Using very large arrays of intelligent sensors

R. M. Newman*

* School of Mathematical and Information Sciences, Coventry University, Coventry CV1 5FB, UK, R.M.Newman@coventry.ac.uk

ABSTRACT

Recently several authors have turned their attention to the design problems of very large arrays of intelligent sensors, such as would be suitable for inclusion as an integral part of a 'smart structure'. Such systems have architectural and data throughput advantages, but to be used effectively need to be designed so as to be 'controllerless' and 'decentralised'. This paper explores some of these design issues, and illustrates how the system properties of a solution to one of them can be described using an established formal method, the pi-calculus.

Keywords: Intelligent sensors, sensor arrays, formal specification, fault tolerance.

1 INTRODUCTION

In 1994 the author patented an 'intelligent vibration sensor' [1]. The device was essentially an accelerometer with integrated analog and digital processing capability as shown in Figure 1, which allowed a large proportion of the diagnostic functions of a complete vibration monitoring system to be localised at the accelerometer.

This development was a response to one of the major problems encountered with design of such systems at the time. The number of accelerometers typically used in a system had escalated steadily, causing severe data processing bottlenecks. An array of intelligent sensors provided a parallel machine, the processing capacity of which expanded to meet the number of sensors used. By localising signal processing and reduction the data communication bandwidth was also reduced.

At the time, the proposal was at the limits of available integration technology, and the patent was not utilised. With current fabrication processes it is feasible to integrate a micromachined accelerometer along with a substantial processing capacity onto a single chip making the intelligent sensor a realistic and very economical possibility. In fact, such a device could be sufficiently inexpensive to make possible the building of systems using much greater numbers than considered before, given the open ended processing capacity of such systems. This paper considers the properties and some of the potential of such systems

2 AN ARRAY SENSOR

An amendment to the original intelligent vibration sensor architecture is proposed, which consists of the addition of three further communications links and a dedicated communications processor (Figure 2). The device then becomes an 'intelligent array vibration sensor', which can be connected together in a square, two dimensional array of sensors of any size required (Figure 3), which could be integrated permanently into the fabric of the machine or vehicle under test, the main constraints being those of power consumption and the ability to calibrate sensors and handle their failure. Previous work by Gaura et. al. [2, 3] shows how the individual sensors might be calibrated and linearised using neural network methods.

