Auto-adaptation Driven by Observed Context Histories

Jürgo Preden
Tallinn University of Technology
Ehitajate tee 5
19086 Tallinn, Estonia
+372-620-2109
jurgo.preden@ttu.ee

Johannes Helander
Microsoft Research
1 Microsoft Way
Redmond, WA 98052, USA
+1-425-882-8080
jvh@microsoft.com

ABSTRACT
Embedded computing devices that interact with humans and the real world hold great promise in making our lives more comfortable and convenient – perhaps allowing independence longer and later in life, or better understand the changes in our natural environment.

The biggest difficulty in taking advantage of these computers is that they need too much assistance from us, starting with configuration, with adapting to new dynamic requirements, and ending in learning from our intent. The ubiquity of computers makes the situation only worse—telling all the little computers what to do is easily harder than simply doing the task yourself.

We claim that the only way to make ubiquitous computing a practicality is to enable the computers to figure out what to do on their own. Observe and learn, or perish.

This paper proposes a framework for automatic configuration and adaptation using learning and prediction based on observed context histories. A software architecture for describing, recording, analyzing, and reacting to physical or computational variables is substantiated with a case study that self-tunes distributed real-time tasks in an entertainment scenario. The measured results are generalized, using stochastic or physical models, to apply to a large number of problems that allow ubiquitous computing to become a reality.

Keywords
Invisible computing, distributed systems, context history

INTRODUCTION
A key point to note when creating invisible computing systems is that the task of such systems is to provide an ongoing service, instead of transforming a single static input into an output. This feature makes the invisible computing systems inherently interactive in a computer theoretical sense, as defined by Wegner in [9], which in turn means that these systems can't be modeled [2] or implemented using traditional algorithmic models. Rather new unconventional approaches are required to successfully realize such systems. One approach that has been suggested is context aware systems or systems that are able to exploit context histories. Context aware systems are very closely related to the fundamental interactive computing concepts, namely that the outcome of current computations (or the behavior of the system) is affected by the current inputs and the past operation and interactions of the system.

The context information can be partitioned in two large sets: *computing context* and *physical context*. In the current approach the *user context* is viewed as being part of the *physical context*. In both categories exist numerous properties that can be collected and analyzed individually, each *context property* expressing the state of a computing or a physical phenomenon.

The computing context contains properties that describe the features of the computing nodes in a system. Generally these properties are hidden from the user and the outside world, although in some cases the properties may be quite apparent – the existence of computing nodes in the vicinity of a node is generally observable. The computing properties are for example the run times of functions on different nodes, the availability of services from peer nodes, network delays, processor or memory utilization of specific nodes, etc. The information on the computational context is obtained by monitoring the behavior of computing systems and the interactions between the computing systems.

The physical properties reflect the state of physical phenomena in the real world. The physical phenomena include both inanimate and animate (including humans) objects, *which in some situations may become subjects*. Generally the nodes obtain information on the physical context properties via sensors, but there may be other ways to attain such information, e.g. from a more powerful node that has a better and possibly more general view on the physical environment.

The values of the properties in the two groups may be correlated in some cases which does not affect the overall partitioning of the properties. For example a physical event that reflects the change of some physical context parameter value may also affect the total network traffic in a network which in turn may result in increased network delays, thereby creating a correlation between the physical and computational context properties.

Depending on the phenomenon that a context parameter describes the methods for utilizing the context history of

the parameter for the prediction of future values of the context parameter may be different. We believe that relatively simple stochastic or statistical methods (when compared to formal mathematical analysis methods) similar to the methods of technical analysis used in economics [5] should suffice for most cases of context parameter prediction. The disclaimer that past performance does not guarantee future results, while true, is less relevant than in stock markets. The stock markets are inherently competitive and the participants try to outsmart each other (including predicting each others' predictions), affecting the market itself and thereby changing the base on which the predictions are made. Instead embedded computing systems are typically collaborative and driven by physical observation. The situation would also be somewhat more complicated if all computing systems (including processors, memories and networks) would be highly adaptive.

