Supporting Context-Aware Engineering based on Stream Reasoning

Dean Kramer and Juan Carlos Augusto

R.G on Development of Intelligent Environments
Department of Computer Science
Middlesex University London
`j.augusto@mdx.ac.uk`

Abstract. In a world of increasing dynamism, context-awareness gives promise through the ability to detect changes in the context of devices, environment, and people. Equally, with stream reasoning using languages including C-SPARQL, continuous streams of raw data in RDF can be reasoned over for context-awareness. Writing many context queries and rules this way can however be error prone, and often contains boilerplate. In this paper, we present a context modelling notation designed to support the creation of context-awareness based on stream reasoning systems. In validating our language there is tool support which, amongst other benefits, can generate context queries in C-SPARQL and context aggregation rules for higher level context knowledge processing. An Android-compatible mobile platform context reasoner was developed which can handle these deployable context rules. This methodology and associated tools has been validated as part of an EU funded project.

Keywords: context-awareness, stream reasoning, context modelling

1 Introduction

Context-awareness is becoming an increasingly core feature of mobile services. Most services so far make use one single atomic context, for example if location services are enabled in a mobile phone an app can suggest eating places nearby. However, contexts of interest can involve a wide range of contextual features e.g. temperature, luminosity, spatio-temporal coordinates, type of activity a user is performing, level of surrounding noise, and others. In combination, different situations of interest for providing useful service can be identified. These services can be created as a group of rules where we specify that given certain contextual conditions are satisfied we want the system to reach in a particular way. However the accumulation of rules and the aggregation of more than one rule to create higher level rules can easily lead to hidden indirect interactions within that set of rules which may go unnoticed to developers and be the cause of undesirable system behaviour.

Hence developers in the area have rightly resorted to different ways to understand contexts and their inter-relationships. For example, the use of ontologies for context-aware systems has been a popular approach for modelling context, and for context reasoning. These ontologies are often modelled using the Web Ontology Language (OWL), which allows for an explicit specification of the different contextual information in the system. The drawbacks to using static queries over ontologies that change frequently, and unsuccessful practical consideration of temporal features have been discussed by Valle et. al [24]. Mobile applications are becoming increasingly based on streams of high frequency real time data, hence there is a need for reasoners to have the ability to reason over temporal elements of the information. As a result, recent developments have started to get increasingly involved with *stream reasoning*. One such stream reasoning language includes C-SPARQL [5]. C-SPARQL extends the SPARQL query language to express and handle continuous queries, and allow for temporal based queries and reasoning through the use of RDF quadruples. By using stream reasoning, RDF tuples from different sensors and services can be queried in real-time.

To generate some form of stream reasoning based on languages like C-SPARQL, different queries often need to be created by hand. Creating these queries however can often be error prone, and also include a lot of boilerplate which could be present in many queries. This is particularly the case when contexts can contain many different states. Understanding a context tree hierarchy can be made more difficult if all rules and queries are based purely in text. In this paper, we propose the use of stream reasoning in mobile context-awareness system. To support developers in using stream reasoning, we propose a new modelling notation developed to assist in model abstraction. This notation can then generate the appropriate queries to be deployable on a mobile context reasoner developed to harness stream reasoning. This contribution is actually very consistent with recent surveys highlighting the lack of methodological support available to the community of context-aware systems developers [1].

The remainder of this paper is structured as following: In Section 2 we introduce a real life scenario which we use to illustrate our contribution. Section 3 presents the state of art regarding context modelling. Then in Section 4, we present our graphical notation used to model context. Next in Section 5, we describe how our context models are transformed to C-SPARQL rules, which are deployable on a running system. In Section 6 we validate our approach, and finally conclude and describe future work in Section 7.

2 Motivation

The research and innovation we report in this paper have been influenced by practical needs and informed by our experience and related research. Our practical needs come from a European project with a varied mix of stakeholders, which include end user organizations and commercial companies from several countries. Each of these add requirements in terms of usability, efficiency, affordability and dependability. Our innovation balances these. To illustrate the type of situations our modelling notation and reasoning software can support, we introduce a context scenario which is extracted from the real life situation we support within the European research project where the method and tool we report in this paper are being used.

