Quality of Context: Handling Context Dependencies

Tobias Zimmer
Telecooperation Office (TecO)
Universität Karlsruhe

ABSTRACT

Context dependencies are one of the major problems in future ubiquitous computing environments. In this paper we introduce *Genetic Relation of Contexts (GRC)*, a lightweight distributed algorithm to analyze interdependencies of context data in an efficient decentralized manner. We present first results of the evaluation of the system, proposing a relevant increase in context quality by the use of GRC, while at the same time energy consumption of computation tasks can be reduced. In our experiments GRC was able to reduce network traffic by up to 60% by filtering low quality contexts.

1. INTRODUCTION

Research on context processing in ubiquitous computing environments has left its early stage. Today we have access to a selection of well understood context recognition techniques and algorithms. Sensor nodes are available to implement and test context-aware application. We have access to communication systems that deliberately support the communication of context information in ad hoc networks.

The next step towards the original vision of Ubiquitous Computing [1] will be to use the available technologies to build personalized context-aware applications and bring them together to form large-scale multi-user ubiquitous computing environments. This new class of environments will consist of a heterogeneous set of applications and artifacts – everyday objects augmented with ubiquitous computing technology – from different sources. The environments will be highly dynamic in terms of interacting applications, as mobile users will enter and leave the scope of an environment continuously, taking some of their personalize applications and services with them. Processing context in these environments needs interoperability of applications from different sources to allow for sufficient diversity in services to attract users. Additionally interaction in context has to be manageable on a common basis to handle the complexity of highly dynamic large-scale ubiquitous computing environments that are hosting many heterogeneous applications.

Today most context-aware systems have comparably flat architectures: Sensor values are processed to contexts and theses contexts are used to adept the behavior of appliances. Only very rarely settings are encountered featuring multi stage context processing; settings where context information is communicated and then fusioned or aggregated to derive new contexts. This will most probably change soon. In large environments personalized context-aware application will need to propagate the context information they can access and derive to provide the user with benefits from all available services cooperating. High level, abstract context information will become more important to provide sophisticated context-aware services [2, 3].

Scaling effects are subject to research in networking and sensor networks. Focusing on hardware and protocol issues, current research in ubiquitous computing does rarely investigate scaling effects in context processing. Most published context processing architectures (context models) are not capable of handling degression in context quality that is caused by a multi-step aggregation processes – most provide no tools for handling context reliability at all.

In large-scale ubiquitous computing environments a new class problems based on the highly dynamic and modular architecture is encountered. Determining "processing trees" – graphs that reflect the path and evolution of context information – will not be possible anymore due to the high complexity. And so new ways of handling problems context dependencies, like *cyclic usage of context* and *splitting and multiplication of context*, have to be found. System engineers will also have to cope with problems that have already been identified to be of relevance for advanced context processing like *locality of context* and *ageing of context* [4, 5].

This new class of scaling induced problems in context processing need to be handled efficiently. Otherwise a large number of interdependent contexts of low quality will be communicated and processed, consuming energy and bandwidth in ubiquitous computing environments, and leading to unacceptably low context recognition rates of applications.

In this paper we present a lightweight context management algorithm that provides a solution to the above mentioned problems of *cyclic usage of context* and *splitting and multi-*

plication of context. It filters interdependent contexts of low quality without the need to extract and analyze context processing trees of ubiquitous computing environments. The algorithm is self-contained and does not need access to any additional semantic context information. So it can be applied independently from the used semantic context model, allowing for a seamless integration into existing context-aware applications and systems.

First we take a closer look at the problem scenario. Then *Genetic Relation of Contexts (GRC)* is introduced. GRC is an algorithm designed to handle typical problems of context dependency in highly dynamic large-scale ubiquitous computing environments. We analyze the function of GRC and finally present first results of the application of GRC in the AwareOffice [6] – our testbed for deploying context-aware application in the real world – and simulative results giving an impression of the performance of the system when applied in larger scale settings.

2. INTERDEPENDENT CONTEXTS

The analysis of large application scenarios [7] and simulations of large-scale ubiquitous computing setting has disclosed a class of problems connected to dependencies of context data. Two of the main representatives of this class of problems that were identified are called "*splitting and multiplication*" and "*cyclic usage*" as stated before.

Splitting and multiplication can happen whenever a context is consumed by many artifacts of a similar kind (see Figure 1). In this case context a is used in parallel by many appliances of type A that all produce contexts of type b. The newly produced contexts are all derived from the same source data – context a. If an application of type B seeks to consume this multiply available context type b, it may be necessary to know whether theses pieces of context information are independent or not – e.g. if data fusion algorithms like Kalman filters should be used on the input contexts.

