A Methodological Approach to Quality of Future Context for Proactive Smart Systems

Yves Vanrompay[1] and Yolande Berbers[2]

[1] MAS Laboratory, Ecole Centrale Paris
Grande Voie des Vignes, F-92 295 Chatenay-Malabry, France
yves.vanrompay@ecp.fr
[2] Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, 3001 Heverlee, Belgium
yolande.berbers@cs.kuleuven.be

Abstract. Many current context-aware systems only react to the current situation and context changes as the occur. In order to anticipate to future situations and exhibit proactive behavior, these systems should also be aware of their future context. Since predicted context is uncertain and can be wrong, applications need to be able to assess the quality of the predicted context information. This allows applications to make a well-informed decision whether to act on the prediction or not. In this paper, we present prediction quality metrics to evaluate the probability of future situations. These metrics are integrated in a structured prediction component development methodology, which is illustrated by a health care application scenario. The metrics and the methodology address the needs of the developer aiming to build context-aware applications that realize proactive behavior with regard to past, present and future context.

Key words: context prediction, quality of context, smart systems

1 Introduction

In pervasive scenarios foreseen by ubiquitous computing, context awareness plays a central role. Context-aware systems are able to adapt themselves to their environment aiming at optimizing QoS for the user and resource consumption by taking into account and reasoning with context information. Many systems have been developed that are aware of the current context. However, these systems only react to changes in context information as they occur. As such they only take into account present (and possibly past) context information to act on. In many situations this is not sufficient since by only looking at the momentarily information it can already be too late to take appropriate actions and keep QoS at an acceptable level. Therefore applications should also take into account future context information. For example, instead of simply detecting a memory shortage as it occurs, a system should be able to predict the memory shortage in order to prevent it.

Instead of purely reactive context-aware systems, proactive context-aware systems are required that use past, present *and* future context information.

Future context information is predicted by learning of patterns and relations that are detected by analyzing the context history. We will argue for the importance of the integration of domain and prior knowledge to make the prediction process more efficient and transparent.

Proactive context-aware systems take actions based on predicted context information. The decision whether to act or not depends heavily on the quality of the predicted information. Since predictions are naturally uncertain and can be wrong, well-informed decision making taking into account the quality of the information is needed. While the cost of acting based on a wrong prediction can be limited to user annoyance or wasted system resources, more damage can be caused in the area of safety-critical systems. Therefore we propose a set of quality of future context (QoFC) metrics which allows the application to form a view on whether the prediction can be trusted enough to effectively be useable. As ubiquitous systems should be as unobtrusive for the user as possible, explicit user feedback should be kept to a minimum in the decision making process.

To support the developer in realizing *future context*-aware applications, we integrate the metrics in an overall structured methodology for developing concrete prediction components. All together, we aim to provide an efficient and transparent approach to context prediction. Efficiency is achieved by supporting the developer with a clear methodology. Transparency means that it is clear for the application what the quality is of a particular predicted context.

This paper is organized as follows. Section 2 introduces the concept of quality of future context together with a taxonomy of concrete quality metrics. We explain how these metrics can be concretely used by context-aware applications. Section 3 presents a step-by-step prediction component development methodology, illustrated by a case study from the health care domain. Section 4 gives an overview of related work, after which we draw conclusions and discuss future research directions.

2 Context prediction quality metrics

Practical applications should not rely on context prediction results per se. The predicted value might be incorrect for several reasons. The given input might be erroneous or there might be several alternatives for the chosen value. As a consequence, the prediction results will not be accurate in all cases. Applications relying on these results should be able to assess the trust they can have that the prediction is correct. This allows the applications to decide whether they want to use the result, even if the probability of being incorrect is high.

In the following we define a set of Quality of Future Context (QoFC) metrics and explain how these metrics can assist an application in making decisions and take appropriate actions based on the quality of the predicted context information. We consider that QoFC is an objective notion when it is provided by a prediction component to an application. However, when it is effectively used by a specific application to determine its worth, it becomes subjective.

Quality of Context (QoC) is any inherent information that describes context information and can be used to determine the worth of the information for a specific application. This includes information about the provisioning process the information has undergone (history, age), but not estimations about future provisioning steps it might run through. [6]

The above QoC definition can equally be applied to QoFC. As such, QoFC metrics are objective metrics about the quality of the provided *future context* information. Applications can prioritize and weigh several QoFC metrics according to their specific requirements, turning these metrics into subjective ones.

