Annales Univ. Sci. Budapest., Sect. Comp. 31 (2009) 57-73

CONTEXT AWARENESS IN DISTRIBUTED COMPUTING SYSTEMS

J. Preden (Tallinn, Estonia)

J. Helander (Redmond, USA)

Abstract. Creating robust systems that deal with and consist of both the physical world and networked computing nodes is relevant to a large class of applications, such as consumer electronics, energy efficient homes, industrial automation, civic infrastructure and mechanical systems, such as airplanes and automobiles. Due to the importance of such cyber-physical systems to our everyday lives and economies, it is worthwile to investigate and develop new methodologies for programming such systems.

While significant advances in programming large numbers of communicating tiny computers - such as sensor networks - have been made in the past decade, the progress has not been as fast as it was expected. The slow advancement of these systems is due to several reasons. Sensor networks are in close interaction with the physical world, having to react to the stimuli received from the physical world, all the while the computers in these systems are interacting with each other. The systems are highly dynamic: new nodes join and leave the network and the timing of message transmission depends on the ever changing environment and the relative location of nodes. Thus the configuration, topology and timing of interactions cannot be known before the system is actually operational, rendering traditional static analysis methods insufficient. The computation in these systems depends on the current and past interactions and is therefore different from that of classical deterministic computing systems. Distributed computing systems consisting of a large number of nodes connected to the real world tend to exhibit emergent behaviour, which the current state of art is not able to predict, analyze and account for.

This article proposes that using context information in the computation may be part of the answer to dealing with the emergent behaviour and dynamicity of cyber-physical systems. The paper then presents some examples of what can be considered context information and how this context information can be used in the computation. Finally, the paper presents a general architecture for collecting and organizing context information.

1. Introduction

Distributed computing systems, consisting of small embedded computers have become increasingly important during the past decade. The conventional term "sensor networks" is used in a broad sense here, a device in a sensor network being a computing device equipped with a processor, some memory, a wireless communication interface, an autonomous power supply and possibly some sensors. So, a sensor network or more broadly a cyber-physical system, is a network of computing nodes that interact with the physical world, with each other and possibly some other external computing nodes. Software intensive devices are an example of devices that can be part of a cyber-physical system. In case of software intensive devices the functionality of the device can only be achieved with the combination of appropriate computing hardware and software, so it is natural that such devices should be able to communicate with each other to provide a better service. Sensor network nodes are sometimes also called motes or smart dust. The nodes in a sensor network (or in a cyber-physical system) form a MANET or a mobile (multihop) ad-hoc network, which is self-configurating network where each node can also perform the task of a router and where nodes can join and leave the network as they desire. Such flexibility means that the configuration of the network is not known at design time but instead the configuration is formed at run-time. A good example of a dynamically formed network is the deployment of motes from an airplane (which is realistic since motes are used in military monitoring applications) - the initial configuration of the network is to a great extent determined by the way the motes fall to the ground, it even cannot be expected that all the deployed motes will be operational or that the deployed network is fully connected. As the configuration of the network is also dynamically changing, the nodes must adapt to the changes in the configuration dynamically. The fundamental problems that arise in such open, unstructured, dynamic and heterogeneous networks are not solvable by conventional computer science methods [1]. Due to the high number of devices and the unpredictable interactions between the devices themselves and the interactions between the devices and the physical world these networks exhibit emergent behaviour, which we cannot predict or foresee, but instead we limited to watching the emergent behaviour develop as the system operates [1], [14]. The paper presents the authors' views on how computation in sensor devices might be organized and how context information as the authors understand it can be utilized in such a computation. The paper also presents some concrete examples of how context information can be used in computation and a general architecture that enables collecting and utilizing context information in a computation.