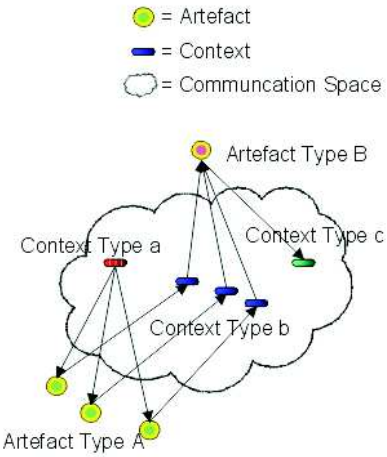

Figure 1: Splitting and Multiplication

A well known example scenario were splitting and multiplication becomes relevant resides in the interactive office domain: To adapt to a meeting situation personal devices like PDAs and mobile phones are equipped with with a context-aware application that can adjust the alarm function of the device according to the current context. If the personal devices propagate the meeting context they detect and use to adapt themselves, to provide other appliances with this abstract information, the meeting context becomes multiplied by the number of artifact providing it. Other applications like interactive doorplates [6] that use the meeting context to adapt themselves, have to decide whether the multiply available meeting contexts are independent – in that case it can be interpreted as an amplification of the context information, increasing its reliability – or whether all the meeting contexts where derived on basis of the same source information. In this case the aggregation of the available meeting contexts would not increase the reliability of this context.

Cyclic usage means that context data is processed in a way that an artifact is consumer of a context that is derived – in one or more steps – from another context that it produced itself. Cyclic usage of context may not be a problem if the number of intermediate steps is high or the cycle is established intentionally. In general it is not possible to rule out that cyclic usage will yield problems in the applications. That is especially true if cycles are present, that conserve contexts over a long time without comprising new contextual information.

Figure 2 shows a schematic view of a cycle formed by three artifacts. In this setting two artifacts – B and D – can produce a context of type b that is consumed by artifact A. Once artifact D produces a context of type b, artifacts A, B and C establish a cycle of context procession.

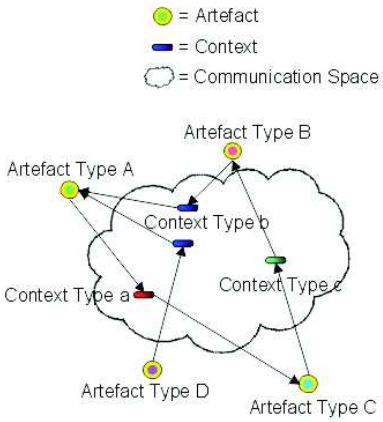

Figure 2: Cyclic Usage of Context

Cycles can easily form in environments when applications are present that can derive a target context from different sources. A simple scenario leading to possible cycles is an extension of the meeting scenario introduced above: If the personal devices – e.g. PDAs and mobile phones – are set up to collaborate for saving energy, they can seek to reduce the needed power for processing by not deriving the meeting context by themselves, but trying to "reuse" the meeting context provided by another artifact. In this case the personal devices can either derive the meeting context from other context types available in the environment or consume a meeting context directly. Still the device would produce a meeting context itself, leading to a situation in that even

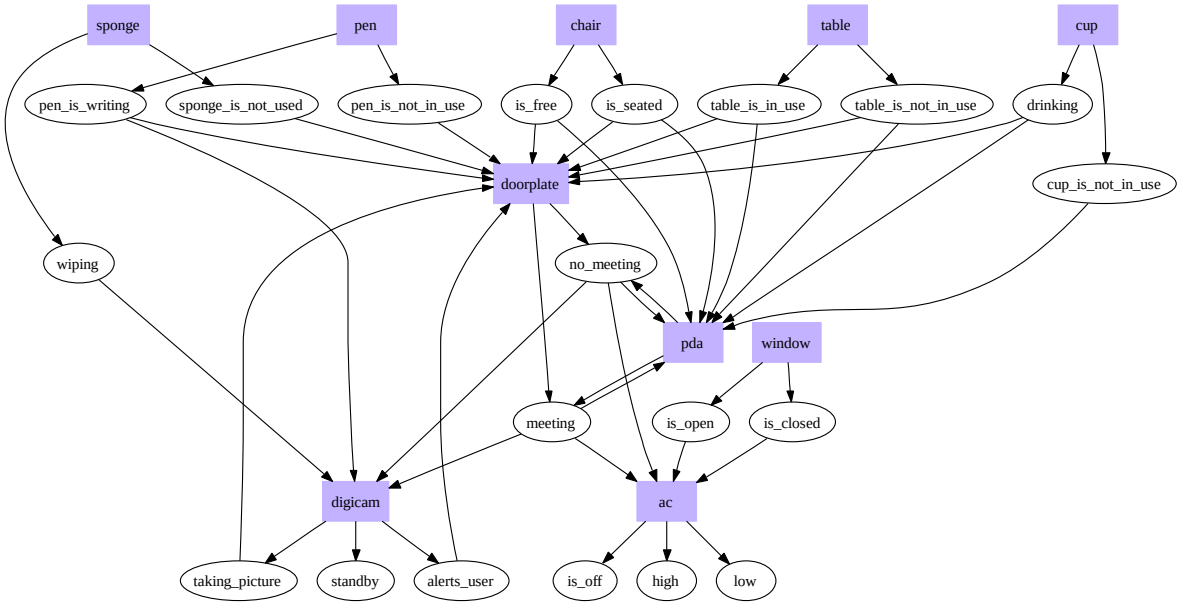

Figure 3: Cyclic Usage of Context

two devices can establish a permanent meeting context by taking turns in processing it.