As a practical scenario, people with different cognitive disabilities, particularly Down's Syndrome can face challenges integrating with society due to difficulties navigating their environment. Many young individuals with these disabilities often still want to retain or gain independence, much like other young people without cognitive disabilities. One way context-awareness can assist those users is by allowing them to navigate safely, giving them the ability to travel alone, however prompting the user if it perceives a problem. One of these problems can relate to following the route on their mobile device. In terms of deviating a route in mobile navigation systems, often the user either makes a small deviation (requiring a short return to their journey) or make a large deviation (where a completely new route needs calculating). To give people with cognitive disabilities independence, yet provide support when in need, we could suggest that those users should get support (asked if they wish to call their carer) if they have made a large deviation, and likely lost. Continuing this, to give some independence, and to prevent frustrating the user, we can allow the user to make a certain number of small mistakes without asking if they need assistance. This can allow them to make some corrections by themselves without interference, but still help them if they continue to make many small route mistakes. We can consider this a navigation assistance context, which can have both states "needed" and "not-needed". These contexts will need to handle temporal elements, as several of the opportunities for assistance will relate to helping them achieve goals at specific times (e.g., reaching school by 8AM or being reminded to pack Gym clothing on Wednesday), time can also be an indicator of certain desirable/undesirable situations (e.g., standing still for a long time in the way to a place, taking too long to reach a place, or being off the expected route for too long).

3 Related Work

In order for a system to be context-aware it must deal with a number of important issues, some of them are internal (system own awareness of current internal status and capabilities) and some are external and relates to the real world (for example, place, time, user preferences and needs). To achieve these, a context-aware system needs to model the context information. Context models and languages have predominantly been either *key-value models, markup scheme models, graphical models, object-oriented models, logic based models,* or *ontology based models* [23].

Key-value models are the simplest form of context models, involving a name and context value pairs, which have been used directly (without the use of AI techniques applied) in context-aware systems and frameworks [17]. Markup models are hierarchical data structures which consist of markup tags, attributes, and content, with an example being the Comprehensive Structured Context Profiles (CSCP) [11]. CSCP is based in RDF, and expresses context information using service profiles, describing context information relevant to different sessions. Graphical models can use different graphical notations including UML, Object-Role Modelling (ORM), and other domain specific modelling languages. ORM based context models include the Context Modelling Language (CML) [12]. Included in CML are constructs for describing information types, type classifications, metadata for quality, and type dependencies. UML based models include ContextUML [21], and the MUSIC context model [19]. These models can also be viewed as object-oriented models, as they use different object-orientation concepts including inheritance, and encapsulation. Other domain specific languages (DSL) for context modelling include PervML [18, 20] and MLContext [13]. Different logic based modelling languages also exist, including the Calculus of Context-Aware Ambients (CCA) [22], CONAWA [15], SCAFOS [14], and an algebra of contextualised ontologies [7]. The CCA proposes a logical language for expressing context properties using context expressions. Context expressions can be composed to form complex expressions and formulas using first order operators. The CONAWA calculus was inspired by the ambient calculus [8], and extends it in a number of ways including syntax extensions, and aims for uniform representation of entities & context.

All these approaches are useful to some extent, however they have not been adopted extensively so far and part of the reason is in their complexity, either conceptually or in terms of their computational requirements. The project which triggered our research in this area requires of a process which combines development support, with real-time efficiency and correctness. After extensive literature review and tests in our labs we identified a stream based query processing language which gave us reasonable efficiency in a mobile platform. Programming a long list of inter-related queries however is not the most pleasurable experience and it is also error prone not only at individual rule level but also in terms of understanding the 'big picture' of how the collection of rules relate to each other. Our context rule modelling language described in the next section facilitates that high level decision making of which contexts to consider and how they inter-relate. The support we provide offers the following: we abstract from the specific query language syntactic complications, and secondly force the team to reflect and judge if the information sources needed for an inference are obtainable. Later on in this paper we explain how once this is determined an automatic translation into a target query language can be obtained and how we used this support to help developing a real-life system.

4 Modelling Notation

To alleviate the need for writing all context rules by hand, we introduce a context modelling notation, illustrated in Figure 1. In these models, we have three main context model constructs: *context source, inference rule,* and *context state*.