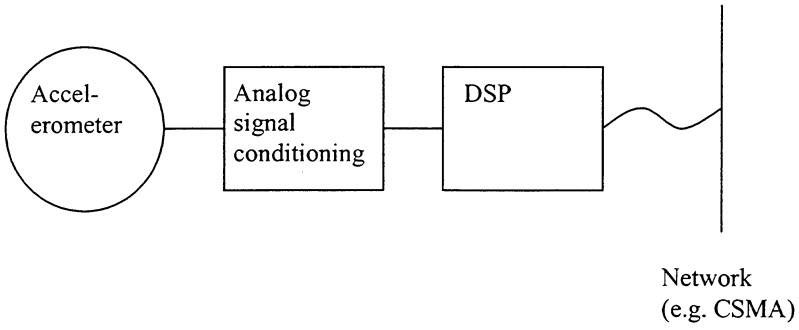

Figure 1: Intelligent Vibration Sensor

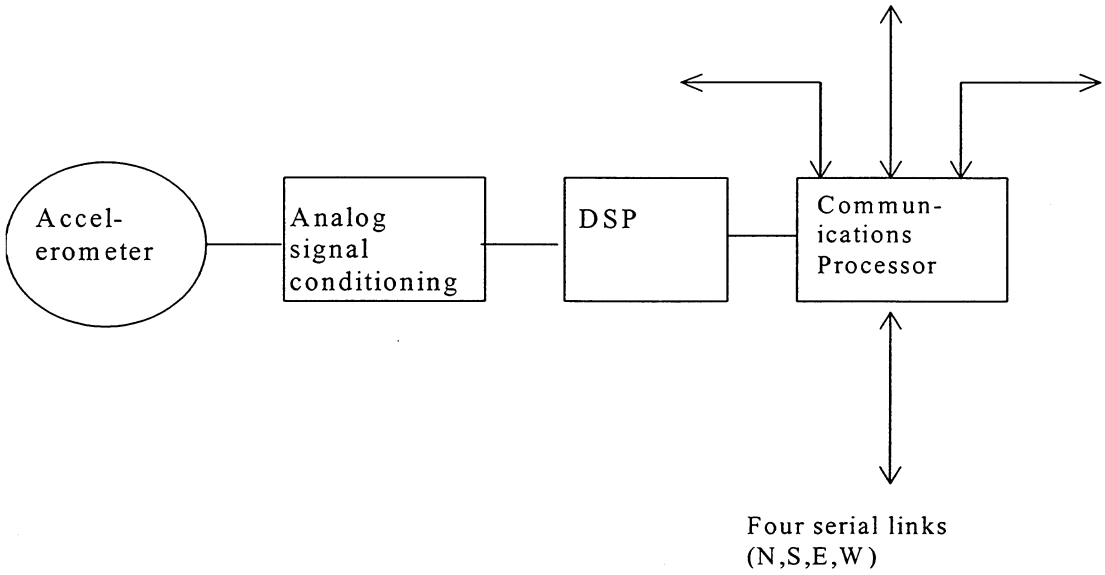

Figure 2: Intelligent Array Vibration Sensor (IAVS)

Such an array may be used in a number of ways.

- The large number of processors in the array may be used to enhance reliability by allowing for redundancy. Means would need to be developed for automatic detection of faults and allocation of tasks to adjacent sensors. This possibility is dealt with later in the paper.
- Different analysis tasks may be distributed to different processors within the array, allowing a single system to perform several analyses simultaneously.
- Recognition and analysis algorithms might be distributed to the array as a whole, using the complete system as a 'pattern recognition' device for different spatial vibration patterns. Such patterns are often diagnostic of particular mechanical faults, and could be detected by such means.

Systems are likely to use all three of these possibilities, providing them with specialised processing and an element of redundancy, providing powerful diagnostic procedures with graceful degradation in the case of individual sensor failure.

3 DECENTRALISED FAULT DETECTION

Several authors have discussed the issues of designing and using systems comprised of very large arrays of intelligent sensors[4, 5, 6]. If an array of sensors is to be built which can function independently of a master controller, then decentralized means of handling the functions of that controller must be found. One such function is detection of faulty sensors, and their removal from the array [7]. One convenient way to model such a system is as a collection of concurrent processes. This can lead simply to programmed solutions from an appropriate formal specification, in an appropriate programming language [8].

This next section develops a means for handling this, described in such a formalism π-calculus [9]. The details of this notation are not given here, but are clearly explained in the cited work.

Consider a rectangular array of smart sensors, $S_{0,0}$-$S_{n,m}$, a section of which is illustrated below.

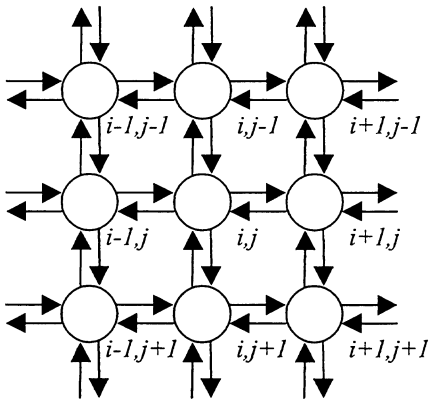

Figure 3: A sensor array

A sensor not at a border, S_{ij} is connected to its four neighbours, $S_{i-1,j}$,$S_{i+1,j}$,$S_{i,j-1}$,$S_{i,j+1}$. by four bi-directional links (or eight unidirectional ones). These are shown as direct physical links, although in practice some sort of multi-drop network might be more practicable, the links shown are the

logical links between a sensor and its neighbours. The processing for each sensor consists of several tasks, data collection *Data*, data analysis *Analysis*, sensor diagnostics *Diagnosis*, and data communication *Communications*. These tasks are described using the process algebra, π calculus. All definitions are parameterised on the co-ordinates of the sensor in the array, *i,j* where necessary.

First, the data collection task. Its precise nature is not of interest to this discussion. Generically, it will consist of reading a sequence of values from the sensor itself, collation of them into a packet of data, and transfer of that data to the analysis task. If we assume that the sampling from the sensor is internal to this task, the only externally visible action is the transmission of the packet, *packet*, so *Data* can be defined in π calculus as follows.

$$Data = new\ d(\bar{d} \mid !\, d.\overline{packet}.\bar{d})$$

This simply denotes that the task just sends a packet of data and repeats, indefinitely. Niceties of initialisation and termination are ignored.