These very basic application scenarios are meant to abstractly illustrate problems that can be solved by the GRC system. In real-world environments many applications from different providers are interwoven by the exchange of context information. An easy way to cope with the problems of interdependent contexts would be to analyze the processing graph of an environment to detect cycles and splits. Unfortunately there are some major problems with that approach: Firstly, in highly dynamic environments the processing graph is not stable. Every time an application leaves the environment or a new application enters, the context processing topology changes and so does the processing graph. Dynamically refreshing and analyzing the processing graph would generate an enormous overhead. Secondly, in large settings the processing graphs become complex, consuming much time and energy to analyze. And thirdly, extracting the processing graph from an environment and analyzing it in its whole, needs a general understanding of all contexts that are communicated and processed in the environment.

To illustrate the complexity of the task, Figure 3 shows a reduced processing graph (subscription graph) from the AwareOffice. The subscription graph only shows the processing dependencies on basis of the types of artifacts and types of contexts they use.

In this reduced type of processing graph only the cycle of meeting contexts established by the PDAs can be detected directly. Other possible cycles in this graph, like the ones established by the doorplate and digital camera or the doorplate, digital camera and PDAs are harder to detect. Originally the setting shown here consists of 5 cups, 6 chairs, three tables, three windows, two pens, one sponge, one digital camera, one doorplate, on air condition and 5 PDAs; altogether an environment of only 28 artifacts.

A section of the complete processing graph is shown Figure 4. It only contains the cups, chairs, pens, PDAs, doorplate and camera. Even though the graph contains all the information on context dependencies, the splitting is hard to detect, because not every single piece of context information is represented as a separate node, but contexts of the same type are subsumed in one node. Still this graph representing only 20 artifacts and their contexts gives an impression of the complexity of processing dependencies in real-world environments.

As a solution to finding cycles and splittings in large-scale ubiquitous computing environments we propose *Genetic Relation of Contexts (GRC)*. This algorithm provides information on the interdependencies of contexts with minimal overhead in terms of computing and memory usage. The main advantage of GRC is, that no information on the topology of context processing is needed, allowing to run it in a distributed manner without centralized management. The algorithm is implemented as part of the software stack of every artifact. By providing common service access points (SAPs), GRC can communicate dependency information to the used semantic context processing layer.

3. GENETIC RELATION OF CONTEXTS

Genetic Relation of Context is a method based on ideas derived from biological genetics and genetic algorithms. It is designed to provide an understanding of the relationship between different pieces of context information. As part of a context management layer, GRC is located between communication and context processing in the application stack as shown in Figure 5.

On that level context information is just a data type with some context specific attributes. The semantical meaning of

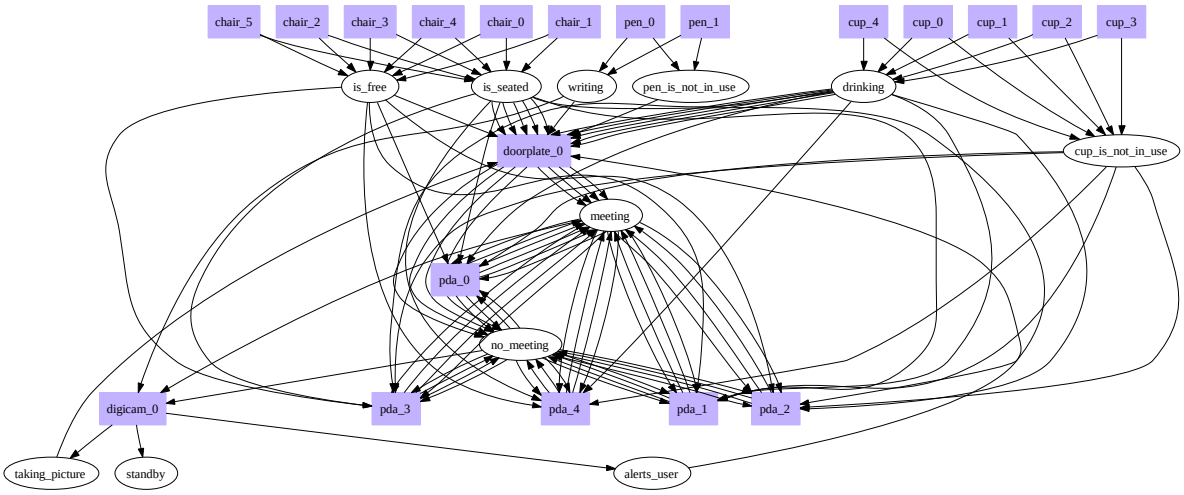

Figure 4: Splitting and Multiplication

context is evaluated one layer higher by the semantic context processing. This separation of context management and semantic context processing allows for the use of GRC without being restricted in the choice of a semantic context model.

The intention of GRC is to provide a direct measure of the degree of relationship of derived context data. This measure together with the original contexts is then provided to the context processing layer.