2.1 Quality of Future Context Metrics

We propose a taxonomy of QoFC metrics used to assess the confidence one can have in the quality of both predictors in general and concrete predicted context values. As can be seen in Figure 1, QoFC metrics are divided into three categories.

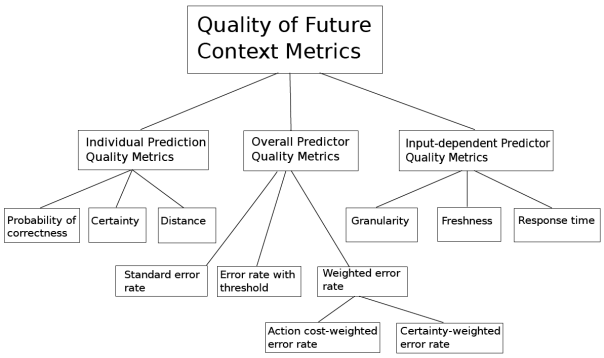

Fig. 1. Quality of Future Context metrics taxonomy

Individual prediction quality metrics: Individual prediction quality metrics assess the confidence an application can have in individual, concrete context predictions of a predictor. For example, the probability of a predicted location tells something about the confidence one can have in that individual prediction, but says nothing about the general accuracy of the predictor over a whole dataset.

- *Probability of correctness P:* The probability of the most likely prediction, i.e. the prediction with the highest probability. In statistics, this corresponds to the confidence one has in a classification.
- *Certainty C:* This metric does not only take into account the probability of correctness P, but also m, the number of possible outcomes, i.e. predictions

with a probability higher than zero. It is defined as follows: $c = \frac{p-\frac{1}{m}}{1-\frac{1}{m}}$. For an extensive evaluation of this metric, we refer to previous work [1].
- *Distance metric D:* Suppose P_a is the probability of the most likely predicted value, and P_b is the probability of the second most likely predicted value. Then D is the difference between P_a and P_b: $P_a - P_b$.

Overall predictor quality metrics: This second category of QoFC metrics, the overall predictor quality metrics, gives information about the overall accuracy of a given predictor. Several variants of the error rate (i.e. the number of erroneous predictions relative to the total number of predictions over a dataset) belong to this category.

- *Standard error rate E_s:* the number of incorrect predictions relative to the total number of predictions.
- *Error rate with threshold E_t:* the number of incorrect predictions relative to the number of total predictions, taking into account only the predictions with a probability of correctness above a threshold.
- *Certainty-weighted error rate E_c:* This error rate penalizes predictions with a high associated certainty c and which are wrong compared to those which have a low certainty c and are wrong by using the certainty values as weights in the error rate formula.
- *Action cost-weighted error rate E_{ac}:* This error rate penalizes predictions which are wrong and for which the actions taken based on the prediction are costly. A 3-dimensional cost matrix is associated with this error rate, expressing the cost of performing action a instead of the correct action b, given the future context.

Input-dependent predictor quality metrics: As a third category, we group quality metrics that can depend on the input needed by the predictor itself. For example, the freshness of the predicted context value can depend on the freshness of the data that is provided to the predictor to make its prediction. Also, the time needed for making a prediction can depend on the time that sensors need to provide the required information to the predictor, in addition to the time necessary to calculate the prediction itself.

- *Granularity G:* Granularity gives an indication on how fine-grained the predicted value is. E.g. an algorithm can be able to predict a user's location onto the street address level or the city level. The granularity of the predicted result can depend on the capabilities of the prediction algorithm itself, but also on the granularity of the input value(s) given to the algorithm, and thus on the kind of sensors that are available.
- *Freshness F:* Freshness gives an indication of the age of the predicted context information. The freshness can depend on the properties of the prediction algorithm, but more critical is the freshness of the context information on the basis of which the prediction is made. E.g. if all required input to make a prediction is only available once each hour, this will naturally influence the freshness of the prediction itself.

– *Response time T_r:* This is the time between the predictor receiving a prediction request and the predictor having calculated the predicted context information together with the relevant QoFC metrics. Response time does not only depend on the computational complexity of the prediction algorithm, but also on the time needed to get the required context information to make the prediction.