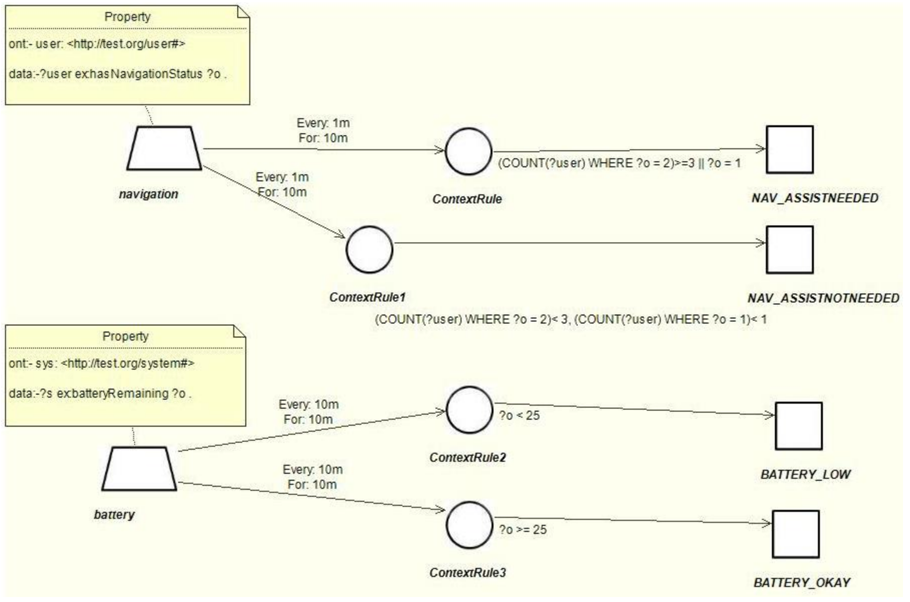

Property
ont:- user: <http://test.org/user#>
data:-?user ex:hasNavigationStatus ?o .
navigation
Every: 1m
For: 10m
ContextRule
(COUNT(?user) WHERE ?o = 2)>=3 || ?o = 1
NAV_ASSISTNEEDED
Every: 1m
For: 10m
ContextRule1
NAV_ASSISTNOTNEEDED
(COUNT(?user) WHERE ?o = 2)< 3, (COUNT(?user) WHERE ?o = 1)< 1
Property
ont:- sys: <http://test.org/system#>
data:-?s ex:batteryRemaining ?o .
battery
Every: 10m
For: 10m
ContextRule2
?o < 25
BATTERY_LOW
Every: 10m
For: 10m
ContextRule3
?o >= 25
BATTERY_OKAY

Fig. 1. An example model using our notation

4.1 Context Source

What separates context-aware systems with regular is the ability to infer the context of the device, user, and environment. For the developer to express rules to carry this out, there needs to be raw data for which can be reasoned over. This raw data can come from a variety of sources include sensors, and other off device services including web services. In the modelling notation, a context source object denotes a particular source of raw context data. This raw data is expected to be received in RDF triples. In the model, context sources are denoted with trapezium objects. Each context source can interact with more than a single inference rule. For each context source, there are different properties that can be added. The first includes an *Information Source*, which is used to define specific SPARQL ontological prefixes. The next property is *Data*, which relates to the RDF triples that context source provides. Just as in SPARQL, we denote variable names with a question mark (?) prefix. These variables can then be used in the Inference rules. A single context source can have a relationship between many different inference rules.

4.2 Inference Rule

Raw data alone is not always ideal for creating context-aware applications. For example, when considering remaining battery life in a mobile device, often people are interested to know when it is low, not if it is 67% charged. Having defined the raw data expected to be received by the device, to create more meaningful atomic context information, this raw data needs to be used in a context inference. This object denotes different inference rules that are used to infer a particular context state. Inference rules in our modelling notation can contain a mix of both simple *logical evaluations*, and more complex *functions*. Logical evaluations in the inference rules allow the developer to create rules based on a particular logical conditions, for example "if battery level is greater than 25%". To use logical expressions, the developer can reference particular variables from the context source. Only variables from a context source related to that particular context rule can be used. For multiple logical expressions, these can be separated with a comma allowing a conjuncture of different expressions.

Function usage within an inference rule are related to the different functions built-in to C-SPARQL/SPARQL. These functions can allow the developer to do far more computations over the raw data to assist inference. An example from our project includes calculating the total sum that a person has walked in a particular interval. In the inference rule, these functions can be used whereby the developer states the function signature, any variables required in the parameter parentheses. If the developer wants a particular set of RDF triples being used with the functions, they can be stated. By default, these functions are called with all RDF triples known the inference rule itself. Lastly, the developer can also apply logical evaluations to the returned result of those functions.