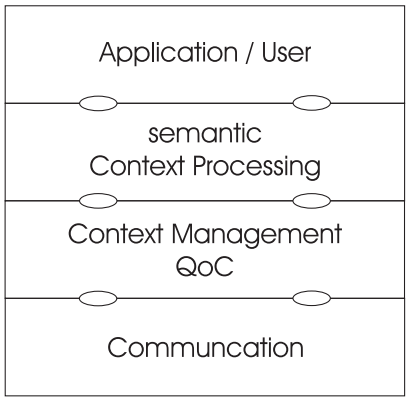

Figure 5: Application Stack

3.1 Genomes for Contexts

GRC establishes a measure of relationship by introducing a context genome. The genome represents the identity of the information carried by a context.

When a basic context is generated, a new genome is generated at the same time; when a context is derived from other context information, the genomes of the parent contexts are combined to build the genome of the newly created context. An application that itself derives new contexts from other source contexts, can compare the genetic finger prints of the source contexts to determine whether these are independent or somehow related. The semantic context processing layer can then decide whether and how to process the contexts. This can range from discarding contexts, to specifically selecting the processing algorithm that suites the source data best.

When initially derived from sensor data, each context C is associated with a randomly generated bit-vector of length l representing its genome. The genome Γ is then sequenced into n genes γ.

$$\Gamma_C := (\gamma_{C,1}, \gamma_{C,2}, \cdots, \gamma_{C,n}) \tag{1}$$

By sequencing Γ ever resulting gene becomes associated to a functional locus denoted by the index n. Other than in genetic algorithms the locus of a gene is of functional importance as it stores the relationship information. In classical genetic algorithms the genome itself is functional data. Its structure is directly mapped to the "fitness" of the pattern (solution) by an objective function.

In GRC the genome itself is not functional in this sense. It does not represent the fitness or quality of the context, but its informational identity. Only by comparing the genetic finger prints of contexts GRC produces a measure of relationship that can be mapped to a notion of context quality.

The parameters influencing the performance of GRC are the number of genes n in a genome and the size of the single genes. The size of genes determines how many different alleles exist. An allele is a value the gene can take. In a bit vector genome the number of genes and alleles is associated with the length of the genome by the following formula:

$$l = \frac{n \ln r}{\ln 2} \tag{2}$$

Increasing n leads to a higher resolution of relationship, whereas increasing r results in a lower systematic error rate. More details on the influence of the choice of system parameters is discussed in Section 4. In general, in contrast to bi-

ological genomes, the context genome has to be very small; e.g. even *escherichia coli* an intestinal bacterium has about 4.500 genes encoded in $4,6 * 10^6$ base pairs each carrying 4 bit of information. This sums up to 2,19 MB of genetic information in total. The bit-vector representing the genome of a context typically carries 50-200 bytes. This reduction is possible because, firstly the context genome does not have to encode any functional information other than the relationship of contexts, where the biological genome stores lots more information. Secondly the context genome can be designed to match the needs of context-aware environments in terms of the number of generations of contexts that have to be distinguishable.

When a context is derived from other source contexts, but not directly from sensor data, the source contexts are already associated with a genome that can be used to derive the genome of the newly produced context. This recombination of existing genomes to form a new one is called "crossover".

3.2 Probabilistic Multi Site Crossover

To preserve the information of relationship a specially designed probabilistic crossover method is applied. In genetic algorithms a large variety of crossover methods is used [8, 9, 10]. These are designed to preserve schemata. Schemata are patterns in the genome used by genetic algorithms that represent parts of the solutions to an optimization problem. A detailed discussion of schemata can be found in [11]. The important feature that make classical crossover methods unsuitable for GRC is that schemata are independent from the locus of the genes in the pattern. So, common crossover operators used in genetic algorithms do not preserve the loci of genes that GRC uses to represent the relationship information.

Probabilistic Multi Site Crossover (PMSC) is a new crossover operator specially designed to preserve relationship information in the genes of contexts. In Figure 6 on the next page, the basic application of the PMSC-operator is shown. Two parent genomes Γ_{C_1} and Γ_{C_2} are recombined into a child genome Γ_{C_3}. The different alleles of genes are represented by capital letters. PMSC steps though Γ_{C_1} and Γ_{C_2} locus by locus randomly choosing one parental gene at each locus and marking it to be handed down to the child. Then the Γ_{C_3} is generated by copying the alleles of the marked genes to the corresponding loci in the child genome. A mutation operator is not applied as that would adulterate the relationship information.

The example above shows the most basic application of the PMSC-operator. Context derivation algorithms can use more than two input contexts, but will produce always one output context at a time. PMSC can be used to recombine the genomes of multiple source contexts. The maximum number depends on the number of genes n, and how many generations of contexts have to be distinguishable in the environment.

Alternatively to handing down the genes of each parent with the same probability, PMSC allows to adjust the heredity probability to represent the amount of information that is handed down from a source context to a target context in the derivation process. By that means GRC can be used also to provide a direct measure for the amount of information that is part of multiple contexts.

3.3 Degree of Relationship

To determine the degree of relationship between input contexts, the consumer has to analyze their genomes. The analysis of genomes has to be done pairwise, resulting in a relation matrix in case of more than two source contexts.