It should be noted that the individual prediction quality metrics and the overall predictor quality metrics are only applicable to prediction algorithms that have a probabilistic outcome. Although a prediction is almost always probabilistic, i.e. a prediction is not absolutely guaranteed to be correct, there exist forecasting algorithms that do not give a probabilistic result. In that case, it is not clear in current research what kind of quality metrics that are transparent and understandable could be applied. However, probabilistic algorithms are the majority of prediction algorithms that are effectively applied for context prediction in current state-of-the-art, and thus this limitation is not an issue in most practical applications.

2.2 Use of QoFC Metrics by Applications

The quality metrics can be used by an application needing future context information at two points in time:

1. The quality metrics can assist an application in choosing the context predictor(s) most suited to the application requirements in case several predictors are available that provide the same context information (with different QoFC).
2. When an application is provided with a concrete predicted value, the associated QoFC metrics help in deciding whether to use the individual prediction or not.

1) Choosing the Appropriate Context Predictor When an application requests a prediction of context information, several predictors can be available for the application. These predictors can be different algorithms, variants of the same algorithm (with different parameters) or being deployed on nodes with different resource characteristics, influencing e.g. response time. The predictors are all able to predict the same context information, but with different QoFC metrics. By taking into account the QoFC requirements requested by the application, a well-informed decision can be made to select one or more particular predictors which are available and which conform to the application QoFC requirements.

The QoFC metrics relevant to the decision process of choosing appropriate predictors are the input-dependent predictor quality metrics and the overall predictor quality metrics. Granularity, freshness and response time are quality metrics that act as preconditions on deciding whether the predictor is useful for the application or not. For example, a time-critical vehicle guidance application

will need guarantees on a fast response time of a future location predictor. On the other hand, for an interactive museum visit application, response time is not that important, but granularity of the provided future location is of significance. Thus, the input-dependent predictor quality metrics are a first filtering mechanism for an application in selecting suitable predictors. In a second step the overall predictor quality metrics are taken into consideration. Since these metrics assess the overall quality of the predictor in terms of the number of correct predictions relative to the total number of predictions, they allow to discriminate between different predictors and to choose the best-performing one(s).

2) Assessing the quality of concrete prediction results The second use of the proposed QoFC metrics is when the application has to decide whether to effectively utilize and act on a concrete predicted context value or not. Both the individual prediction quality metrics and overall predictor quality metrics play a role in the decision making. The metrics that are considered important for the application can be combined with developer-defined weights to obtain an overall confidence value in the predicted result.

For example:

$QoFC(pred) = w_1.P + w_2.C + w_3.E_c$, $where\ w_i = 1$, $0 < QoFC(pred) < 1$

The choice of the particular QoFC metrics included in the overall QoFC value and their associated weights depend on the application requirements. It should be noted that the QoFC metrics on themselves are objective metrics, as already mentioned. However, when they are effectively used by an application and/or combined with weights as in the formula above, the metrics become subjective. The decision whether to use the predicted context information does not only depend on the QoFC metrics mentioned above, but also on the cost of the actions performed as a result of the prediction. As such, both the QoFC metrics and the cost of the actions have to be taken into account and balanced.

The following rule D (adapted from [11]) balances the quality of the predicted context and the cost associated with deciding to act upon a wrong prediction:

$D : IF\ QoFC(pred) > A\ AND\ Cost(actions) < B\ THEN\ takeactions$

By balancing the quality of the predicted context and the possible cost of an action, the application can decide whether to effectively use the predicted context or not.

3 Methodological predictor component development

Reusable context and predictor plug-ins allow developers to separate the future context-awareness of an application from its functional logic. This section gives an overview of the steps a developer needs to take to create a predictor plug-in for the MUSIC context middleware [12]. The methodology consists of the following steps:

1. Identify the context type that will be predicted.

2. Identify the (context and domain) information that is required to make the prediction.
3. Identify context plug-ins for the different context types, take into account dependencies, and associate relevant domain knowledge to the context plug-ins.
4. Identify QoFC metrics applicable to the chosen prediction algorithm and relevant for the domain.
5. Implement the context model.
6. Define the XML descriptor of the plug-in and extend/implement the standard plug-in classes.

In the following, we describe each step in the development methodology, and apply it to a case study in the health care domain. A more detailed description of this case can be found in [2]. In hospital environments, nurses are equipped with a PDA on which they give in patient and other medical data. This data has to be synchronized as much as possible with a central server. Since there are only a few areas where (Bluetooth) network connectivity is available, and to prevent continuous polling for the network in order to save battery power, we predict the most likely time when the nurse will be at the location with network availability.

