Situation aware computing in distributed computing systems

Jürgo Preden[1], Johannes Helander[2]

[1] Tallinn University of Technology, Ehitajate tee 5,
19086 Tallinn, Estonia
jurgo.preden@ttu.ee
[2] Microsoft Research, 1 Microsoft Way,
Redmond, WA 98052, USA
jvh@microsoft.com

Abstract. Cyber-physical systems – distributed computing systems that combine computer networks with embedded devices interacting with the physical world are gaining importance and the need for such systems is growing. Significant advances have been made in programming large numbers of communicating tiny computers, such as sensor networks, in the past decade, however 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 and in addition the computers in these systems are in interaction with each other, as the configuration of the system is not known before the system is operational these interactions cannot be specified beforehand. The computation in these systems depends on the current and past interactions and is therefore different from that of classical computing systems. Distributed computing systems consisting of a large number of nodes connected to the real world also tend to exhibit emergent behaviour which the current state of the art is not able to predict. This article proposes that using situation information in the computation may be part of the answer for solving some of the problems described above. This paper presents some examples of what can be considered situation information and how this situation information can be used in the computation. The paper also presents a general architecture for collecting and organizing situation information.

Keywords: Situation aware computing, invisible computing.

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 be only 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 also called motes or smart dust. The nodes in a sensor network form a MANET or a mobile (multihop) ad-hoc network, which is self-configuring 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 can't be even 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 can't predict or foresee but instead we can only watch the emergent behaviour develop as the systems operates.

The paper presents the authors' views on how computation in sensor devices might be organized and how situation information as the authors understand it can be utilized in such a computation. The paper also presents some concrete examples of how situation information can be used in computation and a general architecture that enables to collect and utilize situation information in a computation is presented.

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 can't 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 interactive computing concepts, where that the outcome of current computations (or the behaviour of the system) is affected by the current inputs and the 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 situation awareness the context information is supplemented with information on the state of the device itself. The above describes how context and situation can be interpreted at the device level as it is also described in other related work [5],[6].

However situation information can be also viewed at the computation level. At this level the context information contains all information that is external to the current computation, the situation information contains also 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 situation information also has 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 view on how the processing of incoming sensor data can be viewed and how the dependencies of the data processing can be solved.

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 (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 as the processing of the stream may be dependent of previous values of data samples in the stream. In addition the processing of individual data samples in the stream may be dependent of the values of situation parameters which change over time. For example the interpretation of the thermistor input may depend of the supply voltage if the resistance of the thermistor is measured with a voltage divider circuit.

3.1 Interpreting local input

In case of a thermistor the input stream is the ADC output from the ADC channel that is connected to the thermistor circuit and the output is the temperature. We use a denotation similar to the Q-model [7] for denoting the interpretation of the data samples from a specific sensor:

$$temperature_t = P_t\left(T_S \times ADC\_Output_{temp\_sensor}\right)$$

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 ADC_{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 situation dependent the stream timeset can be also situation dependent. From a real-time system's developer 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 far from critical but at a higher rate 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 suit better for such computation tasks than classical algorithm theory.

The time it takes to process each data sample is also situation dependent – due to the fact that the interpreting functions may contain internal memory, the computation time may depend of 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 situation 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. In case of devices with low processing power it is common to use a table based approach if 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 satisfactory 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 resistivity corresponding to a specific humidity and the columns contain the resistivity corresponding to a specific temperature at a specific humidity.

Table 1. Humidity sensor resistance

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

To convert the resistance of a humidity sensor to 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 must be 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 of two other streams – the temperature sensor input stream and the supply voltage input stream.

$$voltage_t = P_t\left(T_S \times ADC\_Output_{\sup ply\_voltage}\right)$$

$$temperature_t = P_t\left(T_S \times ADC\_Output_{temp\_sensor}\right)$$

$$humidity = P_h\left(T_h \times ADC\_Output_{humidity\_sensor}\right)\left\{\begin{array}{l} P_h \xrightarrow{dependent} P_t \\ P_h \xrightarrow{dependent} P_v \end{array}\right\}$$

It can be said that 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. In a situation where we view the inputs as streams and we want to perform ongoing interpretations of these streams we elect to abstract that dependency into the 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. It is simpler to hide the details of what situation parameters the interpretation of a stream depends of and only input the necessary stream data to the interpretation. That also frees us from worrying about the functional dependencies when we view it at the abstract level of a stream.

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 for one stream must match the intervals used for the other stream (interpreting the resistance of the humidity sensor based on the ambient temperature that was valid an hour ago has a small chance of being correct). That means that if the interpretation of the resistance of the humidity sensor depends of the ambient temperature and the resistance of the humidity sensor is monitored with a given time constant the time constant used for the temperature stream must match that. Another option is to guess the temperature value but this can only be done by the temperature stream as it possesses information for that. If that is not the case the interpretation of the input stream that represents humidity is not correct. If the time constant for one stream is changed the time constant for the other stream may have to be changed also.

The result of stream processing can be viewed as a situation parameter, characterizing a specific parameter, such as the temperature or humidity, of the current situation. 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 of specific situation parameter values (that correspond to the timeset of the stream). 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 if an external power supply is used. The voltage stream can therefore increase its sampling interval quite significantly, whilst the other streams can continue operation as normal and correctness of computations is still ensured.