2. Computation in a sensor device

The task of a sensor network is to provide an ongoing service, instead of transforming a single static input into an output. This feature makes the systems inherently interactive in a computer theoretical sense, as defined by Wegner in [2], which in turn means that these systems cannot be modelled [3] 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 [4]. Context aware systems can be viewed as being a type of the interactive computing concept, where the outcome of current computations (or the behaviour of the system) is influenced by the current inputs and past operation and interactions of the system. In context aware systems the history of past interactions with the external world is collected to form a context that reflects the state of the outside world as perceived by the current computing node. To achieve context awareness the context information is supplemented with information on the state of the device itself.

However context information can also be viewed at the computation level. At this level the context information contains all information that is external to the current computation, while the context information also contains the information that is internal to the computation, e.g. the intermediate results of past computations. If a computation must transform a sensory input into an output the direct input to the computation is the current sensor value, the context information is everything external to the computation (e.g. inputs from other sensors) and the context information also includes the computation related information (past inputs, past intermediate results and past outputs).

3. Sensor data interpretation

As the name implies, the sensor network nodes must process sensory input. Whilst sensor data has been interpreted by digital systems for decades already this paper presents a different abstract view on how the processing of incoming sensor data can be viewed and how the dependencies in the data processing can be solved. The examples below attempt to illustrate the authors' view on context information using simple computations. The interpretation of an individual sensor input (for example the resistance of a thermistor) can be viewed as the processing of an input stream of data samples. The stream is characterized by a

timeset, which defines the time instances when the samples in the stream arrive for processing (i.e. when the sensor value is sampled). The individual data samples of the stream (that arrive at different time instances) may not be processed the same way since the processing of the stream may depend on previous values of data samples in the stream. In addition the processing of individual data samples in the stream may depend on the context (values of context parameters), which changes over time. For instance, the interpretation of the thermistor input may depend on the supply voltage when the resistance of the thermistor is measured with a voltage divider circuit. In this above case the voltage can be viewed as being part of the computing context for the thermistor signal input.

3.1. Interpreting local input

For thermistor the input stream is the ADC (Analog to Digital Converter) output from the ADC channel that is connected to the thermistor circuit and the output is the temperature. We use a notation similar to the Q-model [7] for denoting the interpretation of the data samples from a specific sensor

$$temperature_t = P_t(T_S * ADCOutput_{temp\_sensor}), \tag{1}$$

where $temperature_t$ is the result of the interpretation (the temperature estimation at time instance t), P_t is the processing of the stream, T_S is the stream timeset and $ADCOutput_{temp\_sensor}$ is the value of the data item (the output of the analog-digital converter) corresponding to a specific time instance.

In addition to the stream processing being context dependent the stream timeset can also be context dependent. From a real-time system developer's viewpoint the approach is quite natural - if the dynamics of the system is known then sampling can be done at a lower rate when the parameter value is not close to critical, but a higher rate is required when the parameter value is closer to the critical value. This approach is especially useful if the available power is limited and sensing is power intensive. The stream based interactive computation model seems to better suit such computing tasks than classical algorithm theory.

The time it takes to process each data sample is also context dependent - due to the fact that the interpreting functions may contain internal memory, the computation time may depend on the values of current and previous data samples (in some cases the intermediate results of a previous computation can be reused).

A quite good, yet trivial example of context dependent stream processing is evident in the case of a resistive humidity sensor, where the resistance of the sensor depends (non-linearly) of humidity and temperature. For devices with low processing power it is common to use a table based approach when a

non-linear conversion is required. The values in the table closest to the actual sensor value are looked up and interpolation is performed between these values (linearity of the function is assumed between the table values, which provides sufficient accuracy for most applications). Table 1 is a section of the sensor resistance table of a resistive humidity sensor H25K5A. The table rows contain the resistance corresponding to a specific humidity and the columns contain the resistance corresponding to a specific temperature at a specific humidity.

