ASL: A Domain Specific Language to Specify Complex Alarm Conditions in the Energy Domain

Alberto Carvalho
albertofilipecarvalho@tecnico.ulisboa.pt
Instituto Superior Técnico, Lisboa, Portugal

ABSTRACT

In energy management timeliness is a fundamental aspect, mainly because faster problem detection enables faster decision making and timely actions in order to solve problems. The combination of energy management systems (EMS) with alarms makes it possible to detect and report anomalies to building energy managers. Effective alarms design typically involves a great deal of domain knowledge and also programming hours — specially when complex conditions are involved. Indeed, EMS alarm definitions require specifying conditions over streams of data collected from meters and sensors for which no adequate systematic solution exists yet. This work aims to create a solution to support building energy managers in the specification of domain specific alarms involving complex conditions over energy meters and sensor data. This solution will enable users to create new alarms that capture complex patterns over streaming data through a graphical interface without the need to modify the application code. Using this solution building energy managers are able to get more control regarding existing alarms and conditions triggering them. To validate this solution we performed usability tests.

1. INTRODUCTION

Up-to-date information improves the decision making process, mostly because building energy managers are able to diagnose and react quickly to anomalous situations [1]. A crucial aspect in energy management is timeliness. As time passes, the window of opportunity for action shrinks and consequently the value of the information obtained from data decreases [2, 3]. Figure 1 illustrates this idea. Due to this fact, anomalies should be discovered as soon as possible — if possible in real-time or in near real-time — enabling the implementation of a corrective action [2, 3, 4, 5, 6].

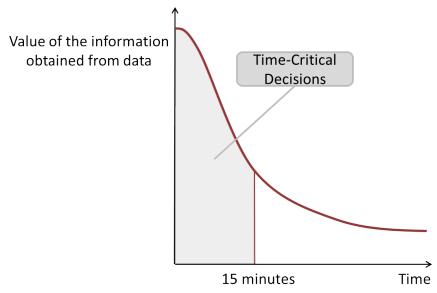

Figure 1: Value of the information obtained from data as a function of time (adapted from [2, 3]).

Previous research has demonstrated that the exploration of time series data, related with energy consumption, for anomaly detection enables reduction in the amount and cost of consumed energy [7]. This task can be done through anomaly detection that refers to the detection of abnormal situations or, in other words, to the detection of events or situations not behaving as expected [8, 9]. Alarms are fired when an anomaly is detected, thus enabling building energy managers to decide and act faster.

An alarm is a notification that an anomaly has occurred and a user response is required [10, 11, 12]. The primary use of alarms is to react promptly in order to solve volatile issues, in other words, to solve problems that have a short lifespan. For instance, in the case of a building, an alarm caused by a variation in the consumption patterns can alert the building energy manager that something is not working in the expected way, like a room with its lights turned on and nobody inside it, or a damaged equipment that is consuming more than usual. Real-time data processing can be explored synergistically with alarms to reduce the time required by building energy managers to detect anomalies in energy and equipment usage. With this approach major savings can be achieved in energy consumption.

Alarms, or more precisely, alarm conditions need to be detected from data arriving in real-time. Stream processing engines (SPEs), for instance data stream management systems (DSMSs) and complex event processing systems (CEPSs), have been developed to overcome the multiple difficulties that arise in scenarios requiring large quantity of real-time data to be processed in real-time, such as the care of data produced by sensors. The main objective of DSMSs is to process multiple data streams as input and provide new data streams as a result. CEPSs treat the input as a stream of event notifications, which need to be filtered and combined to reason with what is happening in terms of higher-level events. SPEs provide stream query languages enabling the creation of continuous queries that can be used, for instance, to process data or for pattern matching [13, 14, 15, 16].

SPEs can be used by applications that monitor continuous data streams, called monitoring applications, in order to achieve the goal of up-to-date information [17]. Generally, monitoring applications monitor multiple data streams, leading to processing rates bigger than ones supported by a traditional database management system (DBMS). In order to provide timely results they should use a DSMS as an alternative to a DBMS [13, 14, 17, 18, 19]. Monitoring applications, using a DSMS as underlying technology, are used in other business environments like in the monitoring

of stock market trades [20], health sensor data [21, 22], social networks [23], and network traffic [24, 25, 26].

Alarm management when performed by building energy managers enable them to get enhanced control regarding existing alarms and the conditions triggering them. This is an important task when it is necessary to redefine an alarm — for instance to modify the alarm trigger condition — or to create a new one. Thus, considering *(i)* the capability of a DSMS to timely process energy data — data streams — and *(ii)* the possibility to deploy on the fly a query — corresponding to the alarm trigger condition — on a DSMS, we can conclude that a DSMS is an adequate solution to timely detect the alarm conditions.

Domain specific languages (DSLs) are languages designed specifically for a domain [27, 28]. Using domain notions a DSL becomes more expressive, resulting in a language that is easier for users to learn, understand, and apply for a specific problem category [27, 29, 30]. Using the results of a systematization of alarm conditions, performed through a literature review and interviews with experienced building energy managers, this work aims to develop a visual DSL to specify alarms regarding the building energy management domain.

Additionally, using a DSMS as underlying technology, this work aims at creating an alarm system to support building energy managers to specify domain specific alarms involving complex conditions. This system will allow users to create new alarms just using the visual DSL on an interface without the need to modify the application code. An advantage of this system is that building energy managers will be able to get enhanced control regarding existing alarms and conditions triggering them. Another possible advantage will be the reduction of the alarms number, helping building energy managers with the immense quantity of generated alarms.

The proposed solution was validated through the validation of both the DSL and the alarm system developed. To validate the DSL we performed interviews with experienced building energy managers, using a DSL validation methodology based on the literature that assesses the DSL effectiveness, efficiency, satisfaction, accessibility, reliability, and productivity. The alarm system was assessed through its usability and performance characteristics. The results show that the DSL is easy to use and has a good domain coverage, being the system easy to use and efficient in the production of information.

This manuscript is structured as follows. Section 2 introduces fundamental concepts and related work of alarms in the energy domain. Sections 3 and 4 outlines the architecture and implementation of the prototype solution. Section 5 describes the validation of the solution, and finally, Section 6 presents the conclusions.

2. BACKGROUND

To develop a DSL and an alarm system to support building energy managers in the specification of alarms it is necessary to analyse a series of themes. These include the requirements of processing data in real-time, domain specific languages, the alarm quality requirements, and finally a survey of existing systems.