In the following section a general architecture is presented that makes provisions for utilizing context histories in an application specific way. The final sections of the paper present a case study and performance measurements of that experiment with some of the concepts of a prototype implementation of the presented architecture.

ARCHITECTURE

To be able to systematically monitor and predict the context parameters of the two context groups as categorized above we propose an architecture that relies on the usage of metadata to describe (among other things) the set of functions involved in a computing scenario, the interactions between the nodes executing the functions and the approaches used for monitoring the execution of these functions. We introduce the concept of a computing partiture – a collection of metadata about a computing scenario – as the source of information for the nodes executing the scenario. In addition to describing a computing scenario the partiture also allows describing how the context information required for a computing scenario is collected and used.

The partiture does not contain details of the implementation of the functions involved in the partiture – it only describes the functions that are involved and the metadata relevant to these functions. Neither does the partiture contain information on the specific nodes that should execute the partiture but it rather describes the functions that are executed as part of the partiture. The functions described in a partiture can run on one or more nodes depending on the

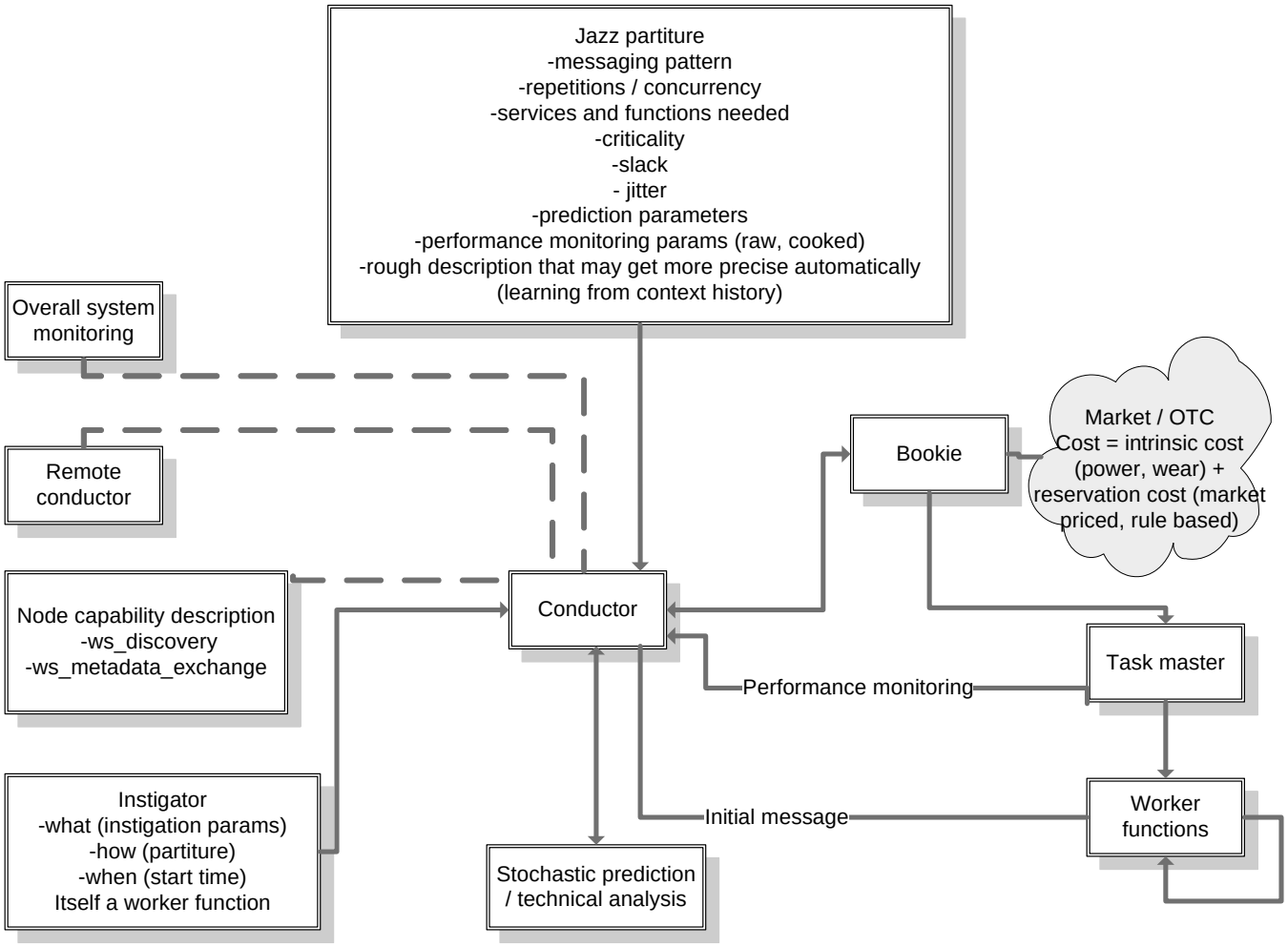

Figure 1. General architecture of the system

details of the partiture and the availability of resources at the nodes in the given network. Each node contains a set of application specific worker functions and a set of functions for computing and predicting the various context parameters.

The partiture describes the interactions (messaging patterns) between the functions including the timing constraints of the individual interactions – intervals of execution, mean, slack and jitter. In addition the possible repetitions and repetition intervals of the partiture are described.

Collecting context information

In order to collect the computing context information the partiture contains information on how the performance of the execution of the individual functions should be monitored at the nodes. The information describes how execution time of the functions is monitored, which allows each node to locally monitor the execution and later provide the performance information on the execution of functions. Based on the computing context history the nodes can also make predictions on the future executions of functions on a node and provide these estimations to the nodes that they interact with.