Step 1: Identify the predicted context type

The primary goal of the health care case is to efficiently synchronize patient data residing on the nurse's PDA with the central server. Since network availability is localized, the future context that needs to be predicted and will be provided by the predictor plug-in is the moment in time at which network connectivity can be expected to be available.

Step 2: Identify the information that is required to make the prediction

The predicted future context information is further analyzed to identify which context information is needed to make the prediction. In the health care case, we need to predict the expected time of network connectivity, which translates into predicting when the nurse will be at the location where the network is available. Firstly, the path the nurse follows through the hospital division depends on what kind of activity she/he is performing (e.g. bringing around food, measuring essential physical parameters, ...). Secondly, since the nurse follows a probable path visiting patient after patient (and thus location after location), the current location is of importance in predicting the series of future locations (including the location with Bluetooth activity). Thirdly, the time needed to reach the location with Bluetooth connectivity also depends on the activity, because for example medical treatment of patients will take longer on average than handing out the lunch. To summarize there are two relevant context types: current location and current activity, while the average time needed to perform the activity on a patient is domain-specific knowledge.

– *Current location:* the location of the nurse (i.e. the room she/he is in) can be inferred by placing RFID tags near the patient's bed or at the room entrance, which can be detected by an RFID-reader in the nurse's PDA.

- *Current activity:* as mentioned, the activity being performed has an influence both on the path typically being followed and on the time needed. The current activity is inferred by checking the kind of data the nurse is entering in the PDA (e.g. long daily patient treatment or just taking basic physical parameters like temperature and blood pressure, or gathering lunch or dinner).
- *Time needed for the activity:* The average time needed for an activity is domain knowledge inferred by analyzing a dataset containing typical nurse's activities and the durations of these activities.

Additional domain knowledge includes the location of the Bluetooth access point. The resulting hierarchy of context types is illustrated in Figure 2:

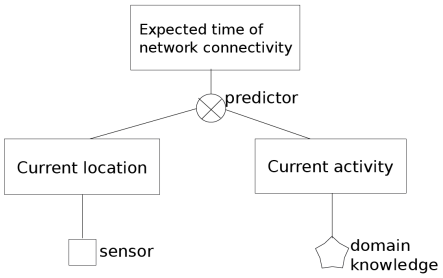

Fig. 2. Context type hierarchy for the nurse application scenario

The above Figure also gives information on the dependencies between the different context plug-ins (sensors, reasoners and predictors) that correspond to the context types, which is necessary information for step 3 below.

Step 3: Identify context plug-ins for the different context types, take into account dependencies, and associate relevant domain knowledge to the context plug-ins.

In this step, a context plug-in is defined for each context type identified in the previous step. For each context plug-in that is identified, an existing implementation can be reused or a new one can be constructed. In most cases, context plug-ins for basic context information will be available. On the other hand, since predictor plug-ins use application-specific domain knowledge, they have to be implemented for each application. However, a set of predictor skeletons can be provided that implement popular context prediction algorithms. For example, part of the prediction process in the health care case is predicting future locations based on the current location. Predictor skeletons implementing standard location prediction algorithms like markov chains are provided as libraries to the developer.

For our case we identified the following context plug-ins:

- *Current location plug-in:* a sensor plug-in that reads RFID tags to provide the current location of the nurse.
- *Current activity plug-in:* A reasoner plug-in that infers the current activity based on the nurse's input of information on the PDA or the time of day.
- *Future locations plug-in:* a predictor plug-in that, based on the current activity and the current location, predicts a series of future locations representing the most likely path the nurse will follow throughout the hospital division in performing the specific activity.
- *Expected time of network connectivity plug-in:* a predictor plug-in that predicts the expected moment of network connectivity by computing the time duration needed for the nurse to reach the location with Bluetooth activity. This predictor takes into account the path the nurse follows in performing her activities, and the average time duration needed to perform the activity to the individual patients that will be visited before reaching the Bluetooth range.

Figure 3 shows the dependencies between the different plug-ins which were identified.

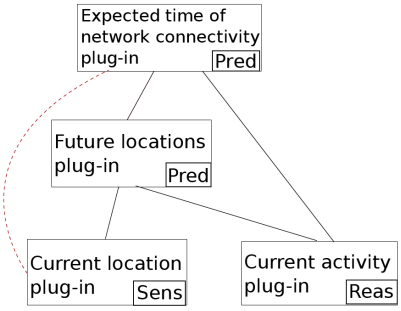

Fig. 3. Context plug-ins and dependencies for the nurse application scenario

Since the possibility must exist to check whether the predicted future context matches with what actually occurs, a dependency must be added between the predictor plug-in and the context plug-in that provides the current value of the predicted context type. This is denoted in the figure above by the red dashed line. For example, to check whether a predicted location is correct, the predictor needs input from the location sensor plug-in concerning the actual location at the predicted time. Also in this step, relevant static and dynamic domain knowledge identified in step 2 needs to be associated with the context plug-ins where the knowledge is effectively used.

Step 4: Identify relevant QoFC metrics

The QoFC metrics that were proposed in the previous section can be divided in two sets: those that are relevant for the predictor plug-ins general characteristics, and those that are relevant for individual concrete context predictions.

The overall predictor quality metrics and the input-dependent predictor quality metrics give information on the overall quality of the predictor plug-in. As such, they are properties of the predictor plug-in and instantiated in the QoFCMetadata field of the plug-in. On the other hand, the individual prediction quality metrics give information on the quality of a concrete prediction. So these metrics belong to the context object that represents an individual prediction and they are instantiated in the metadata field of the ContextElement object.

For the expected time of network connectivity plug-in of the hospital use case the following QoFC metrics are considered relevant:

- *Probability of correctness:* This metric corresponds with the probability given by the markov chain of being at the location with Bluetooth connection within n time steps.
- *Standard error rate:* This metric gives the frequency of the predictor plug-in being correct in predicting the time of network connectivity availability.
- *Granularity:* This quality metric corresponds to the uncertainty of the exact time of network availability due to the variance in the duration of the activity currently performed by the nurse. This means the predictor can provide context information with different granularities which depend on the specific activity being carried out by the nurse.

Step 5: Implement the context model

Each provided and required context type needs to be modeled as an entity-scope pair with associated representation and QoFC metadata according to the MUSIC context model [13]. We illustrate this with the *current location* context type:

- Entity: The current location belongs to the user entity: *concepts.entities.user—self.*
- Scope: The scope is the location of the user: *concepts.scopes.environment.location.*
- Representation: The location is represented as a String, e.g. *room123.*
- QoFCMetadata: The only individual prediction quality metric identified as relevant in step 5 was the probability of correctness, which is represented as a double in the metadata field *ProbCorr.*

Step 6: Define the XML descriptor of the plug-in and extend the standard plug-in classes The provided and required context types must be defined in the XML descriptor of the plug-in. Additionally, the symbolic name of the component, the class implementing the component and the provided interface must be specified to allow OSGi to dynamically activate and deactivate the predictor plug-in. The activate and deactivate methods are needed to start and stop the generation of context events and to initialize local variables and data structures. The *contextChanged* method handles context events that are received from other plug-ins. It analyzes the *ContextElement* encapsulated in the event to infer higher-level context information. Finally, everything is put in a JAR file for OSGi bundle packaging.

This section presented a step-by-step methodology that allows the developer to build context predictor plug-ins in a structured way. While parts of

the methodology (e.g. step 6) are tailored to the MUSIC context middleware, we believe the approach can be applied in general for developing context prediction components in other systems or frameworks. Identification of the relevant context types, quality metrics and the dependencies between the different components, representing context information in an ontology and incorporating domain knowledge are universal requirements in developing context prediction components.

4 Related work

Intensive research has been carried out in the domain of modeling and using quality of context information. Work by Buchholz et al. [3], Henricksen et al. [4] and Manzoor et al. [5] defined several quality metrics for context information. Buchholz et al. [3] argued on the importance of QoC for real-life applications to make effective use of provided context information. They defined QoC as any information that describes the quality of information that is used as context information. Authors presented an initial set of QoC parameters, being precision, trust-worthiness, probability of correctness, resolution and up-to-dateness. Other authors like Krause et al. [6], Sheikh et al. [7] and Abid et al. [8] have further added to these parameters. While the metrics proposed for QoC are also relevant for predicted context information, they are not sufficient. Due to the prediction step, additional uncertainty is added to the predicted context value. The metrics we presented contribute to quantifying this additional uncertainty. Boytsov et al. [9] mention as a major challenge concerning context prediction that there is a need for automated decision making which should be based on the quality of the predicted context. Also, Nurmi et al. [10] acknowledge the need for metrics in order to evaluate the quality of a context prediction. Prediction algorithms which have probabilistic outcomes, like the ones evaluated in our research, give a natural way in assessing their quality.