First an indicator function f_i is defined:

$$f_i(\gamma_{C_1,i}, \gamma_{C_2,i}) = \begin{cases} 0 & : \gamma_{C_1,i} \neq \gamma_{C_2,i} \\ 1 & : \gamma_{C_1,i} = \gamma_{C_1,i} \end{cases} \quad (3)$$

This function produces a 1 if the alleles of the genes in a locus of the genomes of both contexts match and 0 otherwise. The degree of relationship is then computed by summing up the results of f_i over all loci and dividing by the number of genes:

$$rel(C_1, C_2) = \frac{1}{N} \sum_{n=1}^{N} f_n(\gamma_{C_1,n}, \gamma_{C_2,n}) \quad (4)$$

This method allows for very fast and computational inexpensive determination of the degree of relationship of contexts. The resulting ratio can directly represent the ratio of information both tested contexts have in common.

Two standard patterns of context dependencies will be used to illustrate the functionality of GRC: Figure 7 shows a simple pattern called "single line of inheritance".

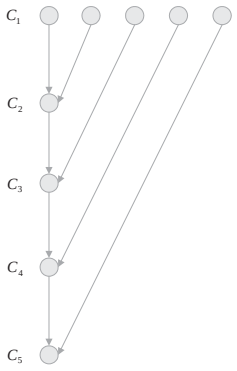

Figure 7: Standard Pattern: Single Line of Inheritance.

In this pattern in every generation of context one new piece of context information is added to an unbroken line of context derivations. Figure 8 shows the values of $rel(C_1, C_i)$ – the degree of relationship – in the lower curve. In this simulation the genes of each parent were handed down with a probability of 0.5, resembling the basic case of PMSC introduced in Section 3.2.

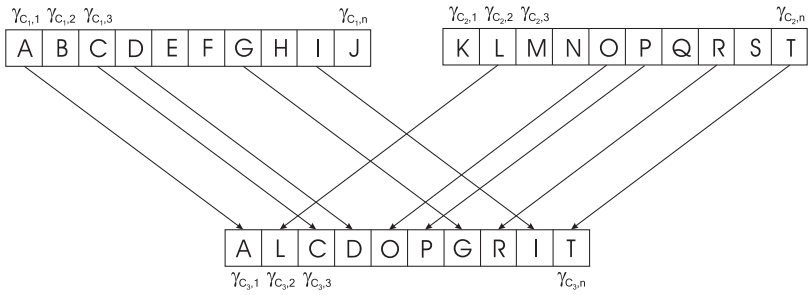

Figure 6: Probabilistic multi site crossover (PMSC)

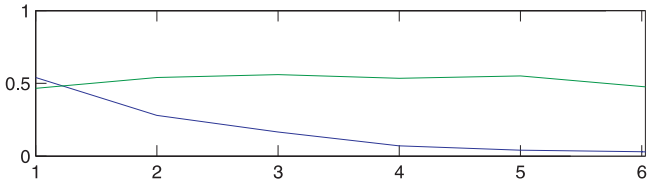

Figure 8: Degree of Relationship: Single Line of Inheritance.

The upper curve shows the $rel(C_{i-1}, C_i)$ values; the relation of every offspring to its direct parent. The y-axis in the figure represents the degree of relationship, the x-axis denotes the number of generations. As expected the degree of relationship drops from a starting value of approx. 0.5 with every generation. The simulation was set up with a genome length of 100 genes with 256 alleles.

The second standard pattern – "multi relation" – is shown in Figure 9. Beginning with two independent contexts C_1 and C_2, new contexts are derived by mating a context with one of its parents. This pattern can e.g. be produced by cyclic processing of contexts.

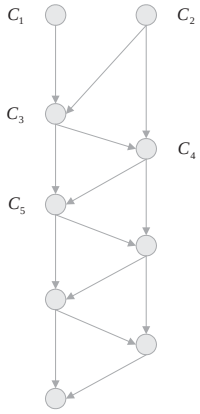

Figure 9: Standard Pattern: Multi Relation.

Figure 10 shows the simulation results for "multi relation", again using a genome length of 100 genes with 256 alleles. The y-axis represents the degree of relationship, the x-axis denotes the number of generations.

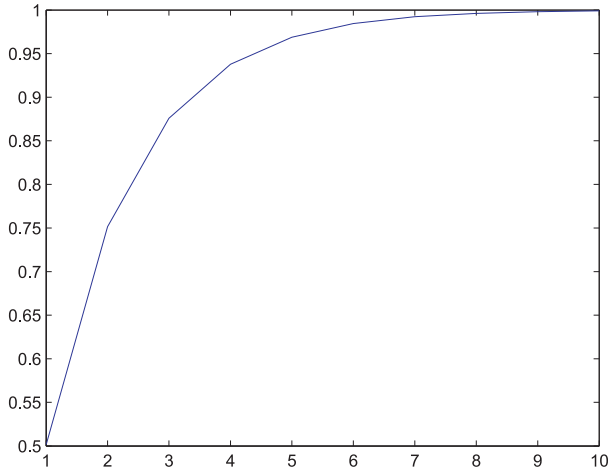

Figure 10: Degree of Relationship: Multi Relation.

