by accessing appropriate web services on behalf of the user and provide a simple rule editor.
- *Smartphone-based TAS's* such as AutomateIt[6] and Tasker[7] run on a smartdevice and have access not only to data via web services, but also to the local resources of the device, e.g. the embedded smartphone sensors.

Task Automation Services rules may be executed according to different *execution profiles* that define where rule execution takes place. According to the above mentioned TAS types, we may distinguish:

1. A *web-driven execution profile* centralizes the rule execution on a server, allowing lightweight clients at the cost of requiring Internet connection. Typically, clients setup and manage the rules by a web page. Alternatively, smartphone apps could provide the same functionality. Web-driven execution profiles may have to cope with a huge amount of incoming events; they may have access to a large set of channels and may coordinate events from different users.
2. A *device-driven execution profile* executes all rules on the device itself, allowing offline rule execution (when only local resources are involved). Usually, when we talk about device-driven TAS, we refer to smartphone apps, although the definition is not restricted to smartphone devices. Rules in a device-driven execution profile can exploit the device-specific data, e.g. provided by the smartphone sensors. Therefore, some rules could derive the users' local context or current situation.
3. A *mixed execution profile* benefits from the advantages of both previous profiles. It distributes the execution of automation rules between clients (smart devices) and servers. However, mixed execution profiles require a distributed and, therefore, more complex system architecture and more complex rules.

Note that current TAS systems are still rather restricted: they allow only the definition of very simple rules. Furthermore, they cannot combine web-driven and device-driven execution profiles, i.e. mixed execution profiles are yet not available.

In the following, we will present an innovative architecture for Distributed Task Automation Services supporting mixed execution profiles. Our approach

[3] http://ifttt.com
[4] http://zapier.com
[5] http://www.elastic.io/
[6] http://automateitapp.com/
[7] http://tasker.dinglisch.net

is based on the employment of Complex Event Processing (CEP). CEP is a novel software technology for processing continuous streams of data in near real-time [9]. The basic concept of CEP is in-memory pattern matching, which means to identify in data streams those patterns of data that represent a meaning-full situation in the application domain.

In our approach, we use CEP to build a Distributed TAS system that is capable of coordinating peoples' tasks in real-time. The approach provides the following features:

- *Context-awareness:* The current activities, contexts and situations of the participating users can be concluded by correlating sensor data of their smartphones (e.g. accelerometer, GPS) and further domain-specific context information. The corresponding rules are realized in a device-driven execution profile.
- *Coordination:* Appropriate TAS rules coordinate various participants by taking into account their current context and situation information. They are realized according to a web-driven execution profile, which is implemented on a central server.

The paper is structured as follows. In the following section, we present a TAS coordination scenario that motivates our approach, and which is used to explain our approach in the subsequent sections. Then in Section 3, we present the basic concepts of Complex Event Processing. In Section 4, we describe our general architecture of a Distributed TAS system. In the subsequent sections, we evaluate our approach and present some implementation issues. The related work is discussed in Section 7. Finally, we summarize the most significant features of our approach and give a brief outlook on future lines of research.

2 TAS Coordination Scenario

Task Automation Service's (TAS's) are highly flexible platforms that users can use to orchestrate task automation addressing many different situations. The scenario we describe in this section presents a complex use case, where various smart devices determine the current situations of their owners, which are then broadcasted to a central Task Automation Service that performs appropriate coordination tasks. Note that each smart device is capable to orchestrate simple automations on their own, but that a centralized TAS platform is used for coordination purposes.

Consider the following use case: Patricia and Thomas live together. They share the housework, which also includes outside tasks such as shopping, sharing the car, or picking up their children from the kindergarten. Since they work in different parts of the city, they cannot devise a fixed schedule beforehand. In the past, it required a high coordination effort for them to organize these things manually by phone calls or text messages. Sometimes it happened that they didn't notify each other, causing that both of them went shopping at the same

time (buying the needed groceries twice) or forgetting to tell that they have already picked up the kids.

Because they both use TAS for their personal automations, they decided to share several rules that help them in coordinating these tasks. They set up rules to automatically inform each other, when they are in a certain situation or doing a certain activity. Using the GPS sensors of Patricia and Thomas' smartphones, the TAS can deduce the concrete situation, in which the two of them are, causing an appropriate action. In the usual TAS terminology, one rule could be read as "When I am at the supermarket, then text my mate that I'm shopping". Then, if Thomas goes to the supermarket after work, Patricia will know he is doing the shopping, so she does not need to go there.

The task "picking up the children" requires that one of them is at the kindergarten shortly before the children are dismissed. Therefore, it requires the TAS to coordinate ahead, taking journey times from their current position to the kindergarten into account. The task could be expressed as "Everyday, either Patricia or Thomas must be at the kindergarten at 17 o'clock. Usual rules for coordinating this task could be "When I'm at home and my mate is still at work, remind me I should pick up the children" and "If my mate was at the kindergarten, inform me that I don't have to pick up the children".

In particular, automatic task coordination avoids manually triggered notifications, which are error-prone and awkward. Furthermore, corresponding messages can take the current situation of the recipient into account, i.e. they are only delivered, when the recipient is in a ready-to-receive mood.

3 Complex Event Processing

Complex Event Processing is an innovative software technology for processing continuous streams of events in near real-time [9], [10]. Everything that happens inside or outside of a system is considered as an *event*. CEP analyses streams of incoming events to detect the presence of *event patterns*.

An event pattern is a particular sequence of events with a special meaning for the application domain. A *pattern match* signifies a meaningful situation or state of the environment and causes either the generation of a new *complex event* or triggers a domain-specific action. Complex events correlate between simple events and provide the real power of CEP.

Event stream processing systems manage the most recent set of events in-memory and employ sliding windows and temporal operators to specify temporal relations between the events in the stream. The core concept of CEP is a declarative *event processing language* (EPL) to express *event processing rules*. An event processing rule contains two parts: a *condition* part describing the requirements for firing the rule and an *action* part that is performed if the condition matches. The *condition* is defined by an event pattern using several operators and further constraints [3].

In the following, we use a simplified pseudo language for expressing event processing rules, which is easier to understand than an EPL of a productive CEP system. This pseudo language supports the following operators:

Operators
`AND, OR` Boolean operator for events or constraints.
`NOT` Negation of a constraint.
`->` Sequence of events.
`Timer` *Timer(time)* defines a time to wait.
Timer.at(daytime) is a specific (optionally periodic) point of time.
`.within` defines a time window in which the event has to occur.

An *event processing engine* analyses the stream of incoming events and executes the matching rules. Event processing rules transform low level simple events into more complex events in order to gain insight into the current state of the environment.

Luckham introduced the concept of *event processing agents* (EPA) [10]. An EPA is an individual CEP component with its own rule engine and rule base. Several EPAs can be connected to an *event processing network* (EPN) that constitutes a software architecture for event processing. Event processing agents communicate with each other by exchanging events.

4 Architecture

In this section, we present an architecture for Distributed TAS supporting mixed execution profiles. In particular, our architecture exploits the sensor data of the smart devices for achieving situation awareness.

4.1 Architecture overview

An overview of the overall system architecture is given in Fig. 1. The distributed architecture shows the different TAS rule engines according to the mixed execution profile definition.We can distinguish the following components:

Smart devices: The system consists of numerous smart devices, which have the following responsibilities in the system. First, they collect all events emitted by its sensors and other local resources (the so-called content providers). The streams of events are processed on each smart device by its own CEP rule engine, which contains appropriate rules for providing semantic inference. In particular, the CEP rules filter, process and enhance the observed data events to produce richer situation events that reflect the current users' context. All the situation events are sent to the server to allow cross-user coordination. Furthermore, smart devices can perform conventional task automation rules. These rules can react on responses of the coordination server, or they are either server-independent and can be processed locally.

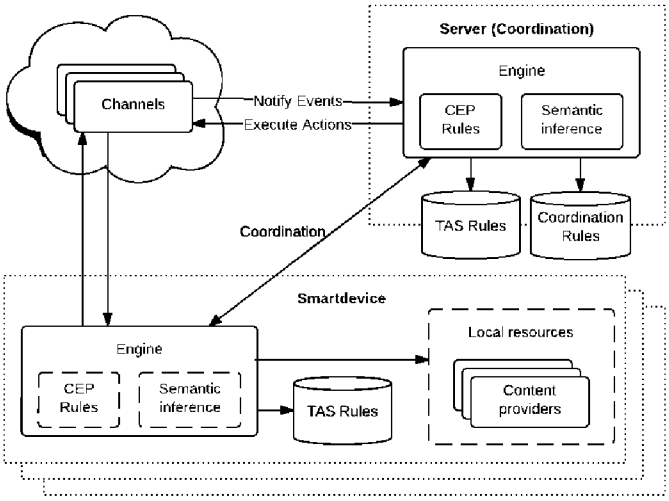

Fig. 1. Architecture overview of a distributed TAS system with coordination.

Coordination Server: The central Coordination Server is deployed to the cloud and responsible for coordinating the smart devices and their users. For this purpose, it also has its own CEP engine with appropriate coordination rules that manage the smart devices taking the users' current context into account. Furthermore, the server provides support for conventional task automation rules, that orchestrate automations between web channels.[8]

Channels: Additionally, both the smart devices as well as the coordination server have access to web services (the so-called *channels*), using specific connectors.[9] These web services can provide the system with more necessary context information, e.g. weather or traffic data.

In our architecture we distinguish CEP rules from Task Automation rules: Because CEP rules allow temporal reasoning, all rules that involve movement, or GPS positioning will be defined as CEP rules. Note that CEP rules provide some form of semantic inference: they shift simple events (e.g. sensor events) to complex events (e.g. situation events) that assign the occurred events a new non-obvious and semantically richer meaning.

[8] This is the case for rules like "Whenever I receive an email with attachment save that attachment on my Dropbox", those are out of the scope of our scenario, but they are still supported by our system.

[9] In most cases, they are implemented by API connectors (because most third party web service developers offer it); however, webhooks or pub-sub are even more convenient approaches to work with events on the cloud.

4.2 CEP for TAS

In this section, we will explain in some more detail, how Complex Event Processing (CEP) in our TAS architecture works (see Fig. 1). In our approach, CEP is based on a multi-staged Event Processing Network (EPN) in order to logically structure and modularize the event processing rules.

To make the explanation of our approach concrete, we will use our application scenario presented in section 2. The following Fig. 2 shows a set-up with two different smartphones[10] and the central Coordination Server. The Event Processing Network contains various Event Processing Agents (EPAs) that are distributed on the different devices.

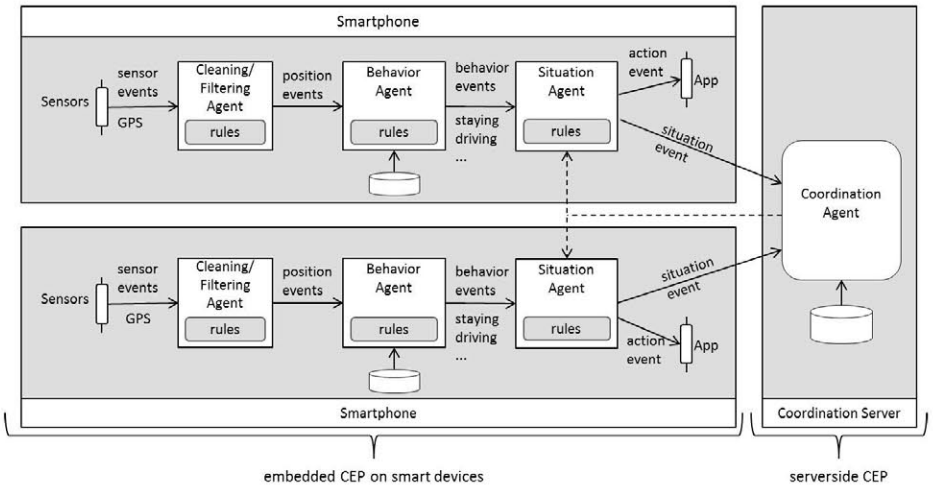

Fig. 2. Event Processing Network for distributed TAS

The EPN defines the archetypal processing stages and the related EPAs, which are common to most TAS systems. However, the particular event processing rules must be adapted to a specific application scenario. In the following we describe the responsibilities of each EPA.

Cleaning/Filtering Agent

The Cleaning/Filtering Agent is deployed to the smartphones and collects all sensor events, such as the *GPS events*. Sensor data is often inconsistent or has redundant information, because sensors are noisy and have a fixed sampling rate. Therefore, in a first step, all technical sensor events have to be pre-processed to overcome inconsistencies or to filter out irrelevant events.

[10] Other types of smart devices like smart home devices are possible, which would have their own domain-specific EPN.

For instance, GPS sensor data is generated with a fixed sampling rate. Thus, many subsequent *GPS events* are logically identical. But the TAS system is only interested in situation changes, and not in the repetition of events carrying similar measured values. Therefore, the Filtering Agent filters out those *GPS events* that are related to the same geographical position. The following event processing rule has the task to find out, if the phone has been moved to a new position.

```
rule: "new phone/user position"
CONDITION   GPS-Event AS g1 ->
            GPS-Event AS g2
            AND (Geo.isDifferent(g1,g2))
ACTION      new PositionEvent( g2.x, g2.y)
```

The rule "new phone/user position" expresses a temporal sequence of *GPS events* (by the following operator "->") and assigns the alias names `g1` and `g2` to them. The newer event `g2` represent the current position of the user and is only relevant if the GPS position has significantly changed, which is checked by the service `Geo.isDifferent(...)`. In the action part of the rule, a new *Position event* with the information of the current position is triggered.

Behavior Agent

The incoming *Position events* are correlated with further sensor events (e.g., *Acceleration sensor events*) to determine the particular behavior of the user. New (more complex) *Motion events* are created that characterize the current behavior of the smartphone user. Here we consider the different types of motion such as "walking", "driving", "staying". The following rule derives that the user is staying for a longer time a certain position.

```
rule: "staying"
CONDITION PositionEvent AS pos ->
          NOT (PositionEvent).within(5 min)
ACTION    new StayingEvent(pos)
```

The above rule assumes that the user is staying at a certain position, if a *Position event* is not followed by a new *Position event* within a time interval of 5 minutes. The operator `.within` defines a time window, in which a certain event has to occur.

The next rule derives a corresponding *Driving event*. The average velocity of a moving user can be calculated by aggregating all *Position events* within the last five minutes and determining the average of the measured speed values. The speed is determined by a method `getSpeed(..)` that is provided by the GPS sensors. If the speed is faster than 15 km/h, it is concluded that the user is driving.

```
rule: "driving"
CONDITION PositionEvent.avg(getSpeed())
                       .within:batch(5 min)
          AS avgVelocity
          AND avgVelocity > 15 km/h
ACTION    new DrivingEvent(avgVelocity)
```

In summary, the Behavior Agent processes a correlation step to synthesize *Motion events*. All *Motion events* are subsequently propagated to the Situation Agent.

Situation Agent

In the next processing stage, the Situation Agent is determining the current situation of the smartphone user. The situations of interest depend on the concrete use case scenario. For instance, in our example scenario 'picking up the children from kindergarten', we want to know, where each family member is and if the children have already been picked up.

The incoming *Location* and *Motion events* are carrying only GPS coordinates that have no specific meaning in the TAS domain, and are not sufficient for further processing. Therefore, the GPS data should be transformed to domain locations. A first enrichment step relates GPS coordinates to a real address, which can be done by a reverse geocoding API, e.g. provided by GoogleMaps. Then the address can be mapped to a relevant location of the user, such as "kindergarten", "home" or "work". An example gives the following simple rule that derives a "working" situation:

```
rule: "In Working situation"
CONDITION ( StayingEvent AS stay
                    -> NOT PositionEvent )
           AND LocationFinder.getLocation(stay.position) == "work"
ACTION    new WorkingEvent(user)
```

If the system has created a *Staying event*, which is not followed by a new *Position event* (i.e. no significant movement has occurred afterwards), then the GPS position is checked in a utility class `LocationFinder.getLocation(..)`. If the positions corresponds to the users' workplace, a new *Working event* will be created.

All *Situation events* are sent to the TAS server in order to allow task coordination based on the current situations of the users.[11] Therefore, the *Working event* will carry information for identifying the smartphone user.

[11] Detected situations can also generate *Action events* which are sent to an app on the smartphone in order to trigger an appropriate app action.

Coordination Agent

The Coordination Agent is deployed to a central cloud server and responsible for coordination tasks. All smart devices send their *Situation events* to the Coordination Agent that coordinates common tasks and conflicts centrally. In the kindergarten example, the following simplified rule could determine that the person, which is not working, has to pick up the children.

```
rule: "picking up children"
CONDITION
          ( WorkingEvent(u1) -> NOT SituationEvent(u1) ) AND
          ( HomeEvent(u2) -> NOT SituationEvent(u2) )
          -> Timer.at(17 o'clock)
ACTION new PickUpChildrenEvent(u2)
```

The rule matches, if for user `u1` a *Working event* and for user `u2` a *Home event* has occurred. To make sure that their situations haven't changed, no subsequent *Situation events* may have occurred. Furthermore, the current time must be 17 o'clock. If all this holds, then a *PickUpChildrenEvent* is created for the user `u2`, who is already at home. Additionally, the SituationAgent triggers an *Action event*, which prompts or signals the user `u2` to pick up the children from the kindergarten.

Note that this a simplified example. For a realistic coordination mechanism more sophisticated rules are necessary.

4.3 TAS Event Model

The event model of our TAS application is depicted in Fig. 3 showing the different types of events that are used by the event processing rules presented above. Note that the grey boxes represent the generic event types common to most classes of TAS coordinating systems. The various subtypes are more specific, here to our use case described in section 2.

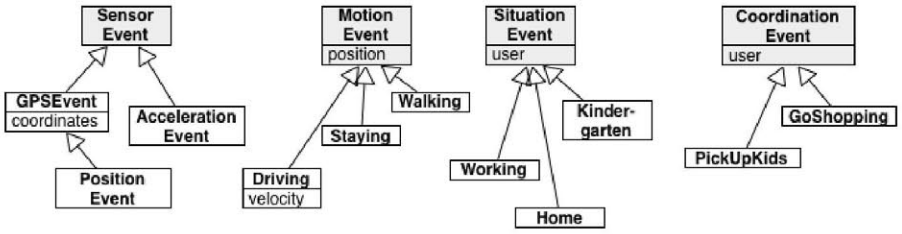

Fig. 3. Event model

The TAS system makes use of the following types of events:

- *Sensor events* are explicit events that are emitted by explicit event sources, here the sensors of the mobile devices. In particular, we can distinguish *GPS events*, *Acceleration events* and *Position events*, which are filtered *GPS events*.
- *Motion events* describe the current motion of a user and are produced by CEP rules that correlate various *Sensor events*. In our example, we distinguish *Driving*, *Walking* and *Staying events*.
- *Situation events* describe the current situation of a user, which is application specific (in our case we consider *Working*, *Home* and *Kindergarten* events).
- *Coordination events* are a result of a coordination rule that correlates various *Situation events* from different users/devices. *Coordination events* are sent back to the related mobile devices. They inform the user about task they are obliged to.

5 Evaluation

The presented architecture distributes TAS coordination on different components: the smartphones provide local situation-awareness for each user. The central web server is aware of the global situation and is responsible for coordinating the tasks of all participating users. Our architecture offers the following advantages:

- *Reduced network traffic*: Sensor data is processed directly on the mobile device and not send to the central server. Because the sensors of potentially many users may produce a high volumes of data, the overall network traffic is reduced significantly.
- *Exploiting local processing power*: Processing data on the smart devices also exploits the processing power of mobile devices, which nowadays is reasonable. The central coordination server doesn't have to track each movement of each device.
- *Privacy:* All participants get only the information that is relevant and necessary for them to know: In our scenario, users are not able to track GPS coordinates of other users, which would violate privacy. They only receive messages about what they are obliged to do. Furthermore, private and sensible user data such as working places, kindergarten or home addresses must not be revealed to a central server.

6 Implementation Issues

The Distributed TAS architecture shown in Fig. 1 has been implemented prototypically in order to prove the feasibility of our approach. As smart devices we used smartphones with the Android operating system. The smartphones are the mobile clients of the central coordination server.

The client application (= app) has been developed with the Android application framework that provides access to the local device resources like hardware

sensors of the device and the data of all installed applications. So far, commercial CEP engines have not been developed for mobile operating systems. However, the popular open source CEP engine Esper[12] (in version 3.2) has recently been ported to the Android platform: the open source CEP engine Asper[13] is based on Esper 4.9.0. Asper provides the most important features of powerful CEP systems for the Android platform, so that we could use it with minor problems as the code base for our Distributed TAS system.

We identity three different types of communication between actors in out architecture. The mobile devices send their events to the HTTP interface of the central coordination server. On the coordination server the incoming events are processed by the open-source CEP engine Esper. The coordination server pushes notifications to smarphone devices (e.i. Android applications) using Google Cloud Messaging for Android (GCM)[14]. GCM enables asynchronous and resource-saving communication from the TAS server to the mobile CEP application. As illustrated in Fig. 1, both the Android application and the central server have access to cloud services by means of so-called channels, which are implemented as web services. By specific connectors, those web services provide further context information like weather and traffic data.

As our implementation responds to a prototype and its objetive it to proof the viability of our architecture and its benefits, we have not considered necessary to include security mechanisns to guarante personal data may not be leaked out from the server. However, it is obvious that the alternative scenario where all GPS information is shared p2p shows more privacy risks. For similar reasons, a rule editor has not been developed. Thus, all user rules are coded according to the pseudo language described in section 3 and stored in-memory.

7 Related Work

The employment of Complex Event Processing for Task Automation Services is a novel field of application, where only very first approaches have been published [5]. In general, related work shows task automation approaches conceived to solved particular problems, that lack of the flexibility and personalization capabilities that characterize TAS's. Automating business rules correlating events coming from different processes is a good showcase with lots of researches behind [6]. Smarthome automations constitute a renewed usecase where smartdevices can coordinate to work in a desired way e.g. for energy saving [7].

On the other hand, commercial TAS like Ifttt or Zapier lack of CEP i.e. they process incoming events as soon as they arrive, so rules are always triggered by a single event. This is not the case of automations on the Internet of Things (IoT) field. Several authors propose systems where built-in rules are triggered by correlated events coming from different sensors [2, 4, 8]. SPITFIRE platform [5] is close to TAS's vision, since it provides a user interface to set up rules (called queries).

[12] `http://esper.codehaus.org/`
[13] `https://github.com/plingpling/asper`
[14] `http://developer.android.com/google/gcm/index.html`

However, they only consider connecting sensors and actuators, not cloud services. They do not address task coordination either. CASAS [13] constitutes a different approach, it uses a Machine Learning algorithm to learn from the resident's daily activities and generate automation polices that mimic these patterns.

In general, CEP engines have been primarily developed for the emerging market of business information systems. The engines are deployed on powerful server systems and process high level events from backend business processes. Commercial vendors of CEP engines have focused on this profitable enterprise market segment [15]. Until a few years ago, mobile operating systems were rather inefficient and the computing resources of mobile devices were very limited. As a consequence, vendors have not been interested to develop a CEP engine for this area of use. Along with the rise of computing power of mobile devices, recently, first proposals demonstrate the applicability of CEP for processing data streams emitted by mobile devices, in particular by the embedded sensors. However, either the mobile devices serve merely as special event sources [1], [12] or the sensor data are only preprocessed on the mobile device in order to achieve context-aware event filtering [11]. The real event processing of mobile data sources is usually still executed on powerful backend servers. The execution of sophisticated event processing rules directly on the mobile device is still a rather new approach [14]. Consequently, mixed execution profiles for distributed event processing have not been proposed so far.

8 Conclusion

Task Automation Services is an emerging area with multiple application domains and challenging technical implications. In this paper, we presented an innovative system architecture for context-aware and personalized TAS. Applying Complex Event Processing and mixed execution profiles are novel concepts for TAS. The proposed TAS architecture possess the following properties.

- *Situation- and Context-Awareness:* The built-in sensors of smartphones or other smart devices provide the TAS system with a continuous stream of context data. Event processing rules are used to aggregate and correlate the sensor data to more abstract and more meaningful situation data.
- *Coordination:* We introduced a TAS cloud server that provides cross-user coordination exploiting the context data of each participant.
- *Real-time Processing:* The real-time capabilities of CEP are exploited on the cloud-based TAS as well as on the device-based TAS.
- *Distributed Processing:* by mixed execution profiles combine the advantages of formerly separated web-driven as well as device-driven execution profiles.

In summary, our approach leads to a new quality of TAS: Distributed Task Automation Services.

In future work, we intend to investigate more complex task coordination scenarios with advanced mixed execution profiles. In particular, the incorporation of diverse smart devices, such as smart home automation devices or smart vehicles, seems to be very promising.

Acknowledgement This work was partly funded by the Spanish Ministry of Economy and Competitiveness through the project Calista (TEC2012-32457).

References

1. Amade, D.: Joining oracle complex event processing and j2me to react to location and positioning events. http://www.oracle.com/technetwork/articles/amadei-cep-090595.html (2010)
2. Arcelus, A., Jones, M.H., Goubran, R., Knoefel, F.: Integration of Smart Home Technologies in a Health Monitoring System for the Elderly. In: 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07). vol. 2, pp. 820–825. IEEE (2007)
3. Bruns, R., Dunkel, J.: Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse. Springer-Verlag (2010)
4. Byun, J., Jeon, B., Noh, J., Kim, Y., Park, S.: An intelligent self-adjusting sensor for smart home services based on ZigBee communications. IEEE Transactions on Consumer Electronics 58(3), 794–802 (Aug 2012)
5. Chatzigiannakis, I., Hasemann, H., Karnstedt, M., Kleine, O., Kroller, A., Leggieri, M., Pfisterer, D., Romer, K., Truong, C.: True self-configuration for the IoT. In: 2012 3rd IEEE International Conference on the Internet of Things. pp. 9–15. IEEE (Oct 2012)
6. Daum, M., Götz, M., Domaschka, J.: Integrating CEP and BPM. In: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems - DEBS '12. pp. 157–166. ACM Press, New York, New York, USA (Jul 2012)
7. Di Giorgio, A., Pimpinella, L.: An event driven Smart Home Controller enabling consumer economic saving and automated Demand Side Management. Applied Energy 96, 92–103 (Aug 2012)
8. Domonte, E.P.: An Integrated and Low Cost Home Automation System with Flexible Task Scheduling. In: XV WORKSHOP OF PHYSICAL AGENTS. pp. 1–10. No. June, Leon (2014)
9. Etzion, O., Niblett, P.: Event Processing in Action. Manning (2010)
10. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley (2002)
11. Mohomed, I., Misra, A., Ebling, M., Jerome, W.F.: Harmoni: Context-aware filtering of sensor data for continuous remote health monitoring. In: Proceedings of Pervasive Computing and Communications (PerCom). pp. 248–251. IEEE Computer Society (2008)
12. Mouttham, A., Peyton, L., Eze, B., Saddik, A.E.: Event-driven data integration for personal health monitoring. Journal of Emerging Technologies in Web Intelligence pp. 144–148 (2009)
13. Rashidi, P., Cook, D.: Keeping the Resident in the Loop: Adapting the Smart Home to the User. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 39(5), 949–959 (Sep 2009)
14. Stipkovic, S., Bruns, R., Dunkel, J.: Event-based smartphone sensor processing for ambient assisted living. 2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS) pp. 221–227 (2013)
15. Vidačković, K., Renner, T., Rex, S., Fraunhofer IAO, S.: Marktübersicht Real-Time-Monitoring-Software: Event-Processing-Tools im Überblick. Fraunhofer-Verlag (2010), `http://books.google.de/books?id=rvbUXwAACAAJ`