5 Conclusions

This paper presented some key issues towards achieving *future context* aware applications. Our main contribution is the introduction of a set of quality metrics for predicted context information, integrated in a development methodology for realizing in a structured and efficient way context predictor components. The methodology identifies the different context types involved, and the associated context plug-ins with their dependencies. Context information is modeled and quality information is added as metadata, both to the plug-ins as to the concrete context values. Future work includes the fusion of quality of context with QoFC information metrics, and the aggregation of different QoFC metrics in order to get a global view of the quality of the context. Also, instead of letting application developers decide when to effectively use the predictions based on the quality metric values, we have initial experiments on automatically learning when to take action or not using reinforcement learning techniques.

References

1. Yves Vanrompay, Stephan Mehlhase, Yolande Berbers, An effective quality measure for prediction of context information, Proceedings of the 7th IEEE Workshop on Context Modeling and Reasoning (CoMoRea) at the 8th IEEE Conference on Pervasive Computing and Communications (PerCom'10), Mannheim, DE, March 2010
2. Yves Vanrompay, Yolande Berbers, Efficient context prediction for decision making in pervasive health care environments: a case study, Supporting real-time decision making: The role of context in decision support on the move, Annals of Information Systems 13, pages 303- 318, Springer 2010
3. Thomas Buchholz and Michael Schiffers. Quality of context: What it is and why we need it. Proceedings of the 10th Workshop of the OpenView University Association: OVUA03, 2003.
4. Karen Henricksen and Jadwiga Indulska. Modelling and using imperfect context information. In Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, PERCOMW 04, pages 33, Washington, DC, USA, 2004.
5. Atif Manzoor, Hong-Linh Truong, and Schahram Dustdar. Using quality of context to resolve conflicts in context-aware systems. In Proceedings of the 1st international conference on Quality of context, QuaCon09, pages 144155, Berlin, Heidelberg, 2009.
6. Michael Krause and Iris Hochstatter. Challenges in modelling and using quality of context (qoc). In Thomas Magedanz, Ahmed Karmouch, Samuel Pierre, and Iakovos S. Venieris, editors, MATA, volume 3744 of Lecture Notes in Computer Science, pages 324333. Springer, 2005
7. Kamran Sheikh, Maarten Wegdam, and Marten van Sinderen. Middleware support for quality of context in pervasive context-aware systems. In Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops, pages 461466, Washington, DC, USA, 2007
8. Zied Abid, Sophie Chabridon, and Denis Conan. A framework for quality of context management. In Proceedings of the 1st international conference on Quality of context, QuaCon09, pages 120131, Berlin, Heidelberg, 2009.
9. Andrey Boytsov and Arkady Zaslavsky. Context prediction in pervasive computing systems: Achievements and challenges. In Supporting Real Time Decision- Making, volume 13 of Annals of Information Systems, pages 3563. Springer US, 2011
10. Petteri Nurmi, Miquel Martin, and John A. Flanagan. Enabling proactiveness through context prediction. CAPS 2005, Workshop on Context Awareness for Proactive Systems
11. Seng W. Loke. Facing uncertainty and consequence in context-aware systems: towards an argumentation approach, 2004
12. Nearchos Paspallis, Romain Rouvoy, Paolo Barone, George Papadopoulos, Frank Eliassen, and Alessandro Mamelli. A pluggable and reconfigurable architecture for a context-aware enabling middleware system. Vol. 5331 of Lecture Notes in Computer Science, pages 553570. Springer Berlin / Heidelberg, 2008
13. Roland Reichle, Michael Wagner, Mohammad Khan, Kurt Geihs, Jorge Lorenzo, Massimo Valla, Cristina Fra, Nearchos Paspallis, and George Papadopoulos. A comprehensive context modeling framework for pervasive computing systems. Vol. 5053 of Lecture Notes in Computer Science, pages 281-295. Springer Berlin / Heidelberg, 2008.