2.1 Real-Time Data Processing Requirements

The correct operation of a real-time software system depends on both the results produced and the time taken to generate those results [31]. Due to the intrinsic nature of data streams and continuous queries to produce real-time results a system must meet the following assumptions [32]:

1. Real-time monitoring applications must be capable of processing data in a timely way, reacting quickly to unusual data values. This way, it is possible to detect anomalies in due time enabling users to react.

2. Data model and query semantics must enable order-based and time-based operations (e.g. know the tuples that have arrived in the last 10 minutes).

3. Due to performance and memory constraints and since data streams are transient, stream processing algorithms will see data only once.

4. The interval of arrival of tuples may change over time (stream rate). Due to this, long duration queries may find changes in system conditions all over their execution lifetimes, however these queries must provide a correct answer in a timely manner.

5. Since it may be impossible, in practice, to store a complete data stream in memory, summary structures must be used. As a consequence, the answers to certain queries may have to be an approximate one.

6. In order to have systems able to timely process data under overload periods it may be necessary to define a policy to discard input data.

7. Blocking operators that require processing all the input to produce any results should be avoided, otherwise the computation of a result for unbounded data streams would never return a result.

2.2 Domain Specific Languages

The interest in language engineering — in other words, the act of creating languages — has increased over the last years [33]. Consequently, the Software Language Engineering (SLE) emerged as an application of a systematic, disciplined, and quantifiable methodology to the development, usage, and maintenance of software languages [34]. Typically, a SLE starts with the domain analysis, in order to identify the domain concepts, followed by the language design that captures the domain concepts and their relationships. Finally, the language is implemented and documented [35].

Domain Specific Languages (DSLs) are languages designed specifically for a domain. In contrast with general purpose languages, DSLs are focused in the domain of the problem and not in the computational solution, therefore they are more expressive and easy to use [27, 28, 33, 36]. Using domain notions in the solution makes DSLs more expressive, resulting in a language that is easier for programmers and non-programmers to learn, understand, and apply for a specific problem category [27, 29, 30].

Another DLSs benefit is that despite reducing flexibility, DSLs are more productive and reliable [36, 37]. This fact enables to reach the market quicker, diminish the maintenance cost, and have a higher optimization potential [38]. DSLs are also recommended by their quality, due to the fact that they produce less errors and more efficient code [34, 36, 39].

2.3 Alarm Quality Requirements

Considering the EEMUA 191 guideline, good alarms must

Symptom	Description
Alarm floods	Occurs when a large number of alarms are generated during abnormal situations, overloading the system user
Cascading alarms	Group of alarms that always occur together
Nuisance alarms	Alarms that are annunciated excessively and unnecessarily
Standing alarms	Users are continuously overloaded with alarms even in normal situations
Usefulness alarm messages	Alarm messages do not provide useful information to the system user
Overload of high priority alarms	Excessive high priority alarms are presented to the user and due to this some of them are not be treated according to their priority

Table 1: Symptoms of unsatisfactory alarm management [41].

respect standard characteristics, such as: being timely, prioritizable, diagnosable, advisory, and relevant to the user [40]. An alarm system may not have a good alarm management when it shows symptoms related, for instance, with alarm floods, cascading alarms, or standing alarms (see Table 1) [41]. Despite the existence of some solutions, this problem is not solved yet [42].

Monitoring systems (as well as EMSs with alarm functionalities) generate a large number of alarms, overloading users with redundant information, a situation known as *alarm flooding* [43, 44, 45, 46]. The typical origin of alarm flooding occurs when a single fault or anomaly raises a lot of alarms in a short time interval [45, 47]. Another cause of alarm flooding is low quality data that often leads to false alarms [48].

Weak alarm management, especially in alarm flooding situations, hampers the capacity of users to accomplish their tasks. Indeed, it has been demonstrated that it is not possible to reliably interpret and act when a large number of alarms is generated [43, 47, 49]. These aspects have been studied in other areas such as industry [50, 51] and medicine [52], where it has been reported that users ignore or shut down the system in order to do their job correctly [53, 54]. Due to this, intelligent alarm processors have to address the following requirements [49]:

1. Reduce the number of alarms.
2. Give clues for the possible root cause of the problem.
3. Recommend a corrective action.

There are different alarm processing approaches that aim primarily at reducing the number of existing alarms, in order to present a manageable number of useful alarms to the user [55]. Some tools allow users to control what information is displayed using display selection and information masking [56]. Another class of techniques is related with alarm compression and alarm suppression. In *alarm compression* [10], multiple occurrences of the same alarm are replaced by a new generated alarm. *Alarm suppression* is a technique used to prevent the alarm indication to the user when the base alarm condition is true, examples of suppression methods are presented in Table 2 [12]. An example of alarm suppression use occurs in the presence of multiple alarms where only the one with the highest priority is displayed and the other ones are temporary inhibited [10].

Suppression method	Description
Shelving	A mechanism used by the user to temporally suppress an alarm
Suppressed by design	The alarm indication to the user is prevented by any system mechanism taking into account a condition different of the one used to define the alarm
Out-of-service	An alarm indication is temporally suppressed, for instance, during maintenance

Table 2: Methods for alarm suppression [12].

It is also possible to reduce the number of alarms by combining related alarms using grouping rules, applied for instance to the time between alarms and their types [45]. More informative alarm messages can be generated through the combination of simpler ones [49]. Nuisance alarms can be reduced using on-delay and off-delay techniques. The first technique is used to prevent unnecessary alarms when a threshold is temporally exceeded, being the alarm triggered only when the threshold is exceed during a specified period of time. The off-delay technique, is similar and is used to lock the alarm indication during a defined period after the alarm trigger condition becomes false [12].

Traditionally, alarms are showed on a summary list chronologically sorted. During alarm floods, this approach presents a huge usability problem because an alarm summary disappears in the display, faster than the user can process it. Other disadvantage of this speed occurs when the user do not see high priority alarms being these ones untreated. An user interface strategy to deal with alarm floods, consists of using two separated lists, or panels, where primary and secondary alarms are split into distinct windows [57].