	Temperature	
Humidity	20	25
30%	3300	2500
35%	1800	1300
40%	840	630
45%	216	166

Table 1. Humidity sensor resistance

In order to convert the resistance of a humidity sensor into relative humidity four lookups must be made from the table based on the measured resistance and the current ambient temperature. When the four values have been acquired interpolations are performed to compute the relative humidity corresponding to the temperature and the resistance of the sensor. To obtain the resistance of the sensor the ADC reading must be interpreted first - if a voltage divider is used this interpretation depends of the supply voltage.

Thus it can be concluded that the interpretation of the humidity sensor input stream depends on interpretation results of two other streams - the temperature sensor input stream and the supply voltage input stream. The interpretation results of the voltage and temperature stream form the computing context for the interpretation of the humidity sensor input stream.

$$voltage_t = P_v(T_S * ADCOutput_{supply\_voltage}), \tag{2}$$

$$temperature_t = P_t(T_S * ADCOutput_{temp\_sensor}), \tag{3}$$

$$humidity_t = P_h(T_h * ADCOutput_{humidity\_sensor}), \tag{4}$$

$$P_h \overset{depends}{\rightarrow} P_t,$$

$$P_h \overset{depends}{\rightarrow} P_v.$$

It may seem there is a direct functional dependency between the relative humidity, the resistance of the humidity sensor, the resistance of the thermistor and the supply voltage at a specific time instance. If we view the inputs as streams and we want to perform ongoing interpretations of these streams we elect to abstract that dependency into P_h - the output of P_t is not a direct input to P_h, but instead P_h uses the output of P_t in its interpretation process. Only P_h can decide which output sample of P_t to use. The knowledge of which context parameters are relevant in processing of one stream is only present within the given processing task, hence the processing task must acquire this data independently from its stream input.

The timing dependencies between the streams are also important - if the interpretation of a stream is dependent of another stream (as is the case with the interpretation of the resistance of the humidity sensor) the time intervals that are used in one stream must match the intervals used in the other stream (interpreting the resistance of the humidity sensor based on the ambient temperature that was valid an hour ago is unlikely to be correct). Which means that if the interpretation of the resistance of the humidity sensor depends on the ambient temperature and the resistance of the humidity sensor is monitored using a given time constant, the time constant in the temperature stream must be the same. Another option is to predict the temperature value, but this can only be done within the temperature stream process as the necessary knowledge is only present there. Otherwise the interpretation of the input stream that represents humidity is not correct. If the time constant of one stream is changed, the time constant of the other stream may have to be changed also.

The result of stream processing can be viewed as a context parameter characterizing a specific parameter, such as the temperature of humidity, of the current context. If such interpretation is used, then instead of saying that the processing of one stream is dependent of another stream we can say that the processing of one stream depends on specific context parameter values (that correspond to the timeset of the stream). In the latter case we are also able to use (more abstract) context parameters instead of using outputs from another stream processing. A simple example to illustrate that point can be drawn, again, from the realm of sensor networks. In case of a node, which can either be powered from an external (stable) power supply or from batteries, the voltage need not be measured when an external power supply is used. The voltage sample stream can therefore be reduced by increasing its sampling interval quite significantly, while the other streams are operated normally, without sacrificing correctness.

3.2. Collecting context information from remote devices

In a distributed computing scenario where data for building context awareness is received from remote computing nodes, the received data must be accompanied with metadata such as a timestamp and the location so that the context information can be built correctly. Functionality similar to the channel information, as described in [7], can be used to manipulate the received data according to the metadata. If data received from different nodes (incoming data from each node can be viewed as a different stream of data) was originated at different time instances, the data from different streams can be conditioned (for example using interpolation, averaging, normal distribution rules or methods similar to technical analysis) to achieve values that can be mapped to a specific time instance. Same applies to the location - if information is received from several locations and must be mapped to one specific location for data fusion purposes the data must be conditioned accordingly so as to build context awareness locally.