3.2 Collecting context information from remote devices

In case of 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 timestamp and location so that the context information could be built correctly. Functionality similar to the channel function, as described in [7] can be used to manipulate the received data according to the metadata – for example if data received from different nodes (incoming data from each node can be viewed as a different streams of data) originates from 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 for the location – if information is received from different locations and must be mapped to some specific location for data fusion purposes the data must be conditioned accordingly to build context awareness locally.

4 Context dependent scheduling

While situational 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 and due to changes to the environment and the state of programs and their workloads. The changes in the network must be taken into account 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 try to 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 [8] shows how even a very simple stochastic process based on the Gaussian distribution and a proportional blending of current measurements to the prior state 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 success probability). 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 useful not only 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 situational 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 to collect information required for building context awareness in a dynamic way.

5 The Partiture

To be able to systematically monitor and predict the context parameters an architecture must be 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 a computing scenario – is 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 of the details of the partiture and the availability 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, mean slack and jitter of the intervals. 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. The information describes how execution time of the functions is monitored, which allows a 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. 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 is the case with computing context parameters the partiture also contains information on how the values for physical context parameters should be computed and what models (functions) should be used to predict the future values of the physical context parameters. 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 (especially considering the limited processing power of the motes) 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 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 accordingly the partiture of

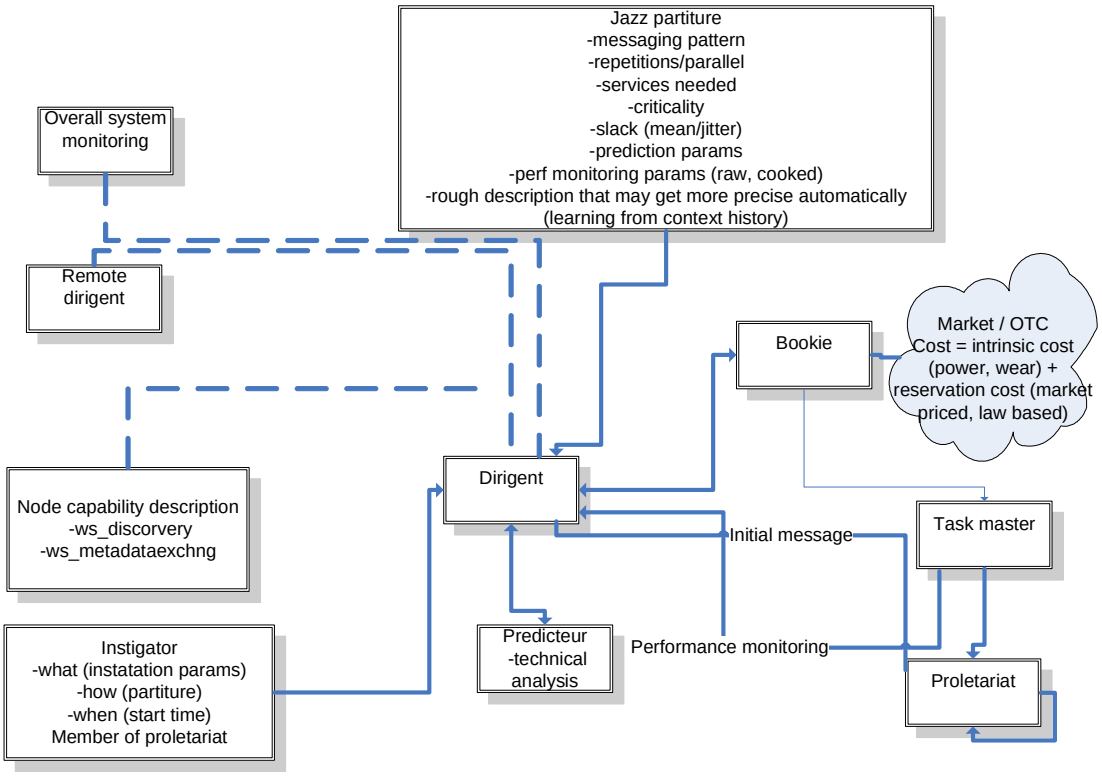

Figure 1. General architecture of an ubiquitous computing system

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 [9] these features – 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.

To execute the partiture a computing node contains a conductor that can execute a partiture. In case of 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 is also responsible for making agreements with conductors on other nodes to execute part of their partiture.

As the conductor reads the partiture and monitors progress, the context history is also 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 the initial partiture prescribing reading data from a disk, and the monitoring observing 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 note can be split into two separate operations. 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 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 good results.

7 Conclusion

The evolution of cyber-physical computing systems promises to make the vision of ubiquitous computing a reality some day. However big advances are required in computer science to enable the design and implementation of such systems with guaranteed results. Using context aware computing nodes is one method that will possibly advance the pervasive computing field. The article presented the authors' views on collecting and utilizing situation information. A general architecture was described that allows describe a distributed computing scenario that utilizes context information.

References

1. Milner R., Stepney S.: “Nanotechnology: Computer Science opportunities and challenges”, Submission by the UK Computing 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, No 5, 80 – 91
3. 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
4. Jürgo Preden, Johannes Helander, "Auto-adaptation Driven by Observed Context Histories", UbiComp workshop ECHISE 2006
5. A Wang Y.: An FSM model for situation-aware mobile application software systems, Proceedings of the 42nd Annual Southeast Regional Conference, 2004, Huntsville, Alabama, USA, April 2-3, 2004. ACM 2004, ISBN 1-58113-870-9
6. C 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
7. Motus L., Rodd M. G., Timing analysis of real-time software, Elsevier Science 1994
8. 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, Seattle, May 2005.
9. 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