Temporally-Relaxed Conditions for Activation of Services in the Web of Things

Martin Alexander Neumann
Karlsruhe Institute of Technology
TECO
Vincenz-Priessnitz-Str. 1
Karlsruhe, Germany
mneumann@teco.edu

Till Riedel
Karlsruhe Institute of Technology
TECO
Vincenz-Priessnitz-Str. 1
Karlsruhe, Germany
riedel@teco.edu

Daniel Hassler
Karlsruhe Institute of Technology
TECO
Vincenz-Priessnitz-Str. 1
Karlsruhe, Germany
hassler@teco.edu

Michael Beigl
Karlsruhe Institute of Technology
TECO
Vincenz-Priessnitz-Str. 1
Karlsruhe, Germany
beigl@teco.edu

Yong Ding
Karlsruhe Institute of Technology
TECO
Vincenz-Priessnitz-Str. 1
Karlsruhe, Germany
ding@teco.edu

Abstract

We present a language of temporal conditions for detecting concurrent events in embedded Web-enabled devices, and for triggering pervasive services on these systems. Based on the assumption that evaluating conditions on distributed devices is relevant for providing robustness and to foster scalability and web real-time, we discuss the feasibility of in-situ evaluation of the proposed conditions and conduct a performance study. In a smart environments use case, it is illustrated how the language can be used to state activation concerns of services on distributed Web resources. Our proposed architecture integrates the language with the Web of Things to foster simplified development of applications that mash up Web-enabled devices.

Author Keywords

Pervasive Service Engineering, Service Activation, Temporal Conditions, Web of Things

ACM Classification Keywords

D.2.6 [Programming Environments]: Integrated environments.; D.2.11 [Software Architectures]: Languages (e.g., description, interconnection, definition).

General Terms

Languages, Performance

Introduction

The Web of things (WoT) turns all devices in our environments into first-class citizens of the Web and interconnects them to deliver added value to our personal lives and businesses. Devices subscribe to data streams of other devices and evaluate conditions on data to detect events. In pervasive service engineering, a problem in implementing applications that detect concurrent events in their history could be formulated as follows:

```
if something
      has happened roughly at the same time then
    if something
      holds now then
        tell something to a user or a device
    end if
end if
```

Applications, such as notification and reminder services, may consider current and past states in their actions. For example, in a smart home environment, a monitor for intermittent ventilation (i.e. fully open a room window for about 15 minutes max) may be implemented by the following sufficient condition: "if a window has been open in the last 15 minutes, and it is still open now, remind the inhabitants to close this window". Or in a smart office environment, a meeting reminder's condition might be: "if the projector has been switched off and has been cold at the same time during the last 5 minutes, and it is switched on now, remind the participants to join".

To implement the first example, a system must be able to remember what states were present in the past and it must be able to check the current state. Furthermore, the second example demonstrates a system's ability to test whether two states were present at the same point in time or over the same period of time.

Whether conditions apply could be evaluated in central locations or on the distributed devices themselves. The important aspect is that the locations at which conditions are evaluated have notions of time to temporally relate states with each other. We propose to determine time information on streamed data locally at receiving devices without enforcing time synchronization, and to formulate *temporally-relaxed conditions* on them. This loosens the coupling between devices, makes them resilient to system failure, and it provides scalability "with the thing" as locality between sending and receiving devices is benefited from. Whether this approach provides sufficient quality to pervasive services in general is still to be evaluated.

We expect pervasive services to be implemented on the WoT, as it provides an application platform that integrates heterogeneous devices for simplified application development. Detecting events by which an action is triggered is a common concern. To focus on it, we propose to separate an application into its services and their *activation conditions*. Each application could programmatically implement interactions with Web resources and evaluation of conditions in a platform's native programming paradigm. However, we propose to constrain the programming paradigm by modularizing the activation concern and by providing a programming framework for formulating conditions to detect concurrent events on Web resources.