4. Context depending scheduling

While context information - the state of the processed sensor inputs, temporal information and interactions - is important in the interpretation of the physical world as explained above, it is also relevant in the cyber side of the computing network. This is because the network itself has topological and temporal properties that change over time as well as due to changes in the environment and the state of programs and their workloads. The changes in the network must be taken into account in order to successfully produce the desired outcomes. For instance if the network latency increases and data is needed at given times at a destination, say a speaker, the data must consequently be sent earlier from a source, such as a disk CD player. Since the changes in network and processing delays can be caused by a multitude of reasons and their complex interactions, it is in practice not feasible to analytically predict the timing characteristics. Instead we employ the well known tool for dealing with chaos: statistics.

The idea here is to predict the future based on the past. While every stock trader is familiar with the disclaimer "past performance does not guarantee future results" it is often the case that recent observed activity is a good indicator of the overall state of the system, just as temperature is an indication of the state of a gas in a thermodynamical system. The general approach to modelling a chaotic changing system is a stochastic process. The stochastic process presented in section "Partiture case study" shows how even a very simple stochastic process based on the Gaussian distribution and a proportional blending of current mea-

surements to the prior states can adapt to changes to a system in a meaningful way. The distribution is used to calculate a confidence interval to produce a timing that will be sufficient for a given reliability (as expressed in a probability of success). A higher confidence requires a higher level of over-provisioning, where the level of resource reservation needed for the given reliability can be precisely quantified.

The stochastic approach is not only useful in quantifying the time or other resources needed for a given reliability but also in predicting when failure is getting too close for comfort. Thus a system does not only have to helplessly resort to trying to recover after a failure when the "can't fail" system inevitably fails anyway due to uncontrollable factors, but can actually go to plan B before the failure as a preventative measure. Thus the uncontrollable factors are not beyond reach of this approach but rather just another factor affecting the measurements that are fed into the stochastic process that produces the context awareness.

A stochastic process is also not beyond the reach of a tiny computing node. As is seen in [8] the inverse integral of the Gaussian (perhaps surprisingly) does not require complex calculations but can be done in terms of a simple binary search, one multiplication and one addition. This is a convenient property of the standard shape of the Gaussian, so the inverse integral can be pre-calculated into a table of fixed point integers.

The above makes the fact that context information can be very useful in a cyber-physical system, very evident. In the following paragraph we present a general architecture that enables dynamic collection of information required for building context awareness in a dynamic way.

5. The partiture

To systematically monitor and predict context parameters an architecture is used that relies on the usage of metadata to describe (among other things) the set of functions involved in a computing scenario, the interactions between functions within a node and the interactions between the nodes executing functions. A computing partiture - a collection of metadata about the computing scenario - is the source of information for the nodes executing the scenario. The description of a computing scenario contains the list of functions or services required to implement the scenario, the contextual information of these functions (such as location and temporal parameters) and the interaction patterns between the functions. In addition to describing a computing scenario the partiture allows describing how the context information required for a computing scenario is collected and used. On Figure 1 is a sample partiture for a simple one producer - one consumer

scenario. The location property has been specified for the producer since we are interested for the sensor data from a specific location (in the current sample from room II305).

```
<partiture name="samplePartiture">
 <function name="producerFunction">
  <node name="sensorProducer">
   <location>room_II_305</location>
  </node>
  <output>
   <consumer>sensorConsumer</consumer>
   <function>consumerFunction</function>
  </output>
 </function>
 <function name="consumerFunction">
  <node name="sensorConsumer">
  </node>
  <input>
   <producer>sensorProducer</producer>
   <function>producerFunction</function>
  </input>
 </function>
</partiture>
```

Figure 1. Cooperation between three components of an operating system

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 node names in the sample partiture are only placeholders - once the partiture is executed the names are replaced with names of real nodes in the network. The functions described in a partiture can run on one or more nodes depending of the details of the partiture and the avaibility of resources at the nodes in the given network.