4.3 Context State

Having defined the particular inferences the system needs to make on the raw context data, the developer can use context states to be applied if the inference rule evaluates as expressed. Context states can be used for both single atomic contexts, and higher level aggregation contexts.

4.4 Relationships

In addition to these context constructs, there are relationships that exist between them. These construct relationships include:

– **Source - Rule:** This type of relationships defines what raw data the inference rule queries over. Included in this relationship is temporal operators regarding the raw RDF, including *data window size,* and *execution frequency*. Data window size relates primarily to how much RDF data can be included within the rule, and is noted by `For`. As featured in C-SPARQL, the data window can be either *physical* (a given number of RDF triples) or *logical* (a triples occurring with a given time inteval). Execution frequency relates to how often the inference rule is run in the reasoner, and is noted by `Every`. This frequency corresponds to sliding logical windows [9] in C-SPARQL, and in essence refers to the possibility of RDF triples being present in more than a single logical window. In practice this will equate to how often that query is updated. If the data window size is physical with exact triples being set, this field is not required.

- **Rule - State:** Inference rules themselves do not include the specific context states the rule equates to. This separation allows the developer to connect more than a single rule to a context state, which can encourage reuse.
- **State - State:** So far, we have considered the modelling of lower level, atomic contexts. These contexts deal with singular situations of an element, based on raw data from sensors. When developing context-aware systems, the ability to aggregate, and reason over a collection of different contexts gives the ability to consider higher level knowledge on a situation. For this, we allow the user to aggregate contexts together in our notation. To aggregate context values, we join the different context states together, and add a logical operator (AND, OR, XOR). If no logical operator is given explicitly, the default value is a conjunction.

4.5 Scenario Context Model

Continuing with the motivating scenario context, we model it using our notation, which can be seen in Figure 1. We need two specific states, one when navigation assistance is needed, and one when it is not needed. We can name these states `NAV_ASSISTNEEDED` and `NAV_ASSISTNOTNEEDED`. Next, we model the context source, which we name "navigation" for ease. In this context source, we expect single RDF triples including the user identifier and its navigationStatus. We assign two variables, one to the user identifier `?user` and one to the RDF value `?o`. This will allow us to use them in the inference rules, two of which we need to create. For the rule we intend on binding to the `NAV_ASSISTNEEDED` state, we wish to express that if within the specified time interval, there needs to be 3 or more occurrences where the value in `?o` is 2, or there is ever a value of 1. To find out the number of occurrences where `?o` has the value of 2, we use the `Count` function passing the `?user` variable. The rule intended for use by the `NAV_ASSISTNOTNEEDED` context state is more complex. We need to check that both the number of occurances of `?o` being 2 is less than 3 times, and also check there are no occurrences of `?o` having the value 1. Each of these two `Count` function uses is separated by a comma, defining the rule as a conjunction of the two.

5 Rule Generation

Following the construction of the overall context model, we need to generate the C-SPARQL queries and aggregation rules to be deployed on the mobile device. First, we get a list of all atomic and aggregate contexts in the model based on object relationships in the model. For atomic rules, context states will need to contain relationships with inference rules. In comparison, aggregation rules will purely have relationships with just other context states.

A C-SPARQL query is created for every atomic context. To do this, first the tree of objects that relate to that atomic context is retrieved. This includes inference rules, and context sources that are linked to those inference rules.

Queries names are generated using a concatenation of both the context state and `_query`. Therefore a context state such as `NAV_ASSISTNOTNEEDED` will create a query named `NAV_ASSISTNOTNEEDED_query`. Next, query prefixes are generated, by using `ont` properties of the related context sources objects. Following this, the type of C-SPARQL query is declared, in our case we use `Construct` queries, which return an RDF graph based on the result of the query. Next, we set the temporal operators of the C-SPARQL rules getting the execution frequency and data window size from the source-rule relationship. In the event there are multiple context sources with different temporal settings, we set the frequency to be same as the most frequent using a data window size as the same as the largest. Using these two temporal settings, we use the data window size value for the `Range`, and execution frequency for the `Step` parameters. Next, `Where` clause of the query are generated. Firstly, we copy all the data values from the related context sources in the rule object tree including all declared variables in RDF. Following this, we generate any subqueries that might be required in the context query. Subqueries within our modelling notation are generated only for handling functions within C-SPARQL. This generation is explained in the following subsection. Finally, we generate SPARQL `Filter` clauses for any inference rule logical evaluations. If multiple logical evaluations have been used, each comma separated, a filter is created for each one.