As is the case with computing context parameters the partiture also contains information what function should be used for the calculation and prediction of physical context parameters. As stated in the introduction we believe that formal mathematical analysis methods are not required to predict future values of context parameters with sufficient accuracy. In addition to being computationally intensive the generation of formal analysis methods requires good information on the physical domain and the creation of adaptive and context history exploiting systems is much more complex using these methods. Instead stochastic, heuristic, physical models or technical analysis tools are used for predicting behavior.

As the nodes monitor the context, add to the context history and make some decisions based on the context history they can also update accordingly the partiture of the computing scenario they are executing.

The architecture outlined above allows measuring different phenomenon according to predefined patterns and predicting the future values of context parameters based on past measurements of phenomenon. The predicted values are used either directly or indirectly in future computations to improve the efficiency and (user-observable) quality of the systems. According to some sources [7] that feature – the ability to anticipate the evolution of its surrounding environment is one characteristic of proactive systems, which the invisible computing systems are expected to be.

The conductor

To execute the partiture every node contains a conductor that can execute a partiture. The conductor is responsible for selecting the nodes that are going to execute the functions described in the partiture and delivering the information required for the execution to the nodes. The conductor sends the relevant part of the instantiated partiture to each node participating in the execution of the partiture. The segment of the partiture sent to the participating node contains information on what functions or services should be executed, timing constraints of the functions, messaging patterns, context collection and analysis. A conductor is also responsible for making agreements with conductors on other nodes to perform their part of the partiture.

Adaptation of the partiture

As the conductor reads the partiture and monitors progress, the context history can also be used to update the partiture itself with additional details of the execution flow. Here is one possible algorithm for modifying the partiture: 1) The scheduler monitors when the application sleeps (e.g. blocks on a semaphore). 2) The scheduler logs a trace of application state transitions (start, sleep, restart, end) and their times. 3) The trace is sent to the prediction engine. 4) The sleeps in multiple runs of the function are correlated and distinct sleeps are identified using pattern matching. 5) Once a consistent sleep pattern has been identified and verified with the given confidence the scheduling pattern for the function is split at the sleep step. To deal with the additional complication of functions with multiple sleep stages where the identification is unclear, the stages are disambiguated by adding an identifier to each potential blocking point. Another method for modifying the partiture is fine-tuning messaging patterns by observing the actual message flow.