The partiture describes the interactions (messaging patterns) between the functions including the timing constraints of the individual interactions - intervals of execution, duration and tolerance to jitter of the intervals called bars. The partiture also contains information on the possible repetitions and repetition intervals of the partiture.

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, i.e. a sampling schedule. The information describes how execution time of the functions is monitored, allowing a node to locally monitor the execution and later provide the performance information on the execution of functions and message delivery. Based on the context history the nodes can also make predictions on the future executions of functions on a node and provide these estimates to the nodes that they interact with. The partiture can also be modified according to the recorded context history if such behaviour has been prescribed by the designer of the system.

As in the case with computing context parameters the partiture also contains information on how the values of physical context parameters should be computed and what models (functions) should be used to predict future values of physical context parameters. We believe that formal mathematical analysis methods are often not required to predict future values of context parameters with sufficient accuracy. In addition to being computationally intensive (especially considering the limited processing power of the motes) the use 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 to predict behaviour.

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

Figure 2 depicts components residing on a node involved in the execution of a partiture. The elements on Figure 1 should be interpreted as follows: conductor - initiates a partiture, parses the partiture description, locates nodes that are able to execute functions described in the partiture, sends fragments of the partiture description to the selected nodes. The conductor is responsible for running the partiture, but the selection of the specific partiture to run may come from a human user, another node or the current node. The bookie deals with requests received from other conductors for partiture execution. Both partitures and functions may be assigned costs and the bookie is responsible for assigning costs to different functions and negotiating costs with other nodes. The task master is responsible for executing functions on the local node as specified by the partiture. The stochastic prediction module deals with observing the local computing context. The observation results are used by the task master to optimize the operation of the local node. Remote conductor is the conductor at another computing device. Overall system monitoring - monitors the network of computing devices and applications for overall performance.

The architecture outlined above allows measuring different phenomena according to predefined patterns and predicting the future values of context parameters based on past measurements of phenomena. The predicted values are