5.1 Handling Functions

As introduced earlier, we allow the use of SPARQL aggregate functions in inference rules. When generating C-SPARQL rules, these are handled through the generation of subqueries. For every function used in the inference rule, a `Select` subquery is generated. These generated subqueries are places within the `Where` clause block in the main context query. In this generated select query, we first get the function name and any passed parameters. Next we have to give the function return variable a name which is added after the `AS` keyword, which for this generate a name `subqres_` with a incremented number. An example can be `COUNT(?user) As ?subqres_1`, which would counts the number of occurrences of the value held in variable `?user`, placing the resulting number in the ?subqres_. Following this, we create the `Where` clause copying the raw data RDF from the context sources, the same as the main query. Finally, where functions usage have a logical evaluations to be used over the resultant variables, we generate `Filter` clauses for each evaluation using the variable generated automatically earlier.

5.2 Aggregate Contexts

As described earlier, we do not use C-SPARQL for context aggregation. We instead, we generate propositional formulas to represent our aggregation rules. These rules are evaluated on the deployed mobile device by means of boolean satisfiability. For satisfiability tests, we generate propositional rules that use

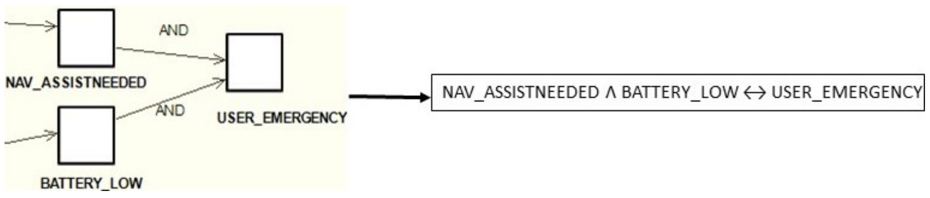


Fig. 2. Aggregation Context Generation

equality as a method of determining if the aggregate context is true. To generate these aggregation rules, we check the model for Context State-to-Context State relationships. These relationships occur when two context states are joined together, which also have a particular logical connective including AND, NOT, OR. In Figure 2, we illustrate a context aggregation, and the aggregation rule that is created.

5.3 Mobile Context Reasoner

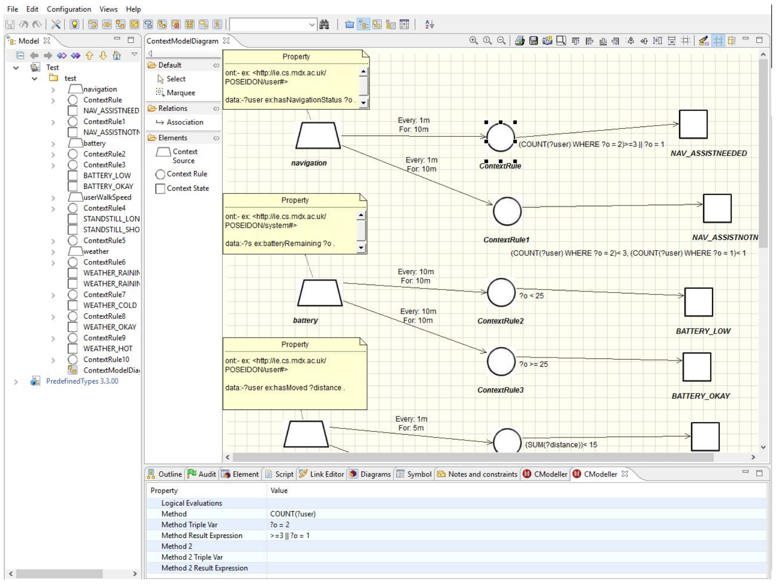
Complementing the modelling notation, generation of the C-SPARQL queries and aggregate contexts, we designed a mobile context reasoner which uses the C-SPARQL queries and aggregations rules for reasoning. This reasoner is designed to be extensible by different applications which want to reasoner over new contexts. This is carried out by separating raw data retrieval and acquisition from reasoning. Each application can add new raw data observers that get data from sensors, device broadcasts, and cloud services, and add new C-SPARQL & aggregation rules to extend the reasoner in new directions and purposes. Further details of the reasoner can be found in [3].