The context history is thus used to evolve the problem description, allowing the original human author to use rough terms of intent and letting the system discover the details. It seems fitting to call this type of a rough partiture a *Jazz partiture*, given that the learning and specialization process is akin to improvisation. The partiture only describes what mechanisms should be used for adaptation; the adaptation process is controlled by each node locally.

A CASE STUDY

The claim that even quite thin embedded nodes are able to perform the predictions on context parameters is not unsubstantial, since in [3] as well as in the case study presented in this paper it is shown how even quite simple mathematical models suffice to predict the future values of context parameters, such as execution times of scheduled functions, with quite good results.

The system in the case study described in the following section follows a subset of the current architecture as it is a previous iteration of our design. However the concept of treating computation times as a stochastic process (and based on that predicting the future execution times) is strongly substantiated by the experiments.

Implementation

The concept of applying simple stochastic methods on predicting context information was experimented on a test platform equipped with a 25 MHz Arm7 microcontroller with 256KB of ROM and 32KB of RAM. In the core of the study was a stochastic planner that used the monitored execution times of scheduled functions to make adjustments to the scheduling pattern of the functions.

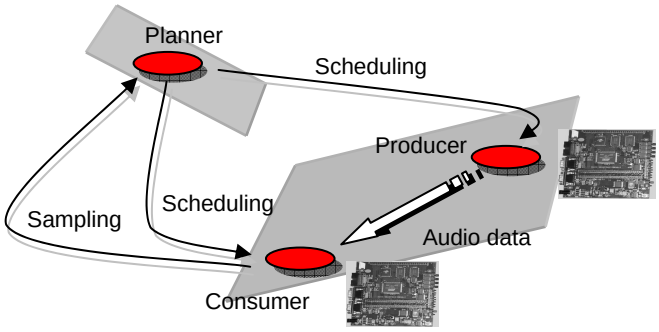

Figure 2. Scheme of the stochastic planner in action

It should be noted that the case study used an architecture where there conductor was distinctly separated from the nodes that executed the application specific functions.

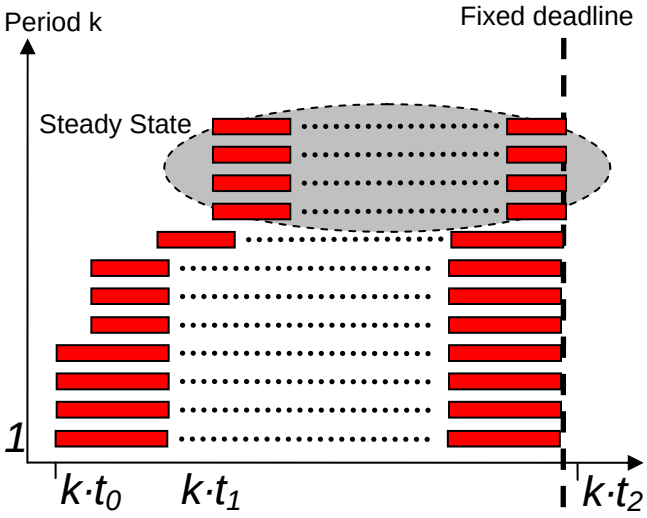

Figure 3. Adaptation of function executions from application supplied defaults to a near optimal steady state

Initially the planner uses an application supplied fixed schedule for scheduling the jobs on the worker nodes. The schedule is adjusted according to the information on the actual execution times received from worker nodes.

PERFORMANCE MEASUREMENTS

The working of the stochastic planner was estimated through sampling. A simple test method does 20000 multiplications. Starting with no context information the planner uses an application provided guess.