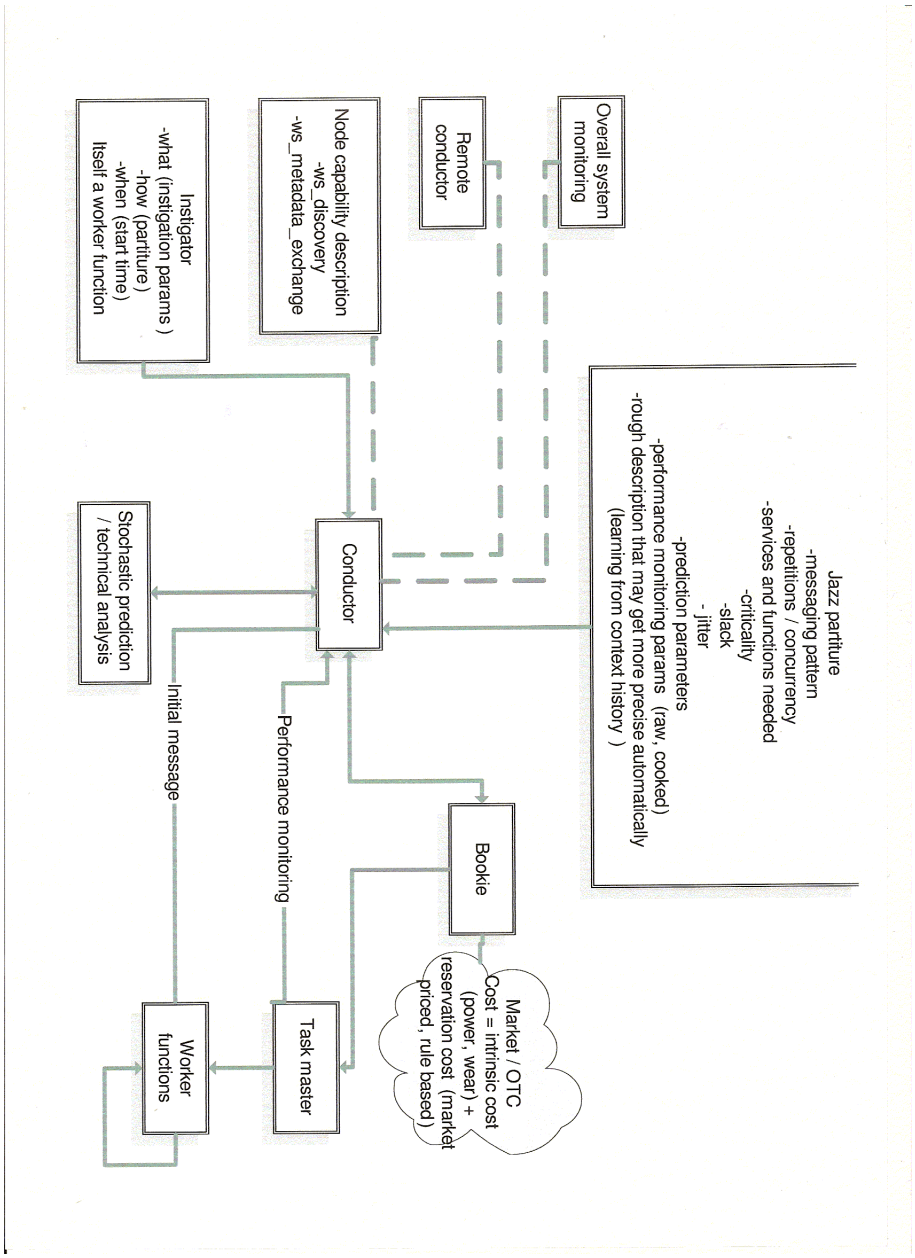

Figure 2. Cooperation between three components of an operating system

used either directly or indirectly in future computations to improve the efficiency and (user-observable) quality of systems. According to [9] these features - the ability to anticipate the evolution of its surrounding environment is one characteristic of proactive systems, which invisible computing systems are expected to be.

To execute the partiture a computing node includes a conductor that can execute the partiture. In a distributed application 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. In addition to the function and interaction information the conductor is also responsible for delivering the information on context parameter collection and context history utilization as described in the partiture. A conductor, with the help of the bookie, is also responsible for negotiating with conductors on other nodes to execute part of their partiture.

As the conductor reads the partiture and monitors progress, the context history is used to update the partiture itself with additional details of the execution flow. For instance the instrumentation of an executed function might reveal that there are two temporally distinct phases in the operation, such as an initial partiture prescribing reading data from a disk. The monitoring might then repeatedly observe that there is some computation leading to the read, then a long pause while the disk is seeking, followed by more computation. Based on the observation the disk read bar can be split into two separate bars. 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. The properties of specific interactions (such as timing) observable by a given computing node form the context history of the interaction. Based on the context history future operations and temporal properties of interactions can be adjusted. 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 claim that even quite thin embedded nodes are able to perform the predictions on context parameters is not unsubstantial, since in [8] 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 useful results.

The main overhead that the partiture approach introduces is from the discovery and setup phase, once the system is running and the nodes are performing the functions described in the partiture, there is no overhead introduced by the partiture architecture. Naturally monitoring the context parameters creates some overhead, but this can be mitigated by reduction of the sampling rate and buffering the samples. It is the a compromise between the quality control overhead and accuracy, much akin to the quality control of a manufacturing process. The trade-off itself can be analyzed statistically as is done in industrial quality control,