The curve shows the $rel(C_{i-1}, C_i)$ values, starting from $i = 3$. As expected the degree of relationship increases with ever generation: From a value of approx. 0.5 in the first generation to 0.75 in second and 0.875 in third.

4. ANALYSIS AND EVALUATION

In this section we analyze and evaluate the performance of the GRC-system. Firstly the theoretical background of GRC is reviewed, then first results of the application of GRC in simulated and real-world environments are presented.

4.1 Systematic Overestimation of the Degree of Relationship

GRC is a probabilistic method to determine the relationship of context in an ubiquitous computing environment. So it does not give the real degree of relationship as an output, but an estimate for that value. The quality of that estimate depends on variable parameters that can be freely set up to suite the demands of the environment GRC is used in, like the number of genes and alleles.

The degree of relationship provided by GRC is intended to be used to filter contexts from processing if they do not meet the requirements of context consuming application in terms of independence from each other. That means, the application developer can provide a threshold for the degree of relationship of the input contexts an application can accept

in different situations. Contexts that fall below this value are processed, where context that exceed the threshold are discarded.

Due to its systems design GRC tends to sightly over estimate the degree of relationship between contexts. The finite size of the single genes – the number of alleles – can lead incidentally, duplicate alleles in a locus when genomes are produced from random bit-vectors. If the bit-vectors are produced with an uniform distribution, the probability $P(i)$ to encounter i duplicate alleles in two random genomes with n genes and r possible alleles is:

$$P(i) = \left(\frac{1}{r}\right)^{i} \left(1 - \frac{1}{r}\right)^{n-i} \binom{n}{i} \tag{5}$$

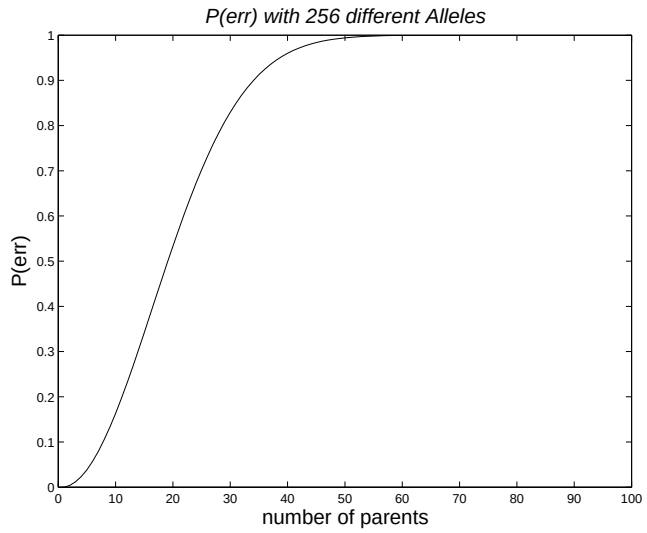

Figure 11: Error Probability: Overview.

The probability to encounter an error caused by duplicates in a derivation process that involves m parent genomes, is bounded by the probability $P(err)$. $P(err)$ describes that from beginning – after the random selection of genomes – at least two of the parents have the same allele in one locus. This is complementary to the probability of none of the parents having the same allele in one locus:

$$P(err) = 1 - \prod_{j=0}^{m-1} \frac{r-j}{r} = 1 - \frac{r!}{r^m(r-m)!} \tag{6}$$

Figure 11 shows the $P(err)$ for a setting with 256 alleles and up to 100 parent contexts involved in one single derivation process. In Figure 12 the same curve is shown for up to 10 parents, which is much more relevant in real-world environments.

Independent from the derivation process, the maximum estimation error would occur if all duplicates would lead to an error. This is not very likely as duplicate errors can vanish

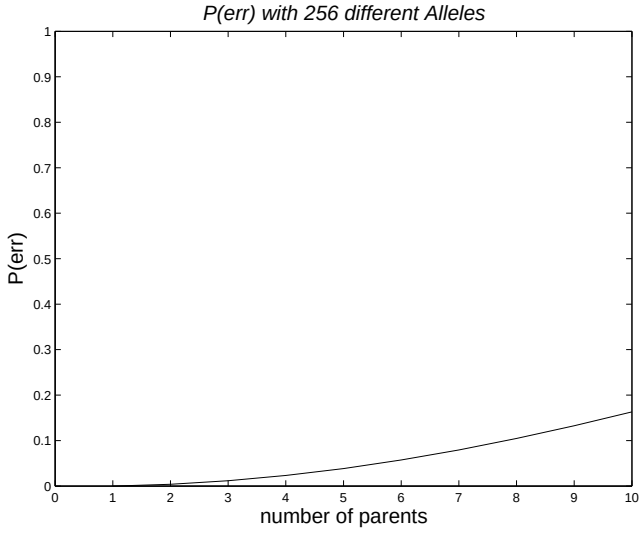

Figure 12: Error Probability for up to 10 Parents.

from a derivation process if a duplicate allele that let to an error in an early stage, is not chosen to be hand down in a later derivation step.

With that knowledge we can provide an estimator $P(\beta|\alpha)$ for the real degree of relationship α from the degree of relationship provided by GRC β. This estimator models the most common derivation process that involves two parent contexts that are processed to build a new context.

$$\alpha_\sigma := \alpha + \left(\frac{1}{r}\right)^{\frac{1}{\alpha}} - \alpha \left(\frac{1}{r}\right)^{\frac{1}{\alpha}} \tag{7}$$

$$P(\beta|\alpha) = \alpha_\sigma^{N\beta} (1 - \alpha_\sigma)^{N(1-\beta)} \binom{N}{\beta N} \tag{8}$$

α_σ is the probability of a locus holding a duplicate allele after the application of the PMSC-operator.

Analyzes of application scenarios [7, 6] have produced a set of standard patterns of context dependencies showing that even in large-scale environments the number of different seed contexts that are involved in one single derivation process is very limited. The above estimator models the most common derivation process, while Formula 6 provides the basis for an estimator for the upper bound of the error in more complex processes by exchanging α_σ for $P(err)$ in Formula 8.

4.2 System Parameters

The first step in applying GRC in a real environment is to find suitable system parameters to set up the system. For our evaluation, these parameters were derived from extensive simulation of the GRC-system in MATLAB. As basis for the simulations we used our results from the application analyzes. Table 1 shows the variances and means of the degree of relationship for a simulated derivation process of

	$n = 10$	$n = 100$	$n = 1000$
$r = 32$	$\overline{rel} = 0.50943$ $var(rel) = 0.024883$	$\overline{rel} = 0.50761$ $var(rel) = 0.0025229$	$\overline{rel} = 0.50774$ $var(rel) = 0.0002514$
$r = 256$	$\overline{rel} = 0.50316$ $var(rel) = 0.024933$	$\overline{rel} = 0.50087$ $var(rel) = 0.0025735$	$\overline{rel} = 0.50076$ $var(rel) = 0.0002514$
$r = 1024$	$\overline{rel} = 0.50074$ $var(rel) = 0.024918$	$\overline{rel} = 0.50059$ $var(rel) = 0.002493$	$\overline{rel} = 0.50031$ $var(rel) = 0.00024781$

Table 1: Simulation of System Parameters

one generation and two parents. The number of simulated derivation processes was 10000.

From the simulation we derived a setup with $n = 100$ and $r = 256$ as the best compromise of performance, memory consumption and needed computing power. This parameter set we used to setup several environment-simulation using our Java Context Processing and Communication Simulator "context_sim". We also used it to perform first real world tests of GRC in the AwareOffice setting.

4.3 Results

First results from the simulations of larger ubiquitous computing environments are very promising. The simulation in context_sim resembles the AwareOffice setting to yield results comparable to those from the real world trail. Figure 3 shows the processing graph of the simulation. As explained in Section 2, this setting contains two problem scenarios – *splitting and multiplication* and *cyclic usage*. 28 artifacts were simulated – 5 cups, 6 chairs, three tables , three windows, two pens, one sponge, one digital camera, one doorplate, on air condition and 5 PDAs. The simulations ran for 1000 activity steps of all artifacts each.

In the first simulation runs the GRC-system was switch off to produce reference results representing a common ubiquitous computing environment with no special quality management system in operation. Based on a survey on common achievable context recognition rates [12] the mean recognition rate for all context was set to a comparably high value of 0.9 with a standard deviation of 0.05. The statistics for one of the PDA devices after the simulation run read as follows:

```
-------------------------------------------
Total active steps:  1000
Total number of contexts consumed:  3348
Total number of contexts produced:  1436
Overall recognition rate:  27.27%
Mean relationship of consumed Cs:  0.6698
Var of relationship of consumed Cs:  0.22008
-------------------------------------------
```

The first line of the print shows the number of active simulation steps for that artifact. The second and third line show the number of contexts consumed respectively produced by the PDA. The device received more than 3 contexts every simulation step, while it needed an average of 2.33 input contexts to derive an output context. This indicates the very high number of contexts communicated in total in the environment during only 1000 active steps. The overall context recognition rate of the PDA is computed from the initial value of 90% for all first level contexts. For the following derivation process, a mean recognition rate of 90% is assumed as well (see above). Splitting of contexts and the possible cycle in the meeting context lead to a recognition rate of only 27.27%, which would not be acceptable for a user. As the PDA devices only produce two different contexts, this is worse than blind guessing the context. The errors and decrease in context quality are also reflected by the high degree of relationship of the contexts the PDA has consumed. The value of over 0.6 indicates that most of the contexts had more than half of their information in common. The high variance of the relationship points to a very unstable system in terms of information independence and context quality.