Once the planner receives samples from the measured execution times it uses the information with smoothing between each step. The calculation times include formatting and sending the reply message. The table below contains the relevant numbers. The estimate is produced by the live planner, while the mean and deviation have been calculated offline for reference from the raw measurements.

Step	Estimate 95% conf	Measured mean	Standard deviation	Confidence 95%	99%
1	339	337	1.7%	1.0	1.4
2	341	337	1.6%	1.0	1.4
3	346	337	1.8%	1.0	1.4

Figure 4. Time measurement and prediction of a CPU intensive task – times in milliseconds, 32 samples per iteration on embedded microcontroller board. The confidence number indicates the extra time allocated for jitter. Fixed point integer arithmetic rounds the number up slightly.

Since the low-level RTOS scheduler did not produce much jitter, the test was also executed on a PC running Windows XP with the XML communications middleware stack on top. Running without an underlying real-time scheduler introduces more uncertainty but the planner still deals with it correctly and produces a larger confidence allocation to cope with the increased jitter. As the CPU is faster a million multiplications is done each time. From a steady state the number of calculations is dropped to half. The table below shows how the planner adapts to the drop. The planner adapts to the larger jitter by padding the estimates.

Step	Estimate 99% conf	Measured mean	Standard deviation	Confidence 95%	99%
1	126	123	6.4%	1.9	2.5
2	124	120	14%	4.2	5.5
3	69	55	2.1%	2.8	3.7
4	58	55	2.9%	3.9	5.2

Figure 5, Time measurement on PC in milliseconds. After the steady state at step 2, the workload is cut in half and the estimate adapts to the new load.

CONCLUSION

Exploiting context histories is an increasingly important topic in the pervasive computing realm. There exists no obvious and widely accepted solution to this matter. In this paper we presented a general architecture which, by providing a description of computing scenarios used in a pervasive computing system, allows exploiting the observed context histories to improve the performance and optimization of the system, driven by context derived predictions of future behavior. In the case study we show that this approach is usable and provides benefits in a realistic embedded solution.

REFERENCES

1. Anagnostopoulos C., Mpougiouris P., Hadjiefthymiades S., Context awareness: Prediction intelligence in context-aware applications, Proceedings of the 6th international conference on Mobile data management MDM '05
2. Goldin D., Keil D., Wegner P.: An Interactive Viewpoint on the Role of UML, Ch. 15 in Unified Modeling Language: Systems Analysis, Design, and Development Issues, K. Siau and T. Halpin (Eds)., Hershey, PA: Idea Group Publishing, 2001, 250 – 264
3. Helander J., Sigurdsson S., Self-Tuning Planned Actions: Time to Make Real-Time SOAP Real, Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05) - Volume 00
4. Helander J., Deeply embedded XML communication: towards an interoperable and seamless world, Proceedings of the 5th ACM international conference on Embedded software, September 18-22, 2005, Jersey City, NJ, USA
5. Mamaysky H., Lo A. W. and Wang J. Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. Journal of Finance, 40(4), August 2000.
6. Meriste M., Helekivi J., Kelder T., Marandi A., Mõtus L., and Preden J.: Location Awareness of Information Agents, Springer Lecture Notes in Computer Science, Volume 3631 / 2005, p199 – 208
7. Motus L., Meriste K. and Dosch W., Time-awareness and Proactivity in Models of Interactive Computation, Electronic Notes in Theoretical Computer Science, Volume 141, Issue 5, 22 December 2005, Pages 69-95
8. Wegner P.: Interaction as a Basis for Empirical Computer Science, ACM Computer Surveys, 27, No. 5 (1995), 80 – 91
9. Wegner P.: Why Interaction is More Powerful than Algorithms, Comm. of ASM, 40, No 5, 80 – 91
10. Schlit B., Norman A., Want R. Context-Aware Computing Applications. In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, December 1994
11. Tran M., Hirsbrunner B., Courant M., A context-aware middleware for multimodal dialogue applications with context tracing, Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing MPAC '05