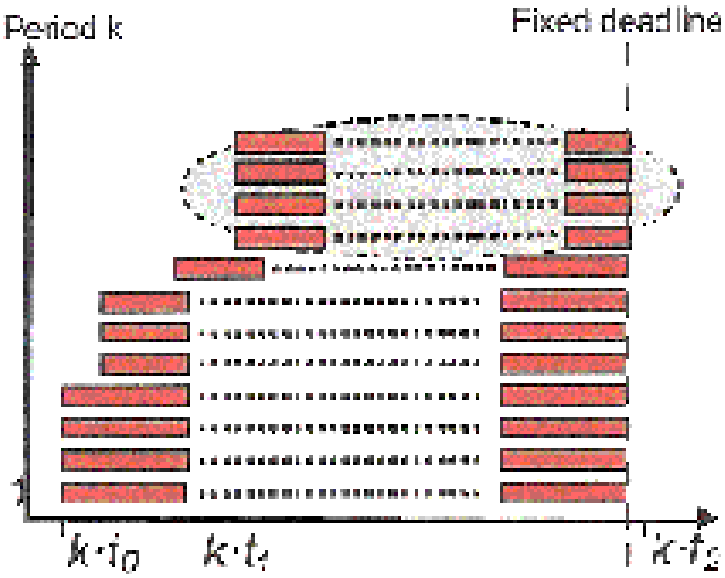

Figure 3. Cooperation between three components of an operating system

but is beyond the scope of this paper.

5.1. Partiture case study

The concept of applying simple stochastic methods to predicting context information was tested on a development platform equipped with a 25 MHz ARM7 microcontroller with 256KB of ROM and 32 KB of RAM running the Microsoft Invisible Computing Platform [15]. At the core of the study was a stochastic planner that used the monitored execution times of scheduled functions to make adjustments to the scheduling partiture of the functions.

It should be noted that the case study used a previous iteration of the architecture where the conductor was distinctly separated from the nodes that executed the application specific functions. Initially the planner uses a predetermined 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.

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

Table 2. 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 WindowsXP with the same XML communications middleware stack used in the previous measurement 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 larger jitter by padding the estimates.

Step	Estimate 95% 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

Table 3. 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

6. Related work

The current paper is related to two distinct topics: context awareness and distributed applications. While quite a substantial amount of work has been done in both fields, there is little work that combines these two areas and allows the collection of situation and context information in a distributed way, while also describing and implementing distributed applications. A superficial overview of related work in these areas is presented below.

Much work has been done in the area of user context identification. The aim in this direction is to provide a better service to the user by customizing the offered service according to the user's context. Anagnostopoulos et al. in [6] and Schilit et al. in [10] make some suggestions on how to determine and predict the user context. The work presented in [6] focuses on predicting context by creating related context entities and observing past and current values of context entities. In [13] a context aware middleware is presented, but the main purpose of the middleware is to capture the user context and act as a middleware between the user and a ubiquitous computing system. However in the current paper the context is viewed from the standpoint of a computing device - the context as it can be perceived by a single computation, a single computing device or a collection of devices. Work described in [11] proposes an architecture that allows building distributed applications in a distributed manner. The authors focus on sensor networks and they introduce the concept of roles to describe the tasks that nodes must perform in a network. Roles in the network are described in a high-level way and all nodes in the network are assumed to possess role information. Once a node is operational it tries to assume a role based on the role descriptions, its capabilities and its property values (current context). The network of nodes configures itself automatically and should start executing an application defined by the role descriptions. The nodes broadcast their property values (context information) to all the neighboring nodes, providing them with sufficient information for the selection of roles. The idea of a decentralized architecture is interesting, but the authors do not suggest how it would be possible to determine whether an application can run on a given network and what components are missing from a node, so it would be possible to run an application on a given network. It seems that the architecture assumes hierarchical data flow, which is reasonable in the context of sensor networks, but not in the context of cyber-physical systems where the network contains many heterogeneous devices and where the data flow is also heterogeneous. The approach also relies on hop count, which is reasonable to some extent in a multi-hop sensor network, but is of little importance in case of other types of networks.

The approach presented in [12] introduces a middleware that tries to cope with

the differences between different types of networks. The proposed middleware promises to meet the QoS requirements of applications running on a network by adapting the network configuration, overcoming the network delay and other similar issues characteristic to specific network.