For the purpose of this definition, we consider the analysis and diagnostic tasks together, since they use common data, and separating them will require replication of that data. Externally the combined task receives a packet of data from the sensor and also a larger packet, which will be called a frame of data, from the communications interface. It then performs the diagnostic or analytical action and sends data to the communications interface, then repeats. It is defined as follows.

$$Analysis = new\ a,b,c(\bar{a} \mid !\,(a.\bar{b}.packet.c.\overline{processed}.sync.\bar{a}) \mid !\,(b.frame.\bar{c}))$$

$$Diagnosis = new\ d(\bar{d} \mid !\,(d.\overline{sync}.\bar{d} + d.\overline{remove}))$$

The analysis produces a packet of processed data, which will include any reduced output required, and possibly the original sensor data, for the purposes of diagnosis or identification of other sensors. Before repeating the *AD* task, it is required to synchronise with the diagnostics.

The diagnostics task either produces a good diagnosis, in which case it does nothing but synchronise with the analysis and repeat, or detects a fault, when it sends a signal to remove this sensor from the array.

We now define a simple version of the communications task. This first version has no fault tolerance built in. Its function is simply to collect data from this sensor and the neighbours and to transmit this sensor's data to its neighbours.

$$Communications_{i,j} = IN_{i,j} \mid OUT_{i,j}$$

$$IN_{i,j} = new\ done,start\ (\overline{start} \mid !\,start.(EI_{i,j} \mid WI_{i,j} \mid SI_{i,j} \mid NI_{i,j} \mid INFIN))$$

$$EI_{i,j} = w_{i,j-1}.\overline{done}$$

$$WI_{i,j} = e_{i,j+1}.\overline{done}$$

$$SI_{i,j} = n_{i+1,j}.\overline{done}$$

$$NI_{i,j} = s_{i+1,j}.\overline{done}$$

$$INFIN = done.done.done.done.\overline{frame}.\overline{start}$$

$$OUT_{i,j} = new\ \ done,start\ (\overline{start} \mid !\,start.processed.(EO_{i,j} \mid WO_{i,j} \mid SO_{i,j} \mid NO_{i,j} \mid OUTFIN) + remove)$$

$$EO_{i,j} = \bar{e}_{i,j}.\overline{done}$$

$$WI_{i,j} = \bar{w}_{i,j}.\overline{done}$$

$$SI_{i,j} = \bar{s}_{i,j}.\overline{done}$$

$$NI_{i,j} = \bar{n}_{i,j}.\overline{done}$$

$$OUTFIN = done.done.done.done.\overline{start}$$

The *IN* process collects data from each of the four neighbours, in any order, then sends the resultant frame to the *Analysis* process, and repeats continuously. The *OUT* process collects a processed frame of data and transmits it to each of its neighbours, in a arbitrary order, and then repeats, unless it receives a remove action from *Diagnosis*.

Finally we assemble together the complete model of a single sensor, which is described by

$$Sensor_{i,j} = new\ packet,processed,remove,sync,\ frame(Data \mid Analysis \mid Diagnosis \mid Communications_{i,j})$$

It will be noticed that externally the only visible actions of $Sensor_{i,j}$ are its own four outputs, $n_{i,j}$, $s_{i,j}$, $e_{i,j}$ and $w_{i,j}$ and the four inputs $n_{i,j}$, $s_{i,j}$, $e_{i,j}$ and $w_{i,j}$.

Sensors signal a fault condition simply by turning off outgoing communications. The entire network is synchronous, with each cycle consisting of every sensor communicating in both directions with its neighbours. A sensor will therefore detect a fault in a neighbour by its failure to communicate in a cycle or, more specifically a second communication from another neighbour before all four communications in the cycle have been completed. This can be accomplished with the following modification to the *IN* process.

$$EI_{i,j} = w_{i,j-1}.\overline{done}.EIMON_{i,j-1}$$

$$EIMON_{i,j-1} = w_{i,j-1}.\overline{deadone} + stopmon$$

$$INFIN = done.done.done.done.$$
$$\overline{stopmon}.\overline{stopmon}.\overline{stopmon}.\overline{stopmon}.\overline{start}$$

The other three input processes are adapted in the same way. After receiving its communication, each process monitors for a second on the same link and, if found, notifies a dead process. These monitor processes are stopped only after all four communications in the cycle have been detected.