In a second run of simulations GRC was activated. All other parameters of the simulation remained unchanged. The results show significant differences to those produces without GRC in operation:

```
-------------------------------------------
Total active steps:  1000
Total number of contexts consumed:  1395
Total number of contexts produced:  465
Overall recognition rate:  72.92%
Mean relationship of consumed Cs:  0.0039
Var of relationship of consumed Cs:  3.77E-5
-------------------------------------------
```

In this simulation run the GRC-system was set to filter out all contexts that had a degree of relationship of greater than 0.5. This results in a significantly reduced number of communicated contexts in the environment. Only 1395 context were consumed and 465 were produced, averaging to 3 consumed input contexts per output context. This means a reduction of the load on the communication channel of about 60%. At the same time the recognition rate of the PDA increased by over 45% to 72.92%. This is due to the filtering of highly interdependent contexts that led to errors in context recognition in the first simulation setup without GRC activated. The increase in context quality is also reflected by the low mean degree of relationship of the contexts that were processed by the device.

As the environment for the real world experiment we used the AwareOffice setting at TecO. In the AwareOffice artifacts are implemented on basis of the Particle Computer platform [13]. The standard context processing and communication stack of the Particle Computer – ConCom [14] – was added the GRC functionality. The AwareOffice environment at the moment host 27 active artifacts: 2 AwarePens, an augmented whiteboard, one digital camera, 12 active chairs, 4 tables, 6 windows and a doorplate. First preliminary test-

ing results support the findings from the simulation. The gain in context quality still was not as significant due to the fact that the real AwareOffice actively tries to cut cycles by means of timestamp and address filtering, both not suitable approaches in large heterogeneous settings.

5. CONCLUSION AND OUTLOOK

With GRC a lightweight context management system becomes available that is able to solve some of the major problems of context communication and processing connected to context dependencies in large heterogeneous ubiquitous computing environments. As more personalized context-aware services become available the number of contexts will increase further, so efficient context management techniques are needed. The processing graphs generated by small simulations (see Figure 3 and 4) give an impression of the complexity in unsupervised heterogeneous and highly dynamic settings.

GRC promises to solve problems that are connected to interdependencies of context information. It can provide semantic context models with information on the degree of relationship of contexts, allowing for filtering highly related information, or selecting optimal algorithms for processing of contexts based on their level of interdependence.

The theoretical analysis of GRC in Section 4 shows the soundness and applicability of the presented system. The results presented in this paper show the high potential of the method in building integrated context management systems ensuring the quality of context in ubiquitous computing environments. The context recognition rate of a simulated artifact could be increased by up to 45%. While the communication overhead caused by low quality contexts dropped by 60% at the same time.

The next step in research on this topic will be to integrate GRC with other context quality algorithms, that focus on problems GRC alone can not solve. This will be in the first place temporal issues of context aging, spatial constrains induced by the locality of context and context reliability to provide a measure of the semantic quality of context information [15].

6. REFERENCES

[1] M. Weiser. The computer for the 21st century. *Scientific American*, 265 (3):66–75, 1991.

[2] Yoshinori Isoda, Shoji Kurakake, and Kazuo Imai. Context-aware computing system for heterogeneous applications. In *Proceedings of the First Internaltional Workshop on Personalized Context Modeling and Management for UbiComp Applications*, 2005.

[3] Tobias Zimmer. Towards a Better Understanding of Context Attributes. In *Proceedings of PerCom 2004*, pages 23–28, Orlando, USA, March 2004.

[4] Albrecht Schmidt. *Ubiquitous Computing – Computing in Context*. PhD thesis, Lancaster University, November 2002.

[5] Saul Greenberg. Context as a Dynamic Construct. *Human-Computer Interaction*, 16 (2-4):257–268, 2001.

[6] Tobias Zimmer and Michael Beigl. AwareOffice: Integrating Modular Context-Aware Applications. In *Proceedings of the 6th International Workshop on Smart Appliances and Wearable Computing (IWSAWC)*. IEEE Computer Society Press, 2006.

[7] Vlad Coroama, Jörg Hähner, Matthias Handy, Patricia Rudolph-Kuhn, Carsten Magerkurth, Jürgen Müller, Moritz Strasser, and Tobias Zimmer. Leben in einer smarten umgebung - ubiquitous computing: Szenarien und auswirkungen. *Gottlieb Daimler- und Karl Benz-Stiftung*, 12 2003.

[8] John H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.

[9] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive systems*. PhD thesis, University of Michigan, 1975.

[10] William Spears and V. Anand. A study of crossover operators in genetic programming. In *Proceedings of the Sixth International Symposium on Methologies for Intelligent Systems*, 1991.

[11] David E. Goldberg. *Genetic Algorithms in Search, Optimazation, and Machine Learning*. Addison Wesley, 1989.

[12] Thomas Ring. Performanceanalyse kontextsensitiver anwendungen. Technical report, Universty of Karlsruhe, Germany, ISSN 1432-7864, 2006.

[13] The Particle Computer Company. Website, accessed: 01/2006. `http://www.particle-computer.de`.

[14] Albert Krohn, Michael Beigl, Christian Decker, and Tobias Zimmer. ConCom - A language and Protocol for Communication of Context. ISSN 1432-7864 2004/19, University of Karlsruhe, 2004.

[15] Tobias Zimmer. Qoc: Quality of context - improving the performance of context-aware applications. In *Adjunced Proceedings of Pervasive 2006*, 2006.