6 Validation

Currently, we are validating this approach in an FP7 European project aimed to assist project with Down's Syndrome (DS). The core aim of this project is to increase the inclusion of such citizens through the assistance of travelling and navigation. This project is using a combination of home learning, through the use of mix reality to assist users in creating and learning routes with their carers', and context-aware navigational services. An output of the project is to also assist other developers to develop context-aware services for people with DS, and other user with cognitive disabilities to assist them with daily tasks. Within POSEIDON, validation of our approach has been carried out both by implementation, and use with real end users.

6.1 CoMo Tool

Firstly, part of the validation is to help support developers in creating context models and rules using the described approach. To validate our modelling notation, tool support was implemented on top of the modelling tool, *Modelio* which we named *CoMo*[1] . Modelio is an opensource modelling environment that supports notations including UML2, and BPNM2. Modelio supports additional notations through the use of extensions which can be added to Modelio projects. Our tool has been implemented as an extension module which can be added by developer end users to their Modelio installations. As described in this paper, we include the ability to generate C-SPARQL queries and aggregation rules using propositional formula. The Long term goal for our tool is to generate context models for multiple languages. To do so, developers can use and extend our `AbstractModelWriter` class to create language printers for different runtime context systems. Using the modelling tool, each of the constructs are drag and drop objects, and includes tabs for inputting the construct properties. When the user is satisfied with their model, they can select to export the model to C-SPARQL queries and our aggregation rules.



ont:- ex: <http://ie.cs.mdx.ac.uk/POSEIDON/user#>

data:-?user ex:hasNavigationStatus ?o .

ont:- ex: <http://ie.cs.mdx.ac.uk/POSEIDON/system#>

data:-?s ex:batteryRemaining ?o .

ont:- ex: <http://ie.cs.mdx.ac.uk/POSEIDON/user#>

data:-?user ex:hasMoved ?distance .

(COUNT(?user) WHERE ?o = 2)>=3 || ?o = 1

(COUNT(?user) WHERE ?o = 2)< 3, (COUNT(?user) WHERE ?o = 1)< 1

?o < 25

?o >= 25

(SUM(?distance))< 15

Property	Value
Logical Evaluations	
Method	COUNT(?user)
Method Triple Var	?o = 2
Method Result Expression	>=3 || ?o = 1
Method 2	
Method 2 Triple Var	
Method 2 Result Expression	

Fig. 3. Context Modeller Modelio Extension

6.2 Android Middleware

To validate the mobile reasoner, we have developed a working prototype on the Android platform. This reasoner has been developed to be deployable in following scenarios:

[1] Source code available at: https://github.com/deankramer/ContextModeller

– **Centralised Application:** The reasoner is a completely separate application which can be installed and used independently. Using this approach, the end user can download the reasoner, which is currently available from the Google Play Store
– **Integrated into an Existing Application:** The reasoner is integrated as a software library into an existing application during compilation.

C-SPARQL has been implemented through the extension of Apache Jena. To include C-SPARQL in Android, we had to make a number of modifications. Firstly C-SPARQL uses Esper, an event processing library by EsperTech, which is not fully compatible with Android. We found an Android compatible version of Esper named Asper, which we used. Secondly, Apache Jena uses the Simple Logging Fascade for Java (SLF4J) library for logging. This as well was changed for the Android equivalent. Other modifications included package renaming in Apache Jena component libraries due to namespace conflicts in the Android SDK. To support the context aggregation rules, we used the Prop4J[2] library, which uses Sat4J to handle propositional formulas.

6.3 Project Validation

Project validation has been carried out using project pilots, and user centred meetings over the course of this project. To assist the users when they are travelling outside, an app has been developed by a project partner to assist with navigation and event planning, which leverages our context-awareness reasoner. In Figure 5, we illustrate some screenshots of this application. This application uses a number of context rules including:

– **Battery**: Informs when the battery is getting close to low, giving the user enough time to charge the phone. This example can be found also in Figure 1.
– **Weather**: Weather conditions for particular locations of interest, used for clothing advice before leaving.
– **Navigation Assistence**: As described in Section 2. Application can check if assistence is needed based on route deviations. One of the generated rules can be seen in Figure 4.
– **Standstill**: Checks if the user has been relatively stationary for the set period of time, useful when waiting for public transport to prevent excessive waiting in poor conditions.