2.4 Survey of existing EMS

A large number of EMSs are currently available in the market. However, many of the existing solutions are proprietary being unfeasible to acquire a paid license to evaluate each of them. Due to this, this survey is limited to the available documentation and overviews. In addition, the documentation provided by product manufacturers is scarce and rarely discloses any technical information. For example, the systems are classified as real-time, but the time scale of their timeliness is not specified. Other example is related with alarm personalisation and creation. Generally, manufacturers do not provide sufficient information about how users personalise and create alarms, instead they just state that these features are available without specifying to what extent.

This survey presents what we believe to be a representative sample of existing EMSs solutions on the market. The systems are evaluated by their ability to present results in real-time, provide FDD (fault detection and diagnosis, i.e. the process of detecting faults in physical systems and diagnose their causes [58]) and alarms. In this last feature, it is also analysed if the systems enable users to personalise or create alarms. The results of the survey are illustrated in

Systems	Features				
	Real-time Monitoring	FDD	Alarms	Alarm Personalisation	Alarm Creation
EEMSuite [59]	•†	•	•	—	—
EnergyWitness [60]	•†	—	—	—	—
EnerwiseEM [61]	○	•	•	○	○
OpenEIS [62]	○	•	—	—	—
ESightEnergy [63]	—	—	•	•*	○
Cylon Active Energy Manager [64]	•	•	•	•*	•*
Noveda EnergyFlow Monitor [65]	•	○	•	•*	○
Wisemetering [66]	—	○	•	○	○
Gridpoint EMS [67]	•	○	•	○	○

Table 3: Survey of Energy Management Systems, where • stands for supported, ○ stands for not supported, and - stands for unknown information. † systems that respond in near real-time. ∗ unknown how it is implemented.

Table 3.

In conclusion, to the best of our knowledge, most solutions do not allow users to create or personalise alarms. Another important aspect is related with the use of a DBMS or DSMS as a solution base. At least four of the surveyed solutions (EEMSuite, EnergyWitness, EnerwiseEM, and OpenEIS) have a DBMS as their main data processing infrastructure instead of a DSMS. This observation is in line with the results of this survey where most systems do not provide results in real-time.

2.5 Discussion

From our study of related work, it is possible to draw several conclusions. First, we can conclude that detecting alarm conditions in the energy domain include the following requirements:

1. The integration of several different data sources, in order to gather relevant data necessary to produce useful information.
2. Input data assumes the form of potentially unbounded data streams.
3. The detection of alarm conditions must occur in a timely way, preferably in real-time or near real-time.
4. It is necessary to run multiple alarms simultaneously.
5. Considering the huge number of possible alarm conditions it is necessary to consider several different mechanisms to detect them. An example of a possible mechanism is the detection of a value above a certain threshold.
6. The detection of an alarm condition, in addition to the data streams currently entering the system, may require some reasoning about historical data. An example occurs when an alarm has a condition like "*a sensor reports 3 times an unexpected consumption within the last 60 minutes*" where it is necessary to count the number of occurrences of a specific event.

As explained in Section 2.1, due to the challenges of dealing with a large volume of continuously incoming data in a data stream scenario, most of the traditional anomaly detection techniques (e.g. classification based techniques) cannot be directly applied [9, 68, 69]. Thus, it is possible to conclude that it is difficult to apply automatic techniques to define and detect complex alarm conditions. However, existing stream query languages (provided by SPEs such as DSMSs), and the operators supported by them (such as logic, sequence, and window operators), have enough power to express such complex conditions through continuous queries. In addition to this, DSMSs are able to process multiple data streams as input, to produce results in a timely way, and to run multiple queries simultaneously.

The different alarm processing approaches (presented in Section 2.3) are important to design and implement alarm systems, especially regarding user interfaces where alarms are presented. Despite they are not directly related with the specification of alarm conditions, these approaches (e.g. on-delay techniques) are useful in alarm specifications in order to reduce the number of alarms.

During the alarm management life-cycle it is necessary to modify or to create new alarms, preferably without the need to shut down the system [12]. Thus, considering the capability of DSMSs in deploying queries on the fly this requirement is satisfied.

Finally, we note that a DSMS is able to satisfy all the enumerated requirements about the specification and detection of alarm conditions. Considering this observation it appears to be a good approach to use a DSMS as underlying technology of a system that aims to support building energy managers to specify domain specific alarms involving complex conditions. However, to the best of our knowledge, these kind of solutions are not currently available.

3. SOLUTION

Considering the DSLs and DSMSs capabilities, this work aims to create a visual DSL and an alarm system to support building energy managers on the specification of domain specific alarms involving complex conditions. The alarm system has a DSMS as underlying technology, and on the system interface users use the visual DSL to specify alarm conditions.

3.1 Architecture Overview

Our solution is organized into three stages: information presentation, data processing, and data acquisition (see Figure 2). The information presentation stage is a sophisticated user interface comprising a Visual Alarm Editor and an Alarm Dashboard. Using the Visual Alarm Editor users can create alarms by dragging and dropping blocks, where each block corresponds to a language primitive. This language, which we refer as Alarm Specification Language (ASL), comprises a number of fundamental operators that enable users to specify alarms using complex conditions.

In the data processing stage, the alarm system translates the user specification into a continuous query representing the alarm trigger condition. This query is installed on a DSMS in order to be continuously evaluated, being the results of its evaluation forwarded to the alarm manager. This last module will process the query results generating an

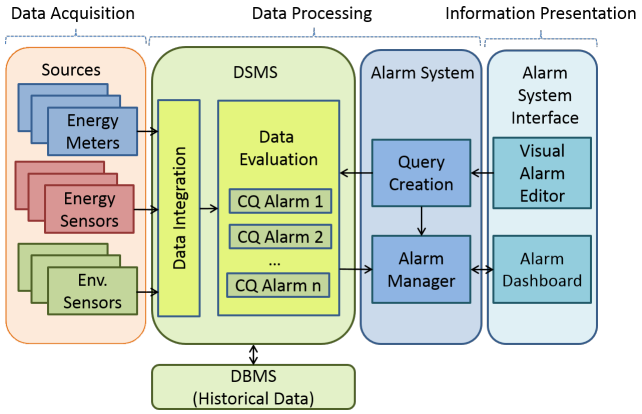

Figure 2: Solution Proposal Architecture. From left to right: the Data Acquisition Stage composed of data streaming sources; the Data Processing Stage responsible for the creation and evaluation of the continuous queries that represent the alarm conditions; the Information Presentation Stage presents the alarm dashboard and enables expert users to create new alarms.

alarm and related information, including the alarm timestamp, priority, and corrective action. The generated alarm and its information will be displayed in the alarm dashboard, enabling users to act and correct abnormal situations in due time. Finally, a DBMS is used in the data processing stage to provide persistent data management.