It is supposed to streamline and ease the development of pervasive services on Web-enabled devices for programmers and users: (1) by providing a platform-independent and common-style separation of activation concerns, and (2) by supporting Web real-time capabilities based on the featured space and time complexities in condition evaluation. The first goal is still to be evaluated. Here, we focus on the performance and the framework's applicability to pervasive applications based on a smart environments use case. Our implementation features an external domain-specific language (DSL) to implement the framework of activation conditions. The DSL is defined on top of URIs and simple JSONiq[1]-style queries (JSONiq selectors and comparisons).

Our paper is structured as follows. Firstly, we discuss relevant work from the domains of device integration in the Web, end-user programming and temporal queries on streams. Secondly, the data model, syntax and semantics of our proposed language are presented. Afterwards, our prototypical implementation is illustrated, which is based on node.js[13] and cloud9[2]. Finally, we evaluate our claims by examining the implementation's performance and by presenting a smart environments use case.

Related Work

"Putting the internet to work for you" is the slogan of the cloud-hosted IFTTT[3] service, which provides a centralized rule engine that routes data streams. Each rule has a source and a target stream, by means of which it interconnects two cloud-services, such as feeds, e-mail, etc. In addition, Web-enabled devices, such as ambient lights, are integrated. As IFTTT rules subscribe to a single source only, they do not have to cope with timing of several unsynchronized streams.

The idea of centrally-hosted workflows composing services of Web-enabled devices is conceptualized in [5]. According to their architecture, to provide a scalable application/integration layer, devices are uniquely addressable and provide resource-oriented service interfaces, which is implemented by REST. This idea is focused towards cloud-hosted RESTful programs in [7].

[1]http://www.jsoniq.org
[2]http://c9.io
[3]http://ifttt.com

Furthermore, [8] takes the idea of cloud-hosted application logic for devices in the Web to the extreme. In contrast to IFTTT, which solely mashes up given cloud-services and integratable devices, the authors promote to move any application logic from devices onto central engines. Devices only expose a RESTful interface to their "bare metal".

In contrast to these approaches, we keep application logic close to devices. This decision is based on the assumption that applications hosted on central engines have a relevantly different perspective on timing of data streams than applications on the devices. This is a problem to applications that have subscribed to streams of several remote data sources and which would like the device to react in case some temporal situation in receiving data has occurred. This claim is still to be evaluated.

An expressive query language on linked data with flexible window-based temporal operators is presented in [9]. Their language, called CQUELS, is an extension to SPARQL and enables to formulate temporal queries on streams of RDF data. A notion of relaxed concurrency is not a native part of their language. Nevertheless, our relaxed notion of concurrency could have been implemented as an extension to their language or based on top of it, i.e. translating our activation conditions into CQUELS queries.

This holds analogously for other languages which provide temporal operators and are more expressive than our proposal. For example, our mapping of relaxed discrete time to continuous time could have been expressed using one of the contextual operators of the language for context modeling presented in [12].

We extend our previous work on simplifying programming of Web-enabled devices. In [3], the idea to bundle development and runtime environment onto a sensor node has been presented. Any application logic of nodes is programmed in an unstructured profile of the BASIC programming language with domain-specific libraries.

Approaches such as hawtio[4] on top of Apache Camel[6], ClickScript[4], DERI Pipes[10], Yahoo! Pipes[1], or WireIt[5] mash up components or streams, such as feeds, in visual workflows. These components can either be provided by libraries, automatically generated from virtual or physical systems in the environment, or be user-programmed. Furthermore, the workflows may either be run on centralized engines, the typical case, or on the distributed devices themselves.

Component frameworks have been proposed for providing a modularized programming model for applications in the WoT. For example, using OSGi, Java-based services exposing RESTful interfaces could be integrated, while providing software life-cycle management [2]. Another recent approach to centralized user-friendly programming of scalable applications on devices in the Web is based on a REST framework for JavaScript (JS)[7].