Use in the Field Over the course of this project, a number of opportunities for testing between end-users have taken place. These include single and multi-day user centred meetings, and month long pilots. We have co-operated with people with DS from different countries including UK, Germany, Norway, Luxembourg, Portugal, and Ukraine. For the month long pilots, these were carried out on 3 families in the UK, Germany, and Norway. In these pilots, the primary users

[2] http://spl2go.cs.ovgu.de/projects/1

```
REGISTER QUERY NAV_ASSISTNOTNEEDED_query AS
PREFIX ex: <http://test.org/project/user#>
CONSTRUCT { ex:navigation <http://test.org/context/is> "NAV_ASSISTNOTNEEDED"}
FROM STREAM <http://test.org/context-stream> [RANGE 10m STEP 1m]
WHERE { ?user ex:hasNavigationStatus ?o .
        { SELECT  (COUNT(?user) AS subqres_1)
        WHERE { ?user ex:hasNavigationStatus ?o .
                FILTER( ?o = 2 )
                }
        }
        { SELECT  (COUNT(?user) AS subqres_2)
        WHERE { ?user ex:hasNavigationStatus ?o .
                FILTER( ?o = 1 )
                }
        }
        FILTER ( subqres_2 < 1 && subqres_1 < 3  )  }
```

Fig. 4. Generated Navigation Assistance Rule

and their carers' were given a smart phone with the context-aware services and navigation application, and mixed reality application to practice their navigation skills. Other works which looked at the effectiveness of services developed on the project includes Kramer et. al [16]. This work found important issues which need to be considered when developing navigational services for people with DS, including safety and directional assistance. For a more detailed description of the validation activities, the reader is recommended to read [2].

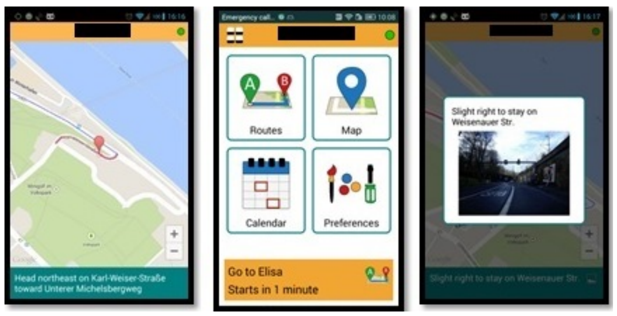

[redacted]
Head northeast on Karl-Weiser-Straße toward Unterer Michelsbergweg
[redacted]
Go to Elisa
Starts in 1 minute
[redacted]
Slight right to stay on Weisenauer Str.

Fig. 5. End User Navigation and Calendar Application

7 Conclusion and Future Work

We presented a new context modelling notation that can leverage the use of stream reasoning for context-awareness, with a mobile reasoner that leverages stream reasoning. Using our modelling notation, developers can handle context-awareness from RDF streams, originated from sensors, and other internet services. This modelling notation reduces boilerplate, and facilitates reuse of different elements of the context rule. To facilitate practical benefits of the new methodology we have implemented both a tool for context modelling. Methodology and tool support have been validated by companies developing an FP7 EU project which assists people with Down's Syndrome in various daily life practical situations. The context reasoner using C-SPARQL rules has been used over months of testing by people with Down's Syndrome and their carers'. Although a visual design tool helps thinking of the overall contextual logic of the system it does not guarantee correctness. Hidden combinations may lead to undesirable system behaviour. The need and benefits of using some of the available tools created in Software Engineering to create products which users can trust more have been already proposed in the past [4]. Consequently, our current work is focused on using model checkers, for example UPPAAL [6] for rule verification. UPPAAL is a toolkit for verification of real-time systems, and has been used before for correctness analysis in smart homes [10]. We are investigating the use of model checkers as an approach to increase the correctness of models created in our notation.

References

1. Alegre, U., Augusto, J.C., Clark, T.: Engineering Context-Aware Systems and Applications: A survey. Journal of Systems and Software 117(55-83) (2016)
2. Augusto, J., Kramer, D., Alegre, U., Covaci, A., Santokhee, A.: The User-centred Intelligent Environments Development Process as a Guide to Co-create Smart Technology for People with Special Needs. Universal Access in the Information Society (January 2017)
3. Augusto, J., Kramer, D.: Poseidon deliverable d3.2 : Reasoning and learning module. Tech. rep., POSEIDON Project (2015)
4. Augusto, J.C.: Increasing Reliability in the Development of Intelligent Environments. In: 5th International Conference on Intelligent Environments (IE-09). pp. 20–21 (July 2009)
5. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: C-SPARQL: A Continuous Query Language for RDF Data Streams. International Journal of Semantic Computing 04(01), 3–25 (Mar 2010)
6. Behrmann, G., David, A., Larsen, K.G., Hakansson, J., Petterson, P., Wang, Y., Hendriks, M.: Uppaal 4.0. In: Third International Conference on the Quantitative Evaluation of Systems, QEST 2006. pp. 125–126 (2006)
7. Cafezeiro, I., Viterbo, J., Rademaker, Alexandre Haeusler Edward Hermann Endler, M.: A Formal Framework for Modeling Context-Aware Behavior in Ubiquitous Computing. In: Leveraging Applications of Formal Methods, Verification and Validation, vol. 17, pp. 519–533 (2008)
8. Cardelli, L.: Mobility and Security. In: Lecture Notes for the Marktoberdorf Summer School 1999 (1999)
9. Golab, L., Ozsu, M.T.: Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams. In: VLDB. pp. 500–511. No. February (2003)
10. Guilly, T.L., Smedegard, J.H., Pedersen, T., Skou, A.: To Do and Not to Do: Constrained Scenarios for Safe Smart House. In: 2015 International Conference on Intelligent Environments. pp. 17–24. IEEE (jul 2015)
11. Held, A., Buchholz, S., Schill, A.: Modeling of Context Information for Pervasive Computing Applications. In: Proc. of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI2002). p. 6 pp. (2002)

12. Henricksen, K., Indulska, J.: Developing context-aware pervasive computing applications: Models and approach. Pervasive Mob. Comput. 2(1), 37–64 (Feb 2006)
13. Hoyos, J.R., García-Molina, J., Botía, J.A.: A domain-specific language for context modeling in context-aware systems. Journal of Systems and Software 86(11), 2890–2905 (2013)
14. Katsiri, E., Seranno, J.M., Serrat, J.: Application of Logic Models for Pervasive Computing Environments and Context-Aware Services Support. In: Multimedia Services in Intelligent Environments, vol. 2, pp. 105–117 (2010)
15. Kjæ rgaard, M.B., Bunde-pedersen, J.: Towards a Formal Model of Context Awareness. In: Proceedings of the International Workshop of Combining Theory and Systems Building in Pervasive Computing-Pervasive 2006. pp. 667–674 (2006)
16. Kramer, D., Covaci, A., Augusto, J.C.: Developing Navigational Services for People with Down's Syndrome. In: 2015 International Conference on Intelligent Environments. pp. 128–131 (2015)
17. Kramer, D., Kocurova, A., Oussena, S., Clark, T., Komisarczuk, P.: An extensible, self contained, layered approach to context acquisition. In: Proceedings of the Third International Workshop on Middleware for Pervasive Mobile and Embedded Computing - M-MPAC '11. pp. 1–7. ACM Press, New York, New York, USA (Dec 2011)
18. Muñoz, J., Pelechano, V., Fons, J.: Model Driven Development of Pervasive Systems. In: Proceedings of the 1st International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES). pp. 2–14 (2004)
19. Reichle, R., Wagner, M., Khan, M.U., Geihs, K., Lorenzo, J., Valla, M., Fra, C., Paspallis, N., Papadopoulos, G.A.: A Comprehensive Context Modeling Framework for Pervasive Computing Systems. In: Distributed Applications and Interoperable Systems, Lecture Notes in Computer Science, vol. 5053, pp. 281–295. Springer Berlin Heidelberg (2008)
20. Serral, E., Valderas, P., Muñoz, J., Pelechano, V.: Towards a Model Driven Development of Context-aware Systems for AmI Environments. In: Developing Ambient Intelligence, pp. 113–124. Springer Paris (2008)
21. Sheng, Q.Z., Benatallah, B.: ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services Development. In: Proceedings of the International Conference on Mobile Business. pp. 206–212. IEEE Computer Society, Washington, DC, USA (2005)
22. Siewe, F., Zedan, H., Cau, A.: The Calculus of Context-aware Ambients. Journal of Computer and System Sciences 77(4), 597–620 (Jul 2011)
23. Strang, T., Linnhoff-Popien, C.: A Context Modeling Survey. In: Proceedings of the Workshop on Advanced Context Modelling, Reasoning and Management. vol. Workshop o, pp. 1–8 (2004)
24. Valle, E.D., Ceri, S., van Harmelen, F., Fensel, D.: It's a Streaming World! Reasoning upon Rapidly Changing Information. IEEE Intelligent Systems 24(6), 83–89 (Nov 2009)