Having thus detected a failed sensor, it remains to reconfigure the array to remove that sensor. The simplest way to do this is simply to receive from the sensor on the other side of the failed one. The precise nature of the interpolation or other processes used to reconstruct the data for the failed sensor does not concern us here.

We can describe this reconfiguration by using the ability of the π-calculus to describe mobility of processes, by means or communication of names in the actions or messages. Rather than writing the names of the four neighbours in the equations, we allow them to be received as each instance of the *IN* process is started, as follows.

Here e', w', n', s' are the recalculated link addresses, bypassing the failed sensor. The monitor starts a new *IN* process with these recalculated links then ensures that other monitors die therefore avoiding replication of the new process. A monitor receiving a *die* action sends out a new one, to make sure all monitors die (or in the case of the last one, is simply deadlocked, since *die* is restricted, and there are no further processes with a complementary action).

$$IN_{i,j} = new \quad start((\overline{start(w_{i,j-1}, e_{i,j+1}, n_{i+1,j}, s_{i+1,j})} \mid$$
$$!start(e,w,n,s).new\ done, dead, stopmon, die.$$
$$(EI \mid WI \mid SI \mid NI \mid INFIN))$$

$$EI = e.\overline{done}.EIMON$$

$$EIMON = e..\overline{start(e',w',n',s')}.\overline{die} + stopmon + die.\overline{die}$$

$$INFIN = done.done.done.done.$$
$$\overline{stopmon}.\overline{stopmon}.\overline{stopmon}.\overline{stopmon}.$$
$$\overline{start(e,w,n,s)}$$

Thus, using a simple, fail safe protocol (in that sensors which simply fail to work are switched out of the array automatically), it is possible for the array to reconfigure itself in a distributed manner, with no central control over the process.

4 CONCLUSION

This paper has explored some of the issues concerning the systems design of large arrays of intelligent sensors, particularly those of decentralized control and fault handling in such arrays. A simple, decentralized, controllerless method of fault detection has been described, and developed in an established formalism, the π-calculus, in a simple and concise manner. Such a specification would enable the properties of such a system to be explored using the π-calculus or an associated modeling tool. It appears likely that some of the other issues discussed should be amenable to a similar treatment.

REFERENCES

[1] Newman, R.M. and Robinson, B. (1994)Intelligent Vibration Sensor, U.K Patent No GB 2251071, Grant Date 3 Aug 1994

[2] Gaura, E. Rider, R.J. Steele, N. (2000) Closed-loop neural network controlled accelerometer, Proceedings of the I. Mech. E, Part I, Journal of Systems and Control Engineering, vol. 214, no.I2, pp.129-138.

[3] Gaura, E. Rider, R.J. Steele, N. (2000). Developing smart micromachined transducers using feed-forward neural networks: a system identification and control perspective. The IEEE International Joint Conference on Neural Networks, IJCNN'2000, Proceedings, ISBN 0-7695-0619-4, Vol. IV, pp. 353-358, Como, Italy.

[4] M. Chu, H. Haussecker, F. Zhao, (2002) Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. Xerox Palo Alto Research Center Technical Report P2001-10113, May 2001.

[5] Estrin, D., Govindan, R., Heidemann, J., Kumar, S., (1999) Next Century Challenges: Scalable Coordination in Sensor Networks. Proc. ACM International Conference on Mobile Computing and Networking, (Mobicon 1999) pp. 263-270.

[6] Wang, A, Chandraskan, W (2002) Energy Efficient DSPs for Wireless sensor networks, IEEE Signal Processing magazine, July 2002, pp. 68-78.

[7] Jaikaeo, C., Srisathapornphat, C., Shen, C.-C., (2001) Diagnosis of Sensor Networks, Proc IEEE International Conference on Communications, Helsinki, Finland, June 11--14, 2001.

[8] Newman, R. M. (1998) The ClassiC Programming Language and Design of Synchronous Concurrent Object Oriented Languages, Journal of Systems Architecture, Elsevier Scientific, September 1998.

[9] Milner, R, (1999), Communicating and Mobile Systems: the Pi-Calculus, Cambridge University Press, Cambridge, England.7. Elson, J. ,Estrin, D. (2001)Time Synchronization for Wireless Sensor Networks, Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Parallel and Distributed Computing.