The data acquisition stage acquires data streams from several sources like energy meters, equipment sensors, and environment sensors. This data is forwarded to the data processing stage where it will be integrated and cleaned. The main purpose of this phase is to combine several data streams, in order to formulate a new set of data streams, with a well defined and agreed upon schema, underlying the semantics of the energy data problem domain.

3.2 Alarm Specification Language

According to Voelter et al. [36], the development of a DSL is an iterative process composed of four stages. These include a domain analysis and the design, implementation, and validation of the language.

To comprehend the domain we performed a systematization of alarm conditions, through a literature review and interviews with experienced building energy managers. Using the domain analysis results it was possible to define the language elements necessary to construct an alarm specification (i.e. an alarm trigger condition). These elements, described bellow, are divided in seven groups, namely: Source, History, Aggregation, Forecast, Threshold, Variation, and Optional Detectors. The first four groups enable the definition of the necessary data, being mandatory the use of a Source and an Aggregation. The last three groups are detectors (enable the detection of a condition), being the use of a Threshold or a Variation mandatory.

The language elements have input and/or output ports in order to share data between different elements. Additionally, they have fields where users introduce more data related with the alarm specification. To specify an alarm it is necessary to connect language elements, however it is only possible to connect ports of the same colour and of different language elements (when connecting History, Repeat, During, or Within). Accordingly, each group of elements has a different port configuration. The Sources have one blue output port; History elements have one input and one output port both with blue colour; Aggregation has one blue input port and one yellow output port; Forecast has one input port and one output port both yellow; Thresholds and Variations have multiple yellow input ports and one green output port. Finally, the Optional Detectors have one green input port and one green output port. The Occupation and Working Equipment elements are Optional Detectors, however they do not have input ports.

A **Source** enables users to determine the data source to be used in the alarm. The sources available include: Energy Consumption, Energy Cost, Energy Tariff, Voltage, Power Input, Produced Energy, Energy Sold, CO_2 Emissions, Interior Temperature, Exterior Temperature, Water Consumption, Gas Consumption, Interior Humidity, and Data Arrival Frequency. These elements, except the Energy Tariff, enable the selection of an id of an equipment or of a building zone, enabling the user to be more selective about the data sources. In addition to this, the Energy Sold element enables users to select if the energy sold data is about the sale value or the quantity of energy sold. Finally, the Energy Tariff element allows only to select the id of a building zone.

To obtain historical data there are two possible **History** elements. The Simple History element enables the creation of a sliding window. A Period History in addition to the Simple History behaviour enables the specification of a time period, enabling for instance to specify that one wants the historical data relative to the 10:00 to 17:30 period of the last month. Using the **Aggregation** element it is possible to specify how to aggregate data provided by the Source or History. One can aggregate data per minute, hour, day, week, month, and year using minimum, maximum, or average operators. A **Forecast** consists in estimating the future of the value received from the Aggregation element.

Regarding **Threshold** detectors, they detect that a certain limit is reached. There are three types of thresholds, namely: Lower Limit Threshold, Upper Limit Threshold, and Within Threshold. The first one detects that a lower limit is reached, i.e. a value decreases and reaches the limit. The Upper Limit Threshold detects that an upper limit is reached, i.e. a value increases and reaches the limit. Using the Within Threshold it is possible to detect when a value is inside a [lower limit, upper limit] range. These three types of thresholds can have different arity of input ports, namely unary and binary (ternary in the Within Threshold case). In case of unary Thresholds the user has to specify manually the limit's value. On the other hand, when they are binary the second input port receives the limit's value from the element connected to that port. The ternary Within Threshold is similar, however it receives the data to analyse on the first input port, receives the lower limit's value on the second port, and on the last input port receives the upper limit's value. When using binary or ternary thresholds the user can specify that the limit's value will be a percentage of the value received as input.

A **Variation** element detects fluctuations of a value received from the first input port. There are two types of Variation elements depending of its arity, namely unary and binary. Unary variation detects the variation of a value during a specified period of time, being possible to detect, for instance, that a value had an increase of 25% over the last

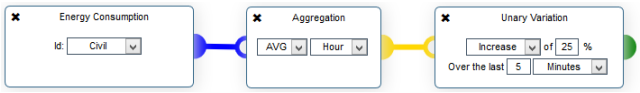

Figure 3: Example of an alarm specification whose purpose is to detect when the energy consumption of the zone with identifier Civil has an hourly average registering an increase of 25% over the last 5 minutes.

5 minutes, as depicted in Figure 3. In relation to the binary variation, it detects a variation comparing the value obtained from the first input port with the value obtained from the second input port. This way, one can detect, for example, that the value obtained from the first input port is 25% higher relatively to the value provided by the second input port.

Finally, optional detectors are a group of elements that enable the specification of alarm conditions with a higher level of abstraction. This group includes the Repeat, During, Within, Followed By, And, and Or elements. Additionally, there are two special elements, namely Occupation and Working Equipment, that work both as data sources and detectors.

Using a **Repeat** element it is possible to detect that a given situation occurred a given number of times in a sliding window. To detect that a condition is true during a given period of time one can use a **During** element. The **Within** element narrows the alarm condition to specific days or time periods, existing two types namely Simple Within and Period Within. The Simple Within enables the user to select the day (i.e. Sunday to Saturday) or day type (e.g. weekday, weekend, holiday, spring, summer, autumn, and winter) during which the alarm condition becomes true. In addition to this, the Period Within element enables the specification of a time period, for instance the 10:00 to 17:30 period.

A **Followed By** provides a way to specify a high level alarm condition that defines an order between the occurrence of sub-conditions. It has an input port where the inputs order specifies the order in which the sub-conditions must occur. Using this element it is possible to specify, for instance, that an alarm condition becomes true when a variation occurs followed by the excess of a threshold. To specify that a high level alarm condition becomes true when multiple sub-conditions are true simultaneously one can use an **And** element, where the sub-conditions are provided by the elements connected to the input port. Using an **Or** it is possible to specify that a high level alarm condition becomes true when one of multiple sub-conditions (provided by the elements connected to the input port) becomes true.

Finally, the **Occupation** element detects if a zone is empty or not, and the **Working Equipment** element detects a working equipment selected by the user. A possible way to detect that equipment is working consists in detect that its energy consumption is bigger than zero Watts. Using this language, a possible way to specify this condition consists in using three elements namely: an Energy Consumption source, an Aggregation — in order to get the maximum consumption, and an Upper Limit Threshold — to detect consumptions higher than zero. In order to make this language easier to use we developed the Working Equipment element as syntactic sugar to the previously mentioned condition. Using this element users are able to specify that an equipment is working in an easier and faster way, being the specification of erroneous conditions prevented.

4. IMPLEMENTATION

Considering the alarm specification as a specification tree, to convert it in continuous queries we use an algorithm based on the **Interpreter design pattern**, where each language element implements a method called *interpret*. According to Mernik et al. and Kosar et al., this design pattern is a possible approach to implement a DSL [30, 70]. This design pattern follows the Composite design pattern being possible to deal equally with the language elements, treating individual elements (the leafs) and element compositions uniformly [71]. Additionally, the use of the Interpreter design pattern increases the language maintainability, i.e. the degree to which a language promotes ease of program maintenance [72].

Since our solution depends on the ability to timely generate alarms, we use Esper [1] which is a DSMS able to process continuous queries over unbounded data streams. It also presents language capabilities for data and event stream processing as well as for pattern matching, which is necessary since it must be possible to translate an alarm specification defined by users in continuous queries using stream query language operators.

5. VALIDATION

Most of the requirements regarding the evaluation of user interfaces are associated with usability [73]. According to the ISO 9241-11 standard, *usability* is defined as the *"extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"* [74]. The ISO/IEC 25010 standard, expands this definition introducing the notion of *quality in use* that refers to the quality perceived by users during the product utilization in a real context of use [75].

The quality assessment of a DSL is a concept in development [72, 76]. However, since the main goal of a DSL is to close the gap between domain experts and a computational solution, a DSL can be used as a structured and comprehensive means to achieve human computer interaction [73]. Therefore, Barišić et al. taking in consideration the ISO/IEC 25010 standard propose that the most relevant attributes to evaluate the achieved quality in use of a DSL in a real context are the following ones [75, 77]:

Effectiveness determines the accuracy and completeness of the sentences implemented by users.

Efficiency states the effectiveness level that is achieved at the expense of resources, such as mental and physical effort.

Satisfaction expresses the satisfaction of the user needs when a system is used in a specified context of use.

Accessibility specifies the degree to which a system can be used by users with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use. Barišić et al. give emphasis to learnability and memorability of the language elements.

These attributes are also supported by other authors [28, 39, 72, 78]. Considering them we developed a set of questions to validate our DSL (i.e. the ASL). These questions are presented in Table 4.

Reliability is another attribute proposed in literature as contributing to the success of a DSL [39, 72]. It refers to a language property regarding the production of reliable programs, preventing errors through, for instance, the preven-

[1] `http://www.espertech.com/products/esper.php`

Question Description
Effectiveness **Q1** - Is it possible to specify all the alarms you need using this language?
Efficiency **Q2** - Are you able to specify alarms in a timely way?
Satisfaction **Q3** - Would you use this language in your EMS?
Accessibility **Q4** - Is it easy to specify alarms using this language? **Q5** - Is it easy to understand the goal/meaning of the language elements?

Table 4: ASL validation questions (based on [28, 39, 72, 77, 78]).

tion of unexpected relations. Having ports with colours, where it is only possible to connect ports of the same colour and of different language elements (when connecting History, Repeat, During, or Within), the ASL prevents error occurrences. Additionally, since our implementation verifies the language syntax we consider that the ASL ensures reliability.

Since we also developed an alarm system, we elaborated a group of questions to validate both the interface and the performance of the alarm system. The questions regarding the alarm system interface are based on the ISO 9241-11 standard [74], which is the standard for ergonomics of human-computer interaction. These questions were prepared taking in consideration five fundamental aspects for a good interface construction, namely: design, functionality, easiness of use, learnability, and satisfaction. Finally, there are two questions to measure if the system performance is acceptable according to users needs. These questions pretend to validate that the time required to convert an alarm specification in continuous queries, and the system performance during the production of information (e.g. regarding existing and active alarms) respects the users requirements. The complete set of questions used during the alarm system validation is depicted in Table 5.

5.1 Methodology and Participants

To validate both the ASL and the alarm system developed, we performed several user interviews each one structured in two stages. The first one to validate the ASL and the second to validate the alarm system. In each interview we performed the following steps: *(i)* a questionnaire to establish the user profile; *(ii)* a brief language explanation to users through a set of examples of alarm specifications using the ASL; *(iii)* a questionnaire with the questions presented in Table 4; *(iv)* a briefing about the alarm system interface where users freely explored the interface; and *(v)* a questionnaire with the questions presented in Table 5.

Involving real users is a major aspect of this validation. Considering this, we preferred to have a small number of participants that are real users, i.e. are building energy managers, instead of involving ad-hoc users in the tests, which could produce biased results. In addition to this, due to the relatively high costs associated it is difficult to arrange meetings with real users (for instance each building energy manager has an hourly cost and assigned work to do during its work period). Due to this fact and because users do not want to perform usability tests where they have to do use case scenarios, we cannot gather metrics such as the time spent on each task, number of mistakes, and the number of accomplished tasks. However, to minimize this problem we question users if they are able to specify alarms in a timely way.

Question Description
Design **Q1** - Is the system interface pleasant to use?
Functionality **Q2** - Is the information provided by the system effective at helping to perform tasks?
Easiness of Use **Q3** - Is the system interface easy to use? **Q4** - Is it easy to find the information you need?
Learnability **Q5** - Is it easy to learn how to use the system interface? **Q6** - Is the information provided by the system easy to learn?
Satisfaction **Q7** - Can you fulfil your tasks effectively? **Q8** - Are you satisfied with the outcome of the performed tasks?
Performance **Q9** - How do you classify the system performance in the alarm creation (i.e. conversion of the alarm specification into continuous queries)? **Q10** - How do you classify the system performance in the production of information?

Table 5: Alarm system validation questions, organized according to ergonomic dimensions of ISO 9241-11 standard [74].

Regarding the profiles of the ten participants, most of them are graduated males and have more than ten years of experience in the building energy management domain. The type of buildings managed include educational buildings (e.g. universities), commercial buildings (i.e. companies/governmental offices, stores, or both in the same building), and museums (i.e. buildings serving both as museums and spaces for conferences/presentations). Some participants manage all the buildings of its institution (like banks and foundations), existing for instance one that manages 700.000 m^2 spread across 850 buildings, other managing 500 buildings, two managing 110.000 m^2 each, and other managing 75.000 m^2.