Our implementation follows the idea of JS components running on the Web. Applications are packaged into node.js bundles which are, then, deployed to devices by a central development environment or an application store. The JS environment is augmented by our DSL to trigger services whenever certain conditions in the environment of a device hold.

[4]http://hawt.io
[5]http://dev.lshift.net/james/wireit/wireit

Activation Conditions

As mentioned earlier, conceptually, each distributed device integrates its own local sensor data with data streams of other remote devices. Since our devices are connected by the Web, all these streams are necessarily unsynchronized. A device's exemplary perspective is schematically shown in figure 1.

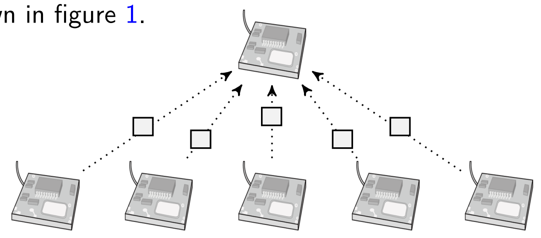


Figure 1: A Device's Perspective on Data Streams

In the following section, a model from this kind of device perspectives is presented. The model establishes a relaxed notion of *two to many events (whose occurrence are reported by streamed data) "occurred at the same point in time"*. Subsequently, a succinct language to formulate logical conditions on these events is presented, which is restricted to the language features motivated before.

Data Model

Our model focuses on the perspective of individual devices on the Web. A device subscribes to a number of streams which are represented by a set S of symbols, as shown in figure 2. A symbol represents a query on a Web resource which evaluates to a boolean, and its evaluations form a stream. The model abstracts from the the details of a query, but it could be composed of an URI, a JSONiq selector, and a JSONiq comparison. For example, "`http://teco/kitchen.temp gt 17`".

$$S = \{s_0, s_1, \ ..., \ s_n\} \tag{1}$$

Figure 2: Set of Known Symbols

New data can arrive on all streams at any time and are unsynchronized. These updates are modeled as elements $su_i = (s_i, t_i)$ of set Sus. Each element contains the updated symbol s_i and a continuous time stamp $t_i \in \mathbb{R}^{\geq 0}$. Formula (4) establishes a linear order of all updates by their time stamps, as depicted in figure 3. t^{now} represents the current point in time.

$$Sus = \{su_0, su_1, \ ..., \ su_n\} \subset S \times \mathbb{R}^{\geq 0} \tag{2}$$

$$su_i = (s_i, t_i) \quad | \ s_i \in S, \ t_i \in \mathbb{R}^{\geq 0} \tag{3}$$

$$\forall su_i, su_j : i \leq j \Leftrightarrow t_i \geq t_j \tag{4}$$

$$t^{now} \in \mathbb{R}^{\geq 0} \wedge t^{now} \geq t_0 \tag{5}$$

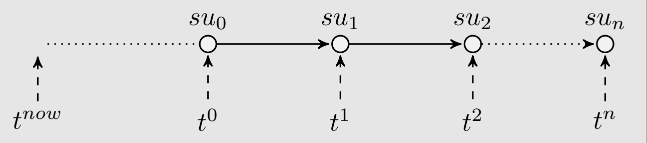


Figure 3: Continuous-Time Symbol Updates

To establish a notion of concurrency, the continuous timeline of updates is discretized using a constant step size t^{step}. This gives a discrete timeline of updates whose temporal granularity is defined by t^{step}. All updates occurring during the time window of one step are perceived to have happened in parallel. To account for updates occurring just at the border between two consecutive steps, the parameter t^{ϵ} is introduced. It defines the temporal overlap of consecutive steps. (The beginning and end of a step are given by $l(j)$ and $r(j)$.) The combination of the two *relaxation parameters* t^{step} and t^{ϵ} is schematically illustrated in figure 4. They allow to configure use case-specific interpretations of concurrency on the unsynchronized streams of updates given in a use case (or situation). Function $v(s_i, j)$, given in formula (8) and (9), provides access to the *temporally-relaxed discrete timeline of updates*. It defines whether symbol s_i has been updated at time step j.