7. Conclusion

This paper presented methods for collecting and analyzing context information and showed that it was possible to construct a system that could deal with a lot of chaos and uncertainty, but still make good choices and function properly. Fairly simple mathematical methods are sufficient to create a system that correctly adapts to its environment in both the cyber and physical domains. The concepts of the software framework and methodology presented in the article were used in a sensor network scenario and appear to contribute valid methodology for bringing the ubiquitous computing vision one step closer to reality. However a substantial amount of practical and theoretical work must be still done to develop these methods to a level where practical applications could be assembled using the described approach.

References

[1] **Milner R. and Stepney S.:** *Nanotechnology: Computer science opportunities and challenges,* Submission by the UK Research Committee to the Nanotechnology Working Group of the Royal Society and the Royal Academy of Engineering, August 2003.

[2] **Wegner P.,** Why interaction is more powerful than algorithms, *Comm. of ASM,* **40** (5) (1997), 80-91.

[3] **Goldin D., Keil D. and Wegner P.,** An interactive viewpoint on the role of UML, *Ch.15. in Unified modelig Language: Systems, analysis, design and development issues,* eds. K.Siau and T.Halpin, Idea Group Publishing, Hershey, PA, 2001, 250-264.

[4] **Preden J. and Helander J.,** Auto-adaption driven by observed context histories, *UbiComp Workshop ECHISE, 2006,* 36-41.

[5] **Wang A.Y.,** An FSM model for situation-aware mobile application software systems, *Proc. 42nd Annual Southeast Regional Conf., Huntsville, Alabama, USA, April 2-3, 2004,* ACM, 2004, 52-57.

[6] **Anagnostopoulos C., Mpoggiouris P. and Hadjiefthymiades S.,** Context awareness: Prediction intelligence in context-aware appliocations, *Proc. 6th International Conf. on Mobile Data Management MDM'05,* 137-141.

[7] **Motus L. and Rodd M.G.,** *Timing analysis of real-time software,* Elsevier Science, 1994.

[8] **Helander J. and Sigurdsson S.,** Self-tuning planned actions: Time to make real-time SOAP real, *Proc. of the Eighth IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing, Seattle, May 2005,* 80-89.

[9] **Meriste M., Helekivi J., Kelder T., Marandi A., Mtus L. and Preden J.,** Location awareness of information agents, LNCS **3631**, 2005, 230-242.

[10] **Schilit B., Adams N. and Want R.,** Context-aware computing applications, *Proc. Workshop on Mobile Computing Systems and Applications, 1994*, 85-90.

[11] **Frank C. and Rmer K.,** Algorithms for generic role assignment in wireless sensor networks, *Proc. 3rd Int. Conf. on Embedded Networked Sensor Systems, 2005,* 230-242.

[12] **Heinzelman W.B., Murphy A.L., Carvalho H.S. and Perillo M.A.,** Middlware to support sensor network applications, *IEEE Network,* **18** (1) (2004), 6-14.

[13] **Tran M.T., Hirsbrunner B. and Courant M.,** A context aware middleware for multimodial dialogue applications with context tracing, *Proc. 3rd Int. Workshop on Middleware for Pervasive and Ad-hoc Computing, 2005,* 1-8.

[14] **Stepney S. et al.,** Journeys in non-classical computation I: A grand challenge for computing research, *Int. J. of Parallel, Emergent and Distributed Systems,* **20** (1) (2005), 5-19.

[15] `http://research.microsoft.com/research/embeddedsystems/ews/`

J. Preden
Tallinn University of Technology
Ehitajate tee 5
19086 Tallinn, Estonia
`jurgo.preden@ttu.ee`

J. Helander
Microsoft Research
1 Microsoft Way
Redmond, WA 98052, USA
`jvh@microsoft.com`