5.2 Results

Regarding the language validation most participants gave the highest score. Therefore, we can conclude that the ASL: *(i)* covers the building energy management domain, allowing the specification of the necessary alarms; *(ii)* enables the specification of alarms in a timely way; *(iii)* satisfies the participants needs; *(iv)* is easy to use to specify alarms; and *(v)* has language elements easy to learn and understand. Additionally, all participants related an increased productivity, i.e. the alarm specification using the language consumes less time that the usual way [39, 72]. Regarding accessibility, some participants consider that using colours in the ports is a major help to understand and remember the language

syntax, reducing the probability of errors and contributing to the language reliability.

With respect to the alarm system validation results, regarding the design most participants gave the highest score. This means that the pleasantness of interaction and likeability of the system interface are extremely good. With respect to functionality most participants gave the highest score. However, there are three participants that did not give the highest score. According them, alarms are not the unique way to obtain information. They also use, for example, graphics about energy consumption and associated costs, giving this a graphical visibility to the information that complements the information given by alarms. All participants gave the highest score to the remaining questions. Thus, we can conclude that the system is easy to use, learn, and produces information in a timely way. In addition to these aspects, users can fulfil their tasks effectively and are satisfied with the outcomes provided.

During the interviews most participants showed great interest and receptivity both in the ASL and in the alarm system. They said that our solution is very innovative and that does not compare with what they have, because they cannot specify alarms or can only specify alarms involving thresholds. Thus, this feedback and the validation results demonstrate that both the ASL and the alarm system are necessary and useful. Additionally, it is also demonstrated that the alarm quality requirements presented in Section 2.3 are important to the development of both the ASL and the alarm system. Using them it is possible to provide the right information to users and to develop language elements (like the Repeat and During elements) to solve some problems regarding the number of active alarms. Finally, considering that the ASL has a good domain coverage, we can conclude that the existing stream query languages (provided by DSMSs) have enough power to express complex conditions regarding alarms. In addition to this, using a DSMS the alarm system is able to provide information in a timely way to users. Indeed, this performance was highlighted by the validation participants. Therefore, we consider that alarm systems can have a new architecture based on a DSMS.

6. CONCLUSIONS

Timeliness is an increasingly important requirement for decision makers since agility in business is critical to success [79]. This is specially true for energy management because faster decisions enable building energy managers to diagnose problems and react quickly to anomalous situations [1]. Energy management systems (EMSs) are used to support the decision making process of building energy managers [6, 80]. Considering the timeliness requirement one important feature of EMSs are alarms because they enable the timely recognition of anomalies [10, 11, 12].

After reviewing the state of the art we concluded that existing EMSs in buildings with alarm functionalities, as well as monitoring systems in general, generate many alarms, overloading users with redundant information hindering their capacity to correctly handle their tasks. Moreover, to the best of our knowledge, most systems do not allow users to create new alarms based on complex conditions without modifying the application code. Finally, that monitoring applications capable of processing real-time data are already taking advantage of DSMSs as an alternative to DBMSs, because the processing rate required is higher than the supported by a traditional DBMS.

Building upon a DSMS-based architecture, this work creates both a visual DSL and an alarm system to support building energy managers to specify domain specific alarms involving complex conditions. This solution enable users to create new alarms just using the DSL on the system interface without the need to modify the application code. Using this solution, building energy managers are able to get enhanced control regarding existing alarms and conditions triggering them.

To validate the proposed solution we validated both the DSL and the alarm system developed. First, the ASL was validated with experienced building energy managers using a DSL validation methodology based on the literature, assessing the language effectiveness, efficiency, satisfaction, accessibility, reliability, and productivity. Second, the alarm system was assessed through its usability and performance characteristics. The results show that the DSL is easy to use and has a good domain coverage, being the system easy to use and efficient in the production of information.

References

[1] L. Copin, H. Rey, X. Vasques, A. Laurent, M. Teisseire, Intelligent Energy Data Warehouse: What Challenges?, in: 2010 22nd IEEE International Conference on Tools with Artificial Intelligence, IEEE, 2010, pp. 337–342.

[2] H. Yang, S. Fong, The Impacts of Data Stream Mining on Real-Time Business Intelligence, in: 2nd international Conference on IT & Business Intelligence (ITBI-10), pp. p. 9–19.

[3] T. M. Nguyen, A. M. Tjoa, Zero-latency Data Warehousing (ZLDWH): the State-of-the-art and experimental implementation approaches, in: 2006 International Conference onResearch, Innovation and Vision for the Future, IEEE, 2006, pp. 167–176.

[4] J. Granderson, M. Piette, B. Rosenblum, L. Hu, Energy Information Handbook: Applications for Energy-Efficient Building Operations. Lawrence Berkeley National Laboratory, LBNL-5272E., 2011.

[5] B. Capehart, Information technology for energy managers, Fairmont Press, 2004.

[6] J. Granderson, M. A. Piette, G. Ghatikar, P. Price, Building Energy Information Systems: State of Technology and User Case Studies. Lawrence Berkeley National Laboratory. LBNL-2899E., in: B. L. Capehart, T. Middelkoop (Eds.), Handbook of Web Based Energy Information and Control Systems, Fairmont Press, first edition, 2011, pp. 133–182.

[7] H. Janetzko, F. Stoffel, S. Mittelstädt, D. A. Keim, Anomaly detection for visual analytics of power consumption data, Computers & Graphics 38 (2014).

[8] V. Chandola, A. Banerjee, V. Kumar, Anomaly Detection: A Survey, ACM Computing Surveys 41 (2009).

[9] N. Meratnia, P. Havinga, Outlier Detection Techniques for Wireless Sensor Networks: A Survey, IEEE Communications Surveys & Tutorials 12 (2010) 159–170.

[10] H. Sizu, Z. Xianfei, Alarms Association Rules Based on Sequential Pattern Mining Algorithm, in: Fifth International Conference on Fuzzy Systems and Knowledge Discovery, IEEE, 2008, pp. 556–560.

[11] S. R. Kondaveeti, I. Izadi, S. L. Shah, T. Black,

T. Chen, Graphical tools for routine assessment of industrial alarm systems, Computers & Chemical Engineering 46 (2012) 39–47.

[12] ANSI/ISA 18.2, ANSI/ISA - 18.2 - 2009 Management of Alarm Systems for the Process Industries, 2009.

[13] G. Cugola, A. Margara, Processing Flows of Information: From Data Stream to Complex Event Processing, ACM Computing Surveys (CSUR) 44 (2012) 1–69.

[14] M. Stonebraker, U. Çetintemel, S. Zdonik, The 8 requirements of real-time stream processing, ACM SIGMOD Record 34 (2005) 42–47.

[15] B. Babcock, B. Shivnath, M. Datar, R. Motwani, J. Widom, Models and Issues in Data Stream Systems, in: Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of database systems, ACM Press, New York, New York, USA, 2002, pp. 1–16.

[16] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik, Aurora: a new model and architecture for data stream management, The VLDB Journal The International Journal on Very Large Data Bases 12 (2003) 120–139.

[17] D. Carney, U. Çetintemel, Monitoring streams: a new class of data management applications, in: 28th VLDB Conference, ACM, 2002, pp. 1–12.

[18] J. Gama, P. P. Rodrigues, Data Stream Processing, in: J. Gama, M. M. Gaber (Eds.), Learning from Data Streams, Springer, 2007, pp. 25–38.

[19] S. Chakravarthy, Q. Jiang, Stream Data Processing: A Quality of Service Perspective, Springer, 2009.

[20] B. Chandramouli, M. Ali, J. Goldstein, B. Sezgin, B. S. Raman, Data stream management systems for computational finance, Computer 43 (2010) 45–52.

[21] X. Jiang, S. Yoo, J. Choi, DSMS in Ubiquitous-Healthcare: A Borealis-based Heart Rate Variability Monitor, in: 4th International Conference on Biomedical Engineering and Informatics (BMEI), IEEE, 2011, pp. 2144–2147.

[22] Q. Zhang, C. Pang, S. Mcbride, D. Hansen, C. Cheung, M. Steyn, Towards Health Data Stream Analytics, in: The 2010 IEEE/ICME International Conference on Complex Medical Engineering, IEEE, 2010, pp. 282–287.

[23] D. Savage, X. Zhang, X. Yu, P. Chou, Q. Wang, Anomaly detection in online social networks, Social Networks 39 (2014) 62–70.

[24] N. Akhtar, F. Siddiqui, UDP Packet Monitoring with Stanford Data Stream Manager, in: International Conference on Recent Trends in Information Technology, IEEE, 2011, pp. 533–537.

[25] C. Cranor, J. Theodore, O. Spataschek, V. Shkapenyuk, Gigascope: A Stream Database for Network Applications, in: ACM SIGMOD International Conference on Management of data, ACM, 2003, pp. 647–651.

[26] M. Sullivan, A. Heybey, Tribeca : A System for Managing Large Databases of Network Traffic, in: USENIX Annual Technical Conference, 98.

[27] S. Zschaler, D. S. Kolovos, N. Drivalos, R. F. Paige, A. Rashid, Domain-Specific Metamodelling Languages for Software Language Engineering, volume 5969, Springer Berlin Heidelberg, 2010.

[28] E. Visser, WebDSL: A Case Study in Domain-Specific Language Engineering, in: R. Lämmel, J. Visser, J. Saraiva (Eds.), Generative and Transformational Techniques in Software Engineering II, Springer Berlin Heidelberg, 2008, pp. 291–373.

[29] P. Klint, A. V. Deursen, P. Klint, J. Visser, Domain-Specific Languages : An Annotated Bibliography, SIGPLAN Not. 35 (2000) 26–36.

[30] M. Mernik, J. Heering, A. Sloane, When and how to develop domain-specific languages, ACM computing surveys (CSUR) 37 (2005) 316–344.

[31] I. Sommerville, Software Engineering, Pearson-Wesley, ninth edition, 2011.

[32] L. Golab, M. T. Ozsu, Issues in Data Stream Management, ACM SIGMOD Record 32 (2003) 5–14.

[33] A. Kleppe, Software Language Engineering: Creating Domain-Specific Languages Using Metamodels, Addison-Wesley Professional, 2008.

[34] A. Barišić, V. Amaral, M. Goulao, Usability Evaluation of Domain-Specific Languages, in: Eighth International Conference on the Quality of Information and Communications Technology, IEEE, 2012, pp. 342–347.

[35] A. Barišić, V. Amaral, M. Goulão, B. Barroca, How to reach a usable DSL? Moving toward a Systematic Evaluation, Electronic Communications of the EASST 50 (2011).

[36] M. Voelter, S. Benz, C. Dietrich, B. Engelmann, M. Helander, L. Kats, E. Visser, G. Wachsmuth, DSL Engineering Designing, Implementing and Using Domain-Specific Languages, CreateSpace Independent Publishing Platform, 2013.

[37] P. J. Clemente, J. M. Conejero, J. Hernández, L. Sánchez, HAAIS-DSL: DSL to Develop Home Automation and Ambient Intelligence Systems, in: Second Workshop on Isolation and Integration in Embedded Systems, ACM, 2009, pp. 13–18.

[38] A. van Deursen, P. Klint, Domain-Specific Language Design Requires Feature Descriptions, Journal of Computing and Information Technology 10 (2002) 1–17.

[39] F. Hermans, M. Pinzger, A. van Deursen, Domain-specific languages in practice: A user study on the success factors, volume 5795, Springer Berlin Heidelberg, 2009.

[40] EEMUA 191, EEMUA Publication 191 Alarm Systems - A Guide to Design, Management and Procurement, Engineering Equipment and Materials Users Association (EEMUA), third edition, 2013.

[41] T. Stauffer, Preventing Unplanned Downtime through Alarm Management: The Five Most Important Alarm Management Techniques for Preventing Unplanned Downtime, Technical Report, Siemens Energy & Automation, Inc., 2006.

[42] L. D. Wilde, D. V. Reising, Where Technology Shapes Solutions - Alarm management: Wasn't that problem already solved years ago?, Technical Report, Honeywel and Abnormal Situation Management (ASM) Consortium, 2011.

[43] G. Lambert-Torres, E. Fonseca, M. Coutinho, R. Rossi, Intelligent Alarm Processing, in: 2006 International Conference on Power System Technology, IEEE, 2006.

[44] J. Cook, A. Meier, Coordinating Fault Detection , Alarm Management , and Energy Efficiency in a Large Corporate Campus, Technical Report, ACEEE Summer Study on Energy Efficiency in Buildings, 2012.

[45] M. Schleburg, L. Christiansen, N. F. Thornhill, A. Fay, A combined analysis of plant connectivity and alarm logs to reduce the number of alerts in an automation system, Journal of Process Control 23 (2013) 839–851.

[46] K. Bhaskar, V. V. S. Sarma, D. Thukaram, K. Ramakrishna, Decision support requirements for intelligent operation of southern grid in India, in: IEEE Power India Conference, IEEE, New Delhi, 2006, pp. 635–641.

[47] F. Dahlstrand, Consequence analysis theory for alarm analysis, Knowledge-Based Systems 15 (2002) 27–36.

[48] M. A. Hossain, P. K. Atrey, A. E. Saddik, Modeling and Assessing Quality of Information in Multisensor Multimedia Monitoring Systems, ACM Transactions on Multimedia Computing, Communications, and Applications 7 (2011) 1–30.

[49] M. Kezunovic, Y. Guan, Intelligent alarm processing: from data intensive to information rich, in: 42nd Hawaii International Conference on System Sciences, IEEE, 2009, pp. 1–8.

[50] I. Izadi, S. L. Shah, T. Chen, Effective resource utilization for Alarm Management, in: 49th IEEE Conference on Decision and Control (CDC), IEEE, 2010, pp. 6803–6808.

[51] F. Yang, S. L. Shah, D. Xiao, T. Chen, Improved correlation analysis and visualization of industrial alarm data., ISA transactions 51 (2012) 499–506.

[52] A. King, A. Roederer, D. Arney, S. Chen, M. Fortinomullen, C. W. H. Iii, V. Kern, N. Stevens, J. Tannen, A. V. Trevino, S. Park, O. Sokolsky, I. Lee, GSA : A Framework for Rapid Prototyping of Smart Alarm Systems, in: 1st ACM International Health Informatics Symposium, ACM, 2010, pp. 487–491.

[53] P. M. Dherte, M. P. G. Negrão, S. Mori Neto, R. Holzhacker, V. Shimada, P. Taberner, M. J. C. Carmona, Smart Alerts: Development of Software to Optimize Data Monitoring, Brazilian Journal of Anesthesiology 61 (2011) 72–80.

[54] N. Stevens, A. R. Giannareas, V. Kern, A. V. Trevino, M. Fortino-Mullen, A. King, I. Lee, Smart Alarms : Multivariate Medical Alarm Integration for Post CABG Surgery Patients, in: 2nd ACM SIGHIT International Health Informatics Symposium, ACM, 2012, pp. 533–542.

[55] D. J. Young, J. R. McDonald, Alarm Processing, in: Intelligent knowledge based systems in electrical power engineering, Springer US, 1997, pp. 119–154.

[56] E. Bristol, Improved process control alarm operation, ISA Transactions 40 (2001) 191–205.

[57] P. T. Bullemer, D. V. C. Reising, M. Tolsma, J. C. Laberge, Towards Improving Operator Alarm Flood Responses: Alternative Alarm Presentation Techniques, Technical Report, Abnormal Situation Management (ASM) Consortium, 2011.

[58] S. Katipamula, M. Brambley, Review Article: Methods for Fault Detection, Diagnostics, and Prognostics for Building Systems - A Review, Part I, HVAC&R Research 11 (2005) 3–25.

[59] McKinstry, Enterprise Energy Management Suite (EEM SuiteTM), 2015.

[60] I. D. Systems, EnergyWitness, 2015.

[61] Enerwise, Energy Manager 3.0, 2015.

[62] B. E. I. S. Tools, P. Monitoring, 2012-2014 OpenEIS (Energy Information System), 2015.

[63] ESightEnergy, Energy Management Software & Services, 2015.

[64] Cylon, Cylon Active Energy Manager, 2015.

[65] Noveda, Noveda EnergyFlow Monitor, 2015.

[66] LMIT Innovation & Technology, Wisemetering, 2015.

[67] Gridpoint, Gridpoint EMS, 2015.

[68] C. C. Aggarwal, Classification in Streams, in: M. T. LIU, LING ÖZSU (Ed.), Encyclopedia of Database Systems, Springer US, 2009, pp. 340–341.

[69] S. Papadimitriou, Anomaly Detection on Streams, in: L. LIU, M. T. ÖZSU (Eds.), Encyclopedia of Database Systems, Springer US, 2009, pp. 88–90.

[70] T. Kosar, P. E. Martínez López, P. a. Barrientos, M. Mernik, A preliminary study on various implementation approaches of domain-specific language, Information and Software Technology 50 (2008) 390–405.

[71] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

[72] G. Kahraman, S. Bilgen, A framework for qualitative assessment of domain-specific languages, Software & Systems Modeling 14 (2015) 1505–1526.

[73] A. Barišić, V. Amaral, M. Goulão, B. Barroca, M. Goulao, Quality in use of domain-specific languages: a case study, in: 3rd ACM SIGPLAN Workshop on Evaluation and usability of programming languages and tools, ACM New York, NY, USA, 2011, pp. 65–72.

[74] ISO 9241-11, ISO 9241-11:1998 Ergonomic requirements for office work with visual display terminals – Part 11: Guidance on usability, 1998.

[75] ISO/IEC 25010, ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, 2011.

[76] Y. Wu, F. Hernandez, F. Ortega, P. J. Clarke, R. France, Measuring the effort for creating and using domain-specific models, in: 10th Workshop on Domain-Specific Modeling, 14, ACM, 2010.

[77] A. Barišić, V. Amaral, M. Goulão, B. Barroca, Evaluating the Usability of Domain-Specific Languages, in: M. Mernik (Ed.), Formal and Practical Aspects of Domain-Specific Languages: Recent Developments, Information Science Reference, 2012.

[78] D. Kolovos, R. Paige, Requirements for domain-specific languages, in: First ECOOP Workshop on Domain-Specific Program Development, pp. 1–4.

[79] Z. Panian, Just-in-Time Business Intelligence and Real-Time Decisioning, International Journal of Applied Mathematics and Informatics 1 (2007) 28–35.

[80] N. Motegi, M. Piette, Case Studies of Energy Information Systems and Related Technology: Operational Practices, Costs, and Benefits, in: Third International Conference for Enhanced Building Operations, Berkeley, California, pp. 1–10.