$$t^{step} \in \mathbb{R}^{\geq 0} \tag{6}$$

$$t^{\epsilon} \in \mathbb{R}^{\geq 0} \wedge \ t^{\epsilon} \ll t^{step} \tag{7}$$

$$v(s_i, j) \mapsto \{\top, \bot\} \quad | \ s_i \in S, \ j \in \mathbb{N}^0 \tag{8}$$

$$v(s_i, j) = \begin{cases} \top, & \text{if } s_i \in \{s_k \mid (s_k, t_k) \in Sus, \\ & \qquad l(j) \leq t_k \leq r(j)\} \\ \bot, & \text{else} \end{cases} \tag{9}$$

$$l(j) = t^{now} - [j \cdot (t^{step} - t^{\epsilon})] \tag{10}$$

$$r(j) = t^{now} - ([j \cdot (t^{step} - t^{\epsilon})] + t^{step}) \tag{11}$$

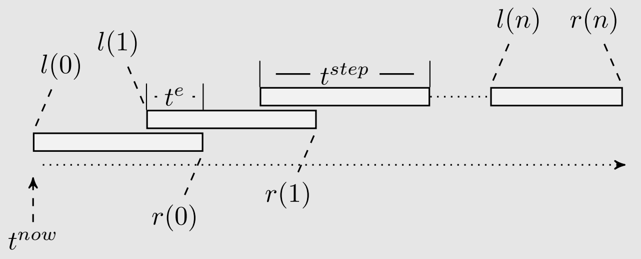


Figure 4: Relaxed Discrete-Time Symbol Updates

Notably, as shown in figure 4, j marks the j-th past time step. For example, $v(s_3, 10)$ assesses whether an update occurred ten time steps ago to symbol s_3. If the model in this situation would be configured to $t^{step} = 13 \ mins$ and $t^{\epsilon} = 3 \ mins$, $v(s_3, 10)$ would be used to determine whether an update has been occurring to s_3 between 100 to 110 minutes ago.

Language

As shown by the grammar in figure 5, we provide a language of fundamental logical connectives ($\wedge, \vee, \neg$) to assess whether some situation holds. In addition, as motivated earlier, we incorporate a "some situation held in recent history" operator ($\uparrow^k$) to be able to contrast what has been in some point in time in the past and what is now. We show, in the next section, that this operator can be efficiently implemented.

$$
\begin{array}{rclccl}
\mathcal{AC} & ::= & \mathcal{AC}_f & & & \\
\mathcal{AC}_f & ::= & \mathcal{AC}_f & \wedge & \mathcal{AC}_f & \\
 & | & \mathcal{AC}_f & \vee & \mathcal{AC}_f & \\
 & | & & \neg & \mathcal{AC}_f & \\
 & | & & \uparrow^k & \mathcal{AC}_t & (k \in \mathbb{N}^0) \\
 & | & & & \mathcal{AC}_t & \\
\mathcal{AC}_t & ::= & \mathcal{AC}_t & \wedge & \mathcal{AC}_t & \\
 & | & \mathcal{AC}_t & \vee & \mathcal{AC}_t & \\
 & | & & \neg & \mathcal{AC}_t & \\
 & | & & & s_i &
\end{array}
$$

Figure 5: Abstract Syntax

Using the connectives and atomic symbols s_i, terms $\mathcal{AC}_t$ can be formed. The connectives and the temporal operator can be used to form formula $\mathcal{AC}_f$ from terms. This design ensures that sentences are only valid in our language if a $\uparrow^k$ operator does not occur in the scope of another $\uparrow^k$ operator. Consequently, any combination of symbols and connectives can only be subject to zero or one $\uparrow^k$ operator. In other words, given the following semantics, the syntax will already ensure that situations (composed of logical connectives) are evaluated only on one temporal context (window of past time). This prevents long durations in evaluation of unfavorably engineered formulas.

$$
\begin{array}{rcl}
[\![\mathcal{AC}_f]\!] & = & [\![\mathcal{AC}_f]\!]_0 \\
[\![\mathcal{AC}_{f/t} \wedge \mathcal{AC}_{f/t}]\!]_j & = & [\![\mathcal{AC}_{f/t}]\!]_j \wedge [\![\mathcal{AC}_{f/t}]\!]_j \\
[\![\mathcal{AC}_{f/t} \vee \mathcal{AC}_{f/t}]\!]_j & = & [\![\mathcal{AC}_{f/t}]\!]_j \vee [\![\mathcal{AC}_{f/t}]\!]_j \\
[\![\neg\, \mathcal{AC}_{f/t}]\!]_t & = & \neg\, [\![\mathcal{AC}_{f/t}]\!]_j \\
[\![\uparrow^k \mathcal{AC}_t]\!]_j & = & [\![\mathcal{AC}_t]\!]_0 \vee \ldots \vee [\![\mathcal{AC}_t]\!]_k \\
[\![s_i]\!]_j & = & v(s_i, j)
\end{array}
$$

Figure 6: Denotational Semantics

Figure 6 shows our semantics based on Propositional Logic. In general, formula $\mathcal{AC}_f$ and terms $\mathcal{AC}_t$ are evaluated in a temporal context j which is defined to be the j-th past step in the discrete time model. The initial context of a formula is $j = 0$. It is evaluated whether the formula is satisfied in the current step. A term in the scope of a $\uparrow^k$ operator is evaluated in the current step and the past k steps. The temporal $\uparrow^k$ operator expands to a set of alternatives that assesses whether a term is satisfied in the current or any of the past k steps. As shown in figure 6, this means that it is checked whether a term is satisfied in the current step, *or* in the step before, ..., *or* in the k-th step before. The order of evaluation is not defined and offers opportunities for optimizations in the implementation.

Complexity

Evaluating an activation condition requires its parsing and interpretation. Parsing is done in an offline phase, therefore its time and space complexities are not relevant. Furthermore, the offline phase produces a data structure for evaluation of a formula at runtime (online phase), such as a binary parse tree, which is appropriate to evaluate a formula, given our semantics, using a canonical recursive algorithm with linear worst-case time and space complexities. On each inner node of the tree, i.e. on each operator, the algorithm evaluates one or two subtrees first and afterwards the current operator. On each leaf, i.e. on each symbol s_i, the algorithm evaluates its value in the current temporal context j by a lookup using function $v(s_i, j)$, which requires constant time and space.

Without the $\uparrow^k$ operator, our sentences would be propositional sentences and the worst-case time and space complexities for evaluation using the sketched data structure and algorithm would be $\mathcal{O}(n)$, with n being the number of operators in a sentence. The required length of formulas is use case-specific, but decided at design or configuration time of applications and therefore not suspect to unanticipated changes at system runtime. Platforms could safely restrict their longest supported formula to prevent performance penalties in rule evaluation.

With the $\uparrow^k$ operator, each term in a formula would in worst-case have to be evaluated *exactly once* on the history of k steps before t^{now}. The complexities for evaluation would be $\mathcal{O}(n \cdot k)$, with k being the number of steps before t^{now}. As with n, the maximum required k is use case-specific, but decided at design or configuration time of applications (given by the maximum of all $\uparrow^k$ operators). Therefore, to make the rule evaluation tractable in general, k has to be bound platform-specifically, too.

Implementation

We chose node.js as underlying framework for application development. Node provides a JavaScript runtime and an event-driven programming model. Besides this, a large ecosystem of libraries and frameworks is available, as for example, a RESTful programming model (restify[6]). In addition, node provides package management (npm[7]).

[6] http://mcavage.github.io/node-restify
[7] http://npmjs.org

System Architecture

Our activation conditions DSL has been implemented using the jison[8] parser generator. Furthermore, we have developed three REST Web services (appool, logitag and berry). In general, the services are not restricted to be run on the same physical machines, neither do they have to run on different ones. Figure 7 shows the interaction between the services, with the appool being centralized. Zeroconf is used for service discovery.

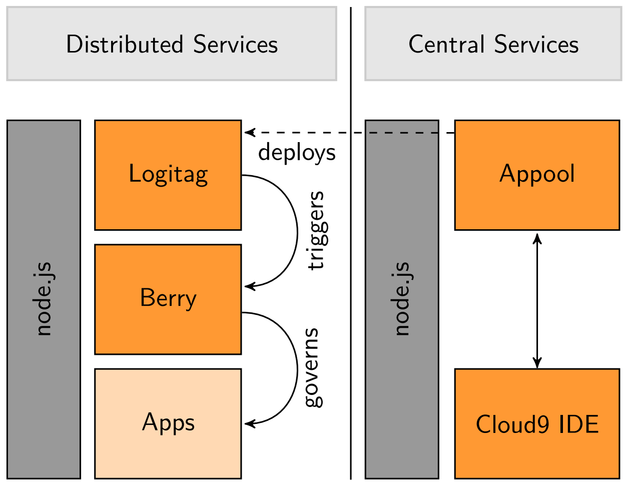


Figure 7: Architecture

The appool Web service manages npm packages in an "app store" way. Packages can be uploaded, and afterwards are available for download. Those packages contain the illustrated activation conditions. A list of available packages can be inspected as well as details on packages.

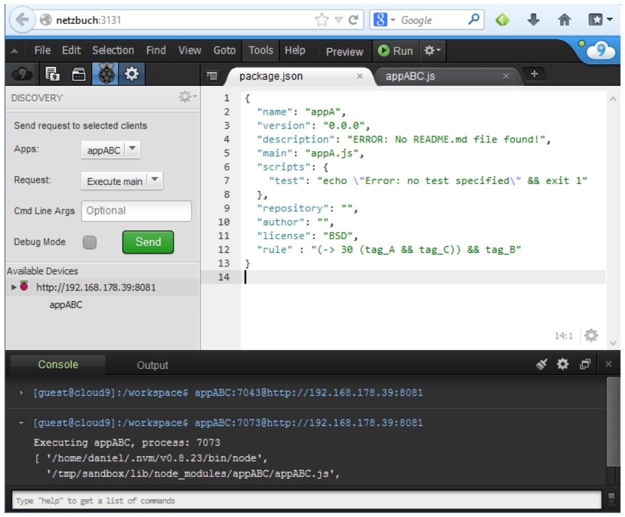
Berry extends the appool service with the ability to run applications from installed npm packages. For every ex-

[8] http://zaach.github.com/jison

ecuting application, a new resource, which streams the standard output from the belonging process, is created. A list of running processes can be obtained and resources can be deleted, which will kill the process.

The logitag service checks the activation conditions and runs applications on known berry services accordingly. Therefore local or remote context information are streamed to the logitag service, which then get evaluated with the implemented DSL. To implement this, our platform features an API to access data in the SmartTECO system[11]. Logitag needs to have access to an appool service as an application source.

Integration into Cloud9



```
{
  "name": "appA",
  "version": "0.0.0",
  "description": "ERROR: No README.md file found!",
  "main": "appA.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": "",
  "author": "",
  "license": "BSD",
  "rule" : "(-> 30 (tag_A && tag_C)) && tag_B"
}
```

```
[guest@cloud9]:/workspace$ appABC:7043@http://192.168.178.39:8081

[guest@cloud9]:/workspace$ appABC:7073@http://192.168.178.39:8081
Executing appABC, process: 7073
[ '/home/daniel/.nvm/v0.8.23/bin/node',
  '/tmp/sandbox/lib/node_modules/appABC/appABC.js',

Type "help" to get a list of commands
```

Figure 8: Cloud9 Interface

Figure 8 shows the integration into the Cloud9 IDE. Discovered berry service are shown on the left. The opened file "package.json" contains all the information for npm to create packages out of source files. We added one more property "rule" which contains the activation condition written in our DSL. Developers are able to create new node applications in their workspaces and augment them with activation conditions. For development and debugging purposes a user may select one of the available applications in her workspace and execute them directly on specific berry services. The standard output of the application gets streamed back to the Cloud9 console as shown in the screenshot.

Evaluation

Performance

A complexity of $\mathcal{O}(n \cdot m)$ for interpreting our temporal rules has been sketched earlier. To validate this claim on our implementation, performance measurements on a node-compatible embedded-class machine (Raspberry Pi[9]) have been taken. Figure 9 shows the results of the runtime required to evaluate a query. n ranges from 1 to 100 in steps of 10, k ranges from 1000 to 10000 in steps of 1000, and the z-axis shows the average duration of 10 query evaluations.

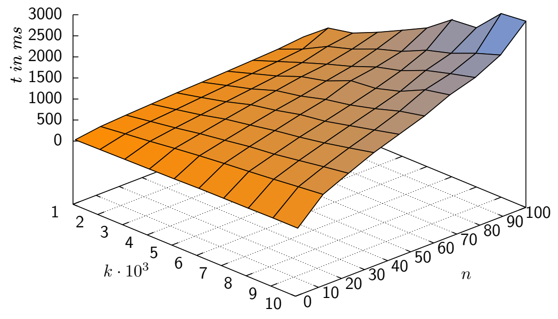


Figure 9: Sentence Evaluation Performance

[9]http://www.raspberrypi.org

Both variables impose a linear dependency on required runtime and memory. Therefore, if in general m and n are properly bound as mentioned before, it is feasible to evaluate programs in our DSL on devices in the Web.

Use Case

Julia works in a smart office. Whenever a local is starting, she wants to be notified of it. This reminder condition could be sufficient to her: "if a meeting is starting, and Julia is logged into the domain, but she has not yet entered the meeting room, then remind her to join".

In our system, Julia is represented by a Web resource that yields her location and whether she is on the domain. Our meeting room's Web resource does not tell whether a meeting is about to start. But we can mash-up our projector with a sensor node in an activation condition. Now, a condition to detect a starting meeting could be: "if the projector has just been switched off and has been cold (less than $40°$ C) at the same time, and it is switched on, then a meeting is approaching". We have selected 2 minutes to be a sensible duration for propagating the starting condition of a meeting. This can be temporally-related to other situations in recent history, for example, Julia being at the office but not in the meeting.

```
step=70s, epsilon=10s,
julia = http://teco/employee/julia,
proj  = http://teco/meeting/projector,
upart = http://teco/upart/1.2.3.4.0.6.1.133,
->2   (!proj.on & upart.temp lt 40) &
->now   proj.on &
->now (!julia.loc eq ‘meeting’ & julia.login)
```

Figure 10: Meeting Reminder Service Activation Condition

The application is separated into two concerns: (1) notifying Julia, and (2) recognizing the situation that triggers the notification. Figure 10 shows an exemplary translation of the previous conditions into the proposed language. The first two detect the event of a starting meeting, which will last for 2 minutes. The third detects whether Julia is not in the meeting room, but logged into the domain.

We implemented a small notification service that lists details about the people currently in the meeting room. The code of the service takes 5 lines of code. Our external DSL adds 3 activation conditions. For comparison we wrote a service, that does not use the DSL, but a library implementing the framework. Although we tried to factor out most of the logic, the service took 14 lines of code.

Conclusion

In this paper, a notion for relaxed concurrency in the WoT and a language of temporal conditions have been proposed. An external DSL has been implemented to use these conditions in designing pervasive services on Web-enabled devices.

The language enables to state expressions on simultaneous events in the past and in the current moment. They serve as triggers of actions. It is an extension of the prominent "if this, then that" programming paradigm.

Evaluating the feasibility of the concurrency notion and the language on the WoT is in an early state. We have discussed the feasibility of evaluating activation conditions down to a certain class of embedded systems. And, we have illustrated the language's expressivity in a smart office scenario. Our future plans are to (1) provide a graphical syntax to our language using ClickScript, (2) investigate optimization potentials provided by our language in evaluating sentences of several applications running on a single system, and (3) implement additional pervasive services in our DSL.

Acknowledgments

The authors would like to acknowledge the funding by the German Federal Ministry of Education and Research as part of the UHUS project (grant number 01IS12051).

References

[1] Fagan, J. C. Mashing up Multiple Web Feeds Using Yahoo! Pipes. *Computers in Libraries 27*, 10 (2007), 10–17.

[2] Flotynski, J., Krysztofiak, K., and Wilusz, D. Building Modular Middlewares for the Internet of Things with OSGi. In *The Future Internet*, A. Galis and A. Gavras, Eds., vol. 7858 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, 200–213.

[3] Gordon, D., Beigl, M., and Neumann, M. Dinam: A wireless sensor network concept and platform for rapid development. In *Networked Sensing Systems (INSS), 2010 Seventh International Conference on* (2010), 57–60.

[4] Guinard, D. Mashing up your web-enabled home. In *Proceedings of the 10th international conference on Current trends in web engineering*, ICWE'10, Springer-Verlag (Berlin, Heidelberg, 2010), 442–446.

[5] Guinard, D. *A Web of Things Application Architecture – Integrating the Real-World into the Web*. Ph.d., ETH Zurich, 2011.

[6] Ibsen, C., and Anstey, J. *Camel in Action*, 1st ed. Manning Publications Co., Greenwich, USA, 2010.

[7] Kovatsch, M., Lanter, M., and Duquennoy, S. Actinium: A RESTful Runtime Container for Scriptable Internet of Things Applications. In *Proceedings of the 3rd International Conference on the Internet of Things (IoT 2012)* (Wuxi, China, Oct. 2012).

[8] Kovatsch, M., Mayer, S., and Ostermaier, B. Moving Application Logic from the Firmware to the Cloud: Towards the Thin Server Architecture for the Internet of Things. In *Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012)* (Palermo, Italy, July 2012).

[9] Le-Phuoc, D., Dao-Tran, M., Parreira, J. X., and Hauswirth, M. A native and adaptive approach for unified processing of linked streams and linked data. In *Proceedings of the 10th international conference on The semantic web - Volume Part I*, ISWC'11, Springer-Verlag (Berlin, Heidelberg, 2011), 370–388.

[10] Le-Phuoc, D., Polleres, A., Hauswirth, M., Tummarello, G., and Morbidoni, C. Rapid prototyping of semantic mash-ups through semantic web pipes. In *Proceedings of the 18th international conference on World wide web*, WWW '09, ACM (New York, NY, USA, 2009), 581–590.

[11] Namatame, N., Ding, Y., Riedel, T., Tokuda, H., Miyaki, T., and Beigl, M. A distributed resource management architecture for interconnecting Web-of-Things using uBox. In *Proceedings of the Second International Workshop on Web of Things*, WoT '11, ACM (New York, NY, USA, 2011), 4:1–4:6.

[12] Schmidtke, H. R., Hong, D., and Woo, W. Reasoning about Models of Context. A Context-Oriented Logical Language for Knowledge-Based Context-Aware Applications. *Revue d'Intelligence Artificielle 22*, 5 (2008), 589–608.

[13] Tilkov, S., and Vinoski, S. Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing 14*, 6 (Nov. 2010), 80–83.