# Implementation of a framework to integrate sensors and controllers

Ryan D. Weir, Gourab Sen Gupta* and Donald G. Bailey

Institute of Information Sciences and Technology,
Massey University,
Palmerston North, New Zealand
E-mail: rdweir@gmail.com
E-mail: G.SenGupta@massey.ac.nz
E-mail: D.G.Bailey@massey.ac.nz
*Corresponding author

**Abstract:** A system architecture is proposed for integrating sensors, controllers, actuators and instrumentation within a common framework. The goal is to provide a flexible and scalable system. Extending the system, by adding additional components such as sensors or actuators, does not increase the overheads and is achieved seamlessly with minimal modification of the core controller programme. The architecture is generic and finds application in many areas such as home, office and factory automation, process and environmental monitoring, surveillance and robotics.

**Keywords:** sensor; controller; actuator; instrumentation; scalable system; data-centric architecture; framework.

**Reference** to this paper should be made as follows: Weir, R.D., Sen Gupta, G. and Bailey, D.G. (2007) 'Implementation of a framework to integrate sensors and controllers', *Int. J. Intelligent Systems Technologies and Applications*, Vol. 3, Nos. 1/2, pp.4–19.

**Biographical notes:** Ryan D. Weir received his BE in Information and Telecommunications Engineering from Massey University, New Zealand in 2004. Currently, he is undertaking his Masters at Massey University. His Masters thesis is on an integrated sensor and controller architecture for remote vehicle monitoring and control.

Gourab Sen Gupta received his BE (Electronics) from the University of Indore, India in 1982 and Masters of Electronics Engineering (MEE) from Philips International Institute, Eindhoven, Holland in 1984. After five years as a Software Engineer in Philips India, he joined Singapore Polytechnic in 1989 as a Senior Lecturer in the School of Electrical and Electronic Engineering. Currently, he is a Visiting Senior Lecturer at the Institute of Information Sciences and Technology, Massey University. His research interests include embedded systems, robotics, real-time vision processing, behaviour programming for multiagent collaboration and automated testing and measurement systems.

Donald G. Bailey has a BE (Hons) and PhD in Electrical and Electronic Engineering from the University of Canterbury, New Zealand. After spending two years applying image analysis techniques to the wool and paper industries within New Zealand, he spent 2.5 years as a Visiting Researcher at the Department of Electrical and Computer Engineering at the University of

California at Santa Barbara. In 1989, he returned to New Zealand as Director of the Image Analysis Unit at Massey University. In 1998, he moved to the Institute of Information Sciences and Technology where he is currently an Associate Professor and Leader of the Image and Signal Processing Research Group.

## 1 Introduction

Sensors are used in a variety of environments with varying platforms. In the real world, sensory data comes from multiple sensors of different modalities in either a distributed or centralised location. There are enormous challenges in collecting data, evaluating the information, performing decision making, formulating meaningful user displays and actuating alarms or other controls.

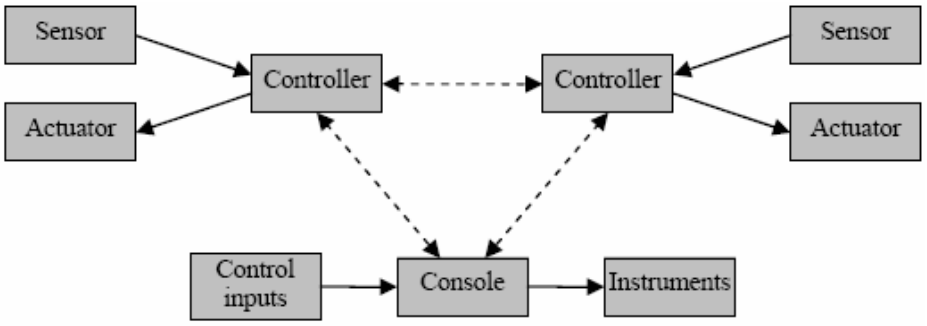
In the last decade, tremendous research efforts have been concentrated in the area of intelligent networked sensors and wireless network sensors addressing issues of self-organisation, scalability and data flow management (Estrin et al., 1999; Mainwaring et al., 2002; Sohrabi et al., 2004). A system architecture for monitoring habitat has been proposed Mainwaring et al. (2002). A tiered architecture comprising sensor nodes, connected to a base station via transit networks, internet connectivity to data services and user interface for data display has been developed. Their architecture does not incorporate any actuators. Moreover, the emphasis is on network architecture, rather than the controller architecture. Sohrabi et al. (2004) presents protocol architectures for scalable self-assembly of sensor networks. Estrin et al. (1999) makes a case for a data-centric network in which data is named by attributes. This approach decouples data from the sensor that produces it, allowing for more robust application design.

While a great deal of work has been done on sensor fusion, only recently are flexible architectures being developed that incorporate sensors, controllers and actuators within a scalable framework. The IEEE-1451 standard is a step in this direction (Lee et al., 2000). It is effectively an application protocol that facilitates the connection of, and communication between, the components of a control or instrumentation system. It supports features such as integrated data sheets, where key calibration parameters or tables can be read by the application, enabling the data to be correctly calibrated. These features enable a modular system incorporating sensors and controllers to be constructed seamlessly. This standard is still evolving and some aspects of it are still being finalised. However, the standard falls short of a framework in the sense that it provides only a standard mechanism for device interconnection. It fails to provide a unified platform for the development of complete control systems.

The CODGER system (Shafer et al., 1986), developed for large remote vehicles, addresses the overall architecture of a controller system. In that system, a manager keeps a list of which tasks are ready to execute and in its central loop selects one task to execute. Each task has a predetermined pattern of data that must be satisfied before it can be considered for execution. Tasks have access to data at various levels and each task is a separate, continuously running programme. Another approach is to separate the applications from the sensors and actuators by using a middleware layer (Branch et al., 2005). However, little has been done on designing a flexible architecture to integrate sensors, actuators, controllers and instrumentation within a single framework.

The traditional approach to control system design is usually controller-centric. The disadvantage of this architecture is that it needs to be reconfigured each time a change is made to the system components. The distributed model (Figure 1) is more flexible in this regard, as a change will result in modifying only one controller. However, such an architecture is still relatively inflexible to change made to the sensors or even instruments because everything is interconnected and a change to one component requires flow-on changes to many other components, including the controller. In a scalable system, it should be easy to add components, such as sensors and actuators, seamlessly as this happens frequently. In most systems, this would entail considerable system reconfiguration, resource remapping and possibly change of control methods.

**Figure 1** Distributed control system model




A number of different hardware platforms could be utilised for implementing a framework. A PC could be used, although embedded designs based on microcontrollers, FPGAs or PLCs may be more appropriate in many control applications. FPGA based systems are hard to programme, with a small change requiring the whole chip to be reconfigured. PLCs are primarily limited to industrial applications, and are quite rigid in their architecture making general application difficult. Traditional microcontroller based designs are tailor made for each application. A generic framework would simplify the design by giving better separation between the individual software components. A flexible, reconfigurable and scaleable framework would decrease development time and allow more time to be spent on designing the control algorithms, rather than implementing the system. This has motivated the authors to undertake this work and we believe that such a framework would be beneficial in many applications such as:

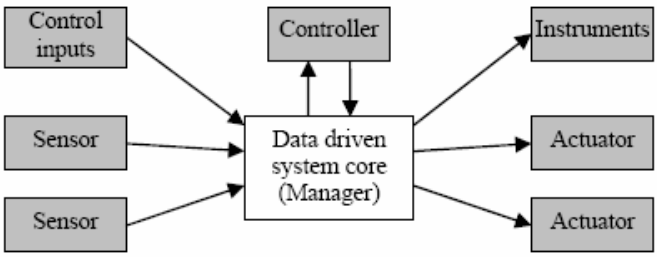
- *Home automation (e.g. Choi et al., 2005)*: temperature and light control; switching and monitoring of appliances; security; relaying status information and alerts to a mobile phone.
- *Environmental monitoring*: weather monitoring; water level and river flow; data logging and telemetry; alarm generation.
- *Industrial control*: shop floor automation; inventory tracking and control; fleet management (e.g. Cohen et al., 2005); process control and monitoring; autonomous robotics (e.g. Hohmann et al., 2003); machine vision.
- *Remote vehicle control (e.g. Armstrong et al., 2000)*: hazardous environments; disaster recovery; surveillance; telerobotics.
- *Security systems*: monitoring; intruder detection; access authorisation.

In this paper, we present the details of an integrated sensor and controller framework and show that this architecture is flexible, scalable and extensible. The next section describes the system architecture. A generic data-driven approach is analysed and the important characteristics of the system components are identified. A configuration mechanism is described that enables the operation of complex control systems to be easily set up. Section 3 discusses a range of implementation issues. We have taken an object oriented approach to the implementation on a PC based platform.

## 2 System architecture

To overcome the problems with a controller-centric architecture, we propose a data driven model (Figure 2). Here the controller is removed from the system core and becomes just another component. This approach explicitly uses the data as the link between the various components. In this way, the sensors, actuators, controllers and instrumentation can all be designed and developed independently. This separation of the system components makes modifying or adding further processing blocks, for example adding new features when required, quite straightforward.

**Figure 2** Data driven architecture




### *2.1 System core*

In the context of a data-driven architecture, it is observed that every device may be thought of as either a data source or a data sink. Sensors and control inputs are both data sources. Actuators and instruments are data sinks. Control operations are both sinks and sources, with the sink corresponding to the input to the controller and the control output being another data source. This is similar to the 'pipes and filters' architecture style (Meunier, 1995), where the pipes carry the information to be processed to the filters, then data are sent on via other pipes to be processed by other filters. The core of the system is therefore a data manager that coordinates the data received from the sources and ensures that it gets passed to the corresponding sinks.

To maintain maximum flexibility, it is important to keep the operations performed on the data separate from the sensing and actuation processes. This may be accomplished by facilitating communication between the various modules through a set of data buffers. These buffers may be thought of as a little like pigeon-holes in a filing clerk's office. In fact, Hohmann et al. (2003) implement a distributed processing controller for a robot with similar buffers using mailboxes. A separate buffer or data slot, is required for each data source, although a single data source may have multiple output data slots. These data slots provide the interface between the data sources and any associated data sinks.

## 2.2 System component characteristics

Before continuing further on the operation of the system, it is necessary to more formally describe the characteristics of the system components that will be interacting with the system core.

Sensors gather information from the real world, to be used by the system. These sensors could be on different buses (RS232, RS485, IEEE-488/HP-IB, $I^2C$, VXI/PXI, USB, ethernet, IEEE-1394, IEEE-1451, etc.) with corresponding physical and application layer protocols. Each sensor therefore requires a driver that interfaces between the sensor hardware and the system core. Streaming based drivers would automatically place the sensor data in the corresponding data slot. The framework also supports polled sensors, by using a timer (or other event) to trigger data gathering.

When the data comes in from the sensors, it is not always in the form that can be used readily. In these situations, it is necessary to convert the data into a more suitable format. It is important to keep data generation and conversion as separate processes for flexibility. The data converter effectively acts as a sink on the sensor output and as a source of newly formatted data. Data converters may be used for providing data compression (e.g. of speech or video signals). They also allow sensor fusion by combining inputs from multiple different sensors and selecting the sensor most appropriate to a particular activity or situation for providing the output. An example would be the integration of a pressure based altimeter, a GPS receiver and an ultrasonic height measurement system. When flying close to the ground or landing, the ultrasonic sensor may be selected over the other sensors.

Control inputs allow an operator the means of manipulating the system. Many of these controls will be sensor based (e.g. switches, potentiometers, joysticks, etc.) and may be considered in exactly the same way as sensor inputs. A computer based implementation also allows virtual controls to be implemented in the application GUI. Both physical and virtual controls would provide their control data through corresponding data slots. The data slot mechanism therefore allows physical inputs and virtual inputs to be interchanged without affecting the rest of the system. This flexibility also allows simulated (or prerecorded) sensor inputs to be input from a file for system testing or debugging.

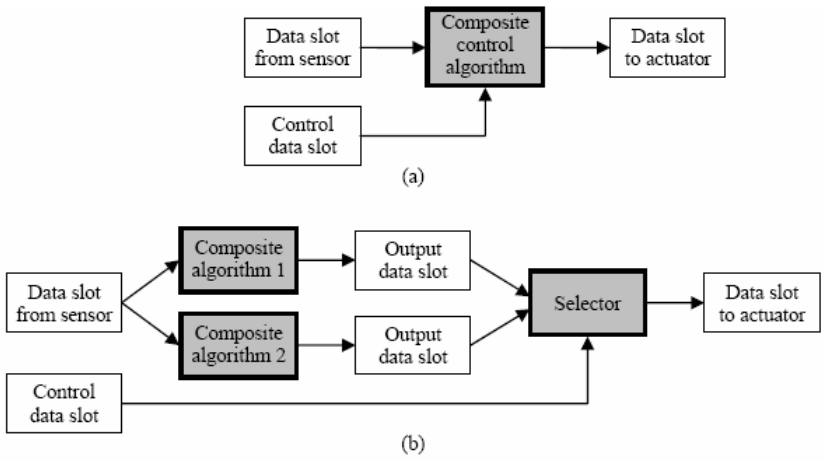
The heart of any control system is obviously the controller. In this framework, the controller is no longer directly connected to the sensors and actuators, but via the data slots. The controller may therefore be thought of as a form of data converter, converting the input data from the sensors (from the data slots) into the form required by the actuators (stored into data slots).

An important advantage of this architecture is that the controller need not be changed when the sensors or actuators change. Similarly, it is relatively easy to change the control algorithm for a given set of sensors and actuators. Indeed the controller behaviour can be selected on the fly based on circumstances (other sensor inputs). This could be implemented as a composite algorithm, using a control data slot to directly modify the operation (see Figure 3(a)). This structure would be most appropriate for adaptive control algorithms where only the control parameters were adjusted. Alternatively two or more control algorithms could be implemented independently, with the control outputs going to separate data slots as shown in Figure 3(b). A third control algorithm (a simple selector in this case) would then route the corresponding data to the data slot actually used to control the actuator. This structure is best suited to the situation where the

controllers use completely different algorithms (or even different sensor inputs). By keeping the higher level control separate from the lower level controllers, hierarchical control schemes may be developed and implemented relatively easily and efficiently. Such hierarchical controllers are suited for implementing autonomous navigation (Sen Gupta et al., 2004).

Actuators take the data from a data slot and convert it to some physical realisation. They would therefore require device drivers in a similar way to sensors. Data converters may also be required to convert the data into the correct format for the actuators. Physical instruments could also be incorporated in exactly the same way as actuators. A PC based system can also support virtual instruments, implemented within the user interface.

**Figure 3** Composite control architectures. (a) A single composite control algorithm and (b) hierarchical control of independent control algorithms




## *2.3 System operation*

The key to the operation of the framework is the mechanism by which data is passed from one processing block to another via the data slots. During initialisation (described in detail below), all processes that make use of input data are registered with the corresponding data slots as data sinks of that particular slot. The operation of the data slots is illustrated in Figure 4.

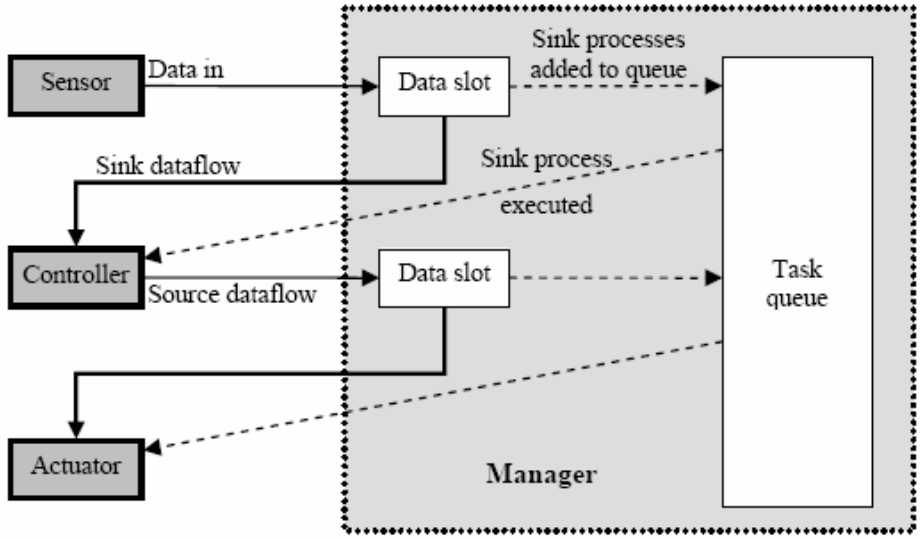
As new data becomes available from a source (either because a sensor reading is available, a control input has changed or a control algorithm has produced a new output) the new data is placed into the corresponding data slot. The source then notifies the data manager which adds the sink processes associated with that data slot to a task queue. Multiple task queues have been used to ensure that key data is processed in a timely manner. The priority of each task queue is set depending upon the type of tasks handled by the queue. When a sink process is executed, it reads the data from its input data slot(s) for processing. This may result in new data being produced (if the sink process was a controller or data converter).

Polled sensors require a periodic trigger to acquire data. Polling may also be accomplished via the data slot mechanism by configuring a timer to periodically update a data slot when it times out. By registering the sensor process with the timer data slot,

it would then be periodically queued for execution triggering it to poll for input data. In fact any input condition could be used to trigger polling. A data converter could monitor one or more data slots, and produce an output when a particular trigger condition is established, initiating an event-triggered process.

Keeping data sources and data processors separate in this way enhances flexibility and greatly simplifies reconfiguration. It enables a complex control system to be broken into a number of simpler processes or units and each of these processes can be designed and analysed separately. This is similar to the approach taken by Cohen et al. (2005) in keeping event generation separate from event processing. It also allows a single operation to be modified without having to rebuild the complete system. Data dependencies between the individual processes are implicit through the data slots.

**Figure 4** Data driven mechanism




## *2.4 Data communication*

In many applications, a distributed or networked control system is appropriate. Such a system would consist of two or more nodes connected by a set of communications channels. Each node in the network would independently implement the architecture described in the previous sections. The communication channel could be wired or wireless and based on any number of different protocols. In any distributed control system, care must be taken to ensure that communication delays do not result in system instability (Marti et al., 2004). This is particularly important if the communication link may have significant latency, jitter or lost data. It is important to minimise the use of such communication links for time critical control tasks. If necessary, this may be achieved by using a local controller.

The role of the communications network is to link the separate nodes into a common system. It achieves this at the data slot level by creating links between selected data slots. A data transmission process is registered as a data sink on all of the data slots that require transmission to another node. Each communication data slot is supplied with a data

ID used to identify the data to the receiver. When new data is available, the transmit task is queued. When executed, it sends the data to the receiver node where the data ID is used to determine the corresponding data slot at the destination. This will then activate any dependent data sinks at the destination node. The purpose of the communication link therefore is to keep the corresponding data slots on the source and destination nodes synchronised.

In communication intensive applications, such as those involving video communication, channel bandwidth may be a significant limitation. Where bandwidth is an issue, it may be important to make the system adapt to the available channel capacity. This would require the communications process to monitor throughput and latency and output key channel usage statistics to a data slot. A communication controller could then be set up as a sink on this data slot and then alter the data rate (e.g. video frame rate or resolution) based on available channel capacity. Such a system would degrade gracefully by optimising its performance to available resources rather than giving unpredictable performance when there is insufficient capacity.

## *2.5 System initialisation*

The configuration of any particular control system is determined by the linkages between the processes (sinks and sources) and data slots. This configuration is specified by a configuration file which is loaded at system initialisation. Each node within a distributed network system has its own configuration file.

The configuration file is a text file that contains a series of keyword and value pairs. There are two sections to the data file: the first defines the data slots and the second defines the processes. Figure 5 shows the configuration and associated configuration files for a simple remote sensor and actuator controlled from a base station.

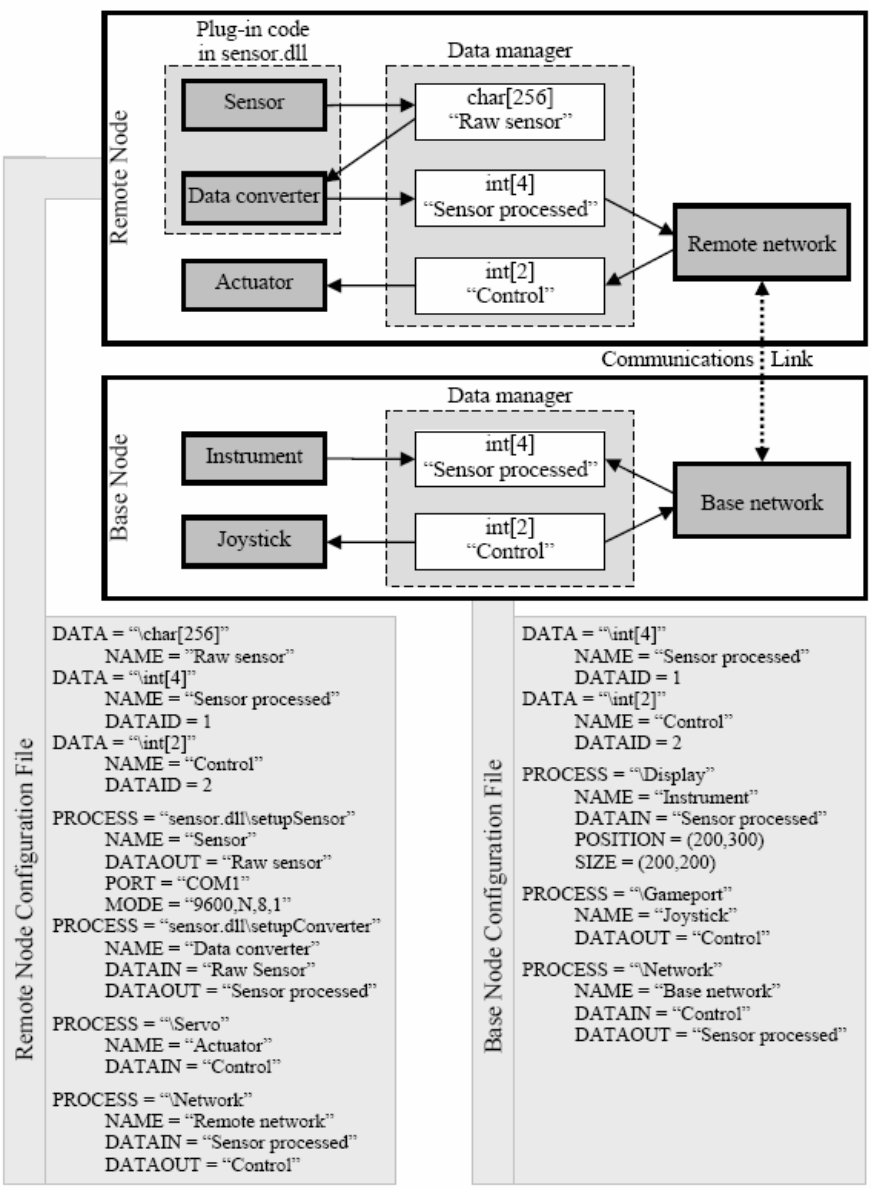
The ‘DATA’ keyword specifies the type and size of data contained in a data slot. The data slot name must be unique as it is used to identify the data buffer to any processes that may use data from or produce data for that slot. Custom data slots allow more complex data structures to be implemented. In Figure 5, there are three data slots created in the remote node and two in the base node. The ‘Raw sensor’ data slot is a string of up to 256 characters. The ‘Sensor processed’ data slots (one at each node of the communications link) are arrays of four integers, while the ‘Control’ slots are two integer arrays.

The ‘PROCESS’ keyword defines a data source or sink and specifies the procedure that will implement the process. This may either be a built-in process, or contained within a DLL. By making use of DLL plug-ins, whenever a new component is added, the code for implementing that component can be easily provided without having to rebuild the complete system. Data sources use the ‘DATAOUT’ keyword to specify which data slot the output data is to be written to. If a process is a data sink, the ‘DATAIN’ keyword indicates where the data is to come from. This registers the process as a sink on the corresponding data slot. Many of the remaining attributes are process dependent and may specify additional setup information required by the process, such as what port or protocol to use to gather the data from a sensor or the location of a virtual instrument on the screen. Where necessary, a process may use additional keywords to distinguish between multiple inputs and outputs. Processes may also specify a task queue to use instead of the default queue. This will be described in more detail in the next section.

## 3 Implementation

The basic core of the system has been implemented. Basic processors have been developed for connecting with $I^2C$ and RS232 based devices. A TCP/IP based communications module has been implemented to allow testing of distributed controllers and remote monitoring and control. A minimal set of display components has been developed to test the functionality of the framework.

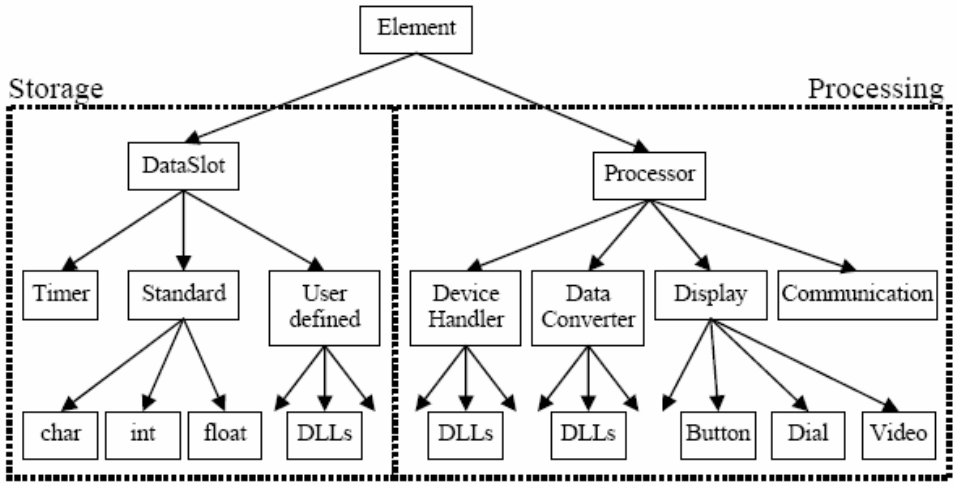
**Figure 5** Flow of information and configuration files in a simplified example. This shows manual operation of a remote actuator from a base node, using sensor data obtained from the remote node



### *3.1 Class structure for a PC based implementation*

For a PC based implementation, an object-oriented approach using C++ was taken to build the framework. This allowed for easy implementation of any given control system because the base classes provide the functionality of the framework. In a new application, the specific processor blocks may be derived from existing classes. The core functionality is inherited and does not need to be reimplemented; only the new features for a particular application need to be added. Figure 6 shows the class hierarchy.

**Figure 6** C++ class hierarchy showing how the various entities of the framework relate to one another




The main flexibility of the framework lies in the configuration mechanism. To facilitate parsing of the configuration file, all data slots and processes are derived from a common `Element` class. This contains methods which interpret the keyword and value pairs and enables the control system for any application to be constructed.

The `DataSlot` class performs the data related operations such as signalling when new data is available and queuing associated data sink processes. Each different type of data is derived from the `DataSlot` class and incorporates the code required for allocation of the storage buffer and enables the type and size checking to be performed by associated processes. The architecture is easily extended by deriving new data types from an existing class.

The `Processor` base class contains the aspects of the framework that relate to process execution. The `Processor` class is further extended depending on the type of processing performed. `DeviceHandler`s are `Processor`s that interface with a physical device, whether a sensor or actuator. `Display` components are those which have a visual representation on the computer monitor. These are used to implement soft controls and virtual instruments. `DataConverter`s implement data conversion and controller algorithms. The `Communication` class is the base class for handling communications between nodes of a networked system. It implements the high level packet handling, with derived classes handling the particular requirements of the communications channel.

Most of the `Processor` classes are implemented as DLL plug-ins. The value associated with the 'PROCESSOR' keyword specifies the DLL containing the implementation and the constructor of the corresponding `Processor` object. This is illustrated in Figure 5 where the 'Sensor' and 'Data converter' processes are loaded from sensor.dll.

### *3.2 Performance issues*

The performance of the data-driven system ultimately depends on the production and consumption of messages, which is related to the speed of the CPU, the number of sensors and the complexity of any of the processes that manipulate the data. There is an inevitable overhead associated with keeping each of the processes separate and forcing all communication between the components of the system to take place via the data slots. However this is a small price to pay for the considerable increase in flexibility and component reusability.

As each data source may trigger multiple processes, the task queues may potentially grow without limit, as each task to complete may queue multiple additional tasks. It is important that under overload situations this growth is bounded and there is a graceful degradation in performance. In our system, such scalability is ensured by only allowing each process to be on a task queue once at any one time. If the contents of a data slot changes before the sink process is executed, the sink process maintains it place in the queue. When it eventually executes, it will process only the most recent data. In applications where every sample is important, such as data-logging, it is necessary to implement the data slot as a FIFO buffer rather than as a simple data element. Then, when the sink process is called, it can process all of the samples that have arrived since it was last called.

Implicit in using multiple queues is to process each queue in a separate thread. It is possible to have two or more threads servicing each queue to ensure that one slow or computationally intensive process will not necessarily hold up the others within the same queue. Time critical processes may use their own separate task queues and run with higher priority. Another advantage of using a multithreading environment is that it also allows a watchdog process to be implemented. The watchdog process can then abort any process that takes excessive time to respond. This will ensure that in a complex system, failure of one part of the system (or one sensor) will not severely impact other, unrelated, parts of the system.

### *3.3 Microcontroller implementation*

To be a truly portable framework, it needs to be able to be implemented on other hardware platforms such as PDA's and microcontrollers. On a microcontroller code and data size is much more limited, making an explicit object oriented approach harder to use. The key principle is to make the system data-driven, by separating each of the processing steps: interfacing to sensors, data conversion, control, actuation and linking them through shared memory. The configuration process, rather than being based on parsing a file is more likely to be linked directly or based on software interrupts. The data slots may be declared variables and configuration becomes another code module that ties all of the independent processing modules together.

## 4 Example applications

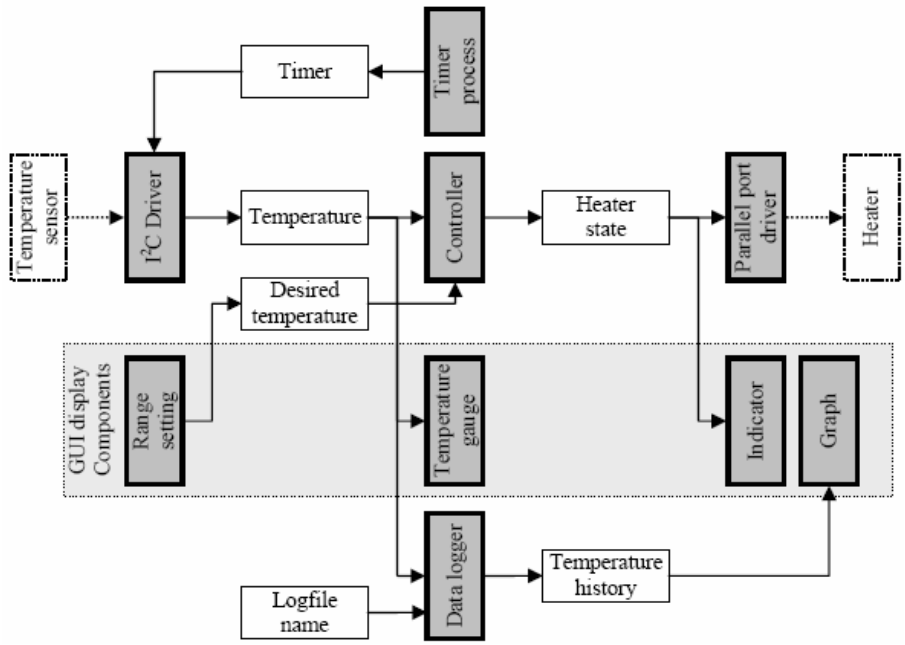
This framework will be demonstrated through two applications, brewery temperature control and robot soccer.

### *4.1 Brewery temperature control*

During brewing, the brewing vat must be kept within a certain temperature range to ensure optimal conditions for fermentation. Departure from the optimal temperature may result in reduced quality or process yield.

Figure 7 shows the components of the system used in this application. The temperature is sensed using an $I^2C$ temperature chip. The temperature is polled every two seconds. A simple on–off controller with hysteresis has been implemented, with the heating unit switched via the parallel port. The temperature is logged so a graph can display the temperature over a period of time. Other user interface components show the heater state, current temperature and allow the temperature range to be entered.

**Figure 7** Brewery control application showing the components and how they are linked together



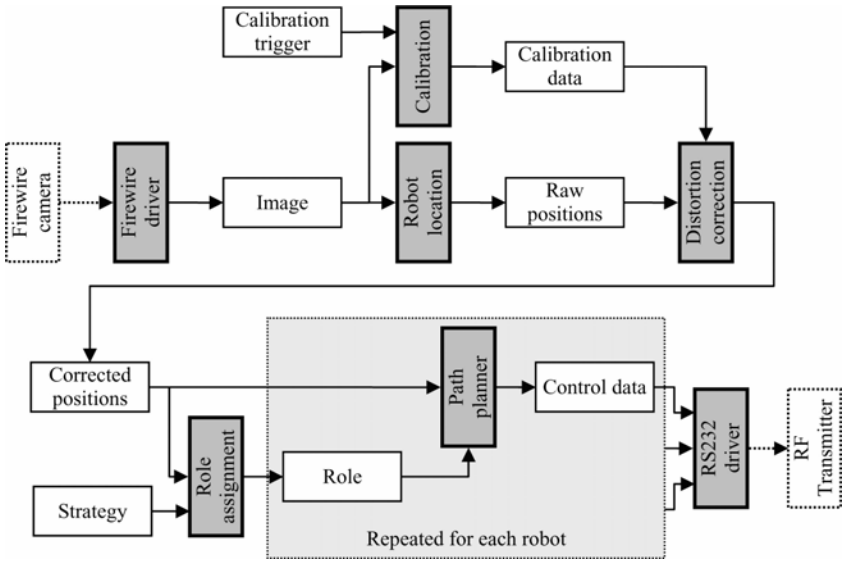

### *4.2 Robot soccer*

We are currently applying the framework to robot soccer. Figure 8 shows our proposed control structure. A firewire camera continuously captures an image of the soccer field. During initialisation, the image is processed to extract calibration data that enables the robot position data to be corrected for lens distortion, perspective distortion and parallax (Bailey and Sen Gupta, 2004). During game play, the 'Robot Location' process locates each robot from the captured image. These raw positions are then corrected for the

various distortions to give the true location in field coordinates. The 'Role Assignment' process determines the role of each of the controlled robots based on the selected game strategy. A separate 'Path Planner' process is instantiated for each robot, which uses its role information and the locations of the other robots to determine where it should be positioned. The derived control data is transmitted to the robots via an RF transmitter attached to the serial port.

This structure demonstrates several features of the framework. The controller is split into a two level hierarchy (Sen Gupta et al., 2004) by separating the role assignment and path planning. The robot soccer application requires that the complete control loop be executed for each image that is captured, typically 60 times per sec. This will test the responsiveness of the framework and demonstrate that it can be applied to time-critical applications.

**Figure 8** Control flow for the robot soccer application (for clarity, the user interface components have been omitted)




## 5 Discussion

In this paper, we have discussed the implementation of a data-centric framework that integrates sensors, controllers, actuators and instrumentation. The key principle behind the architecture of the framework is to separate each step of the control into independent processes, which communicate via data slots. This reduces the coupling between the components, enabling better code reuse, faster changes and easier debugging. Components are registered in the framework using a configuration file which defines the data slots and how the processes connect between them. Changes can be made to the system by modifying the configuration file alone. Any component can thus be added or removed seamlessly without having to recompile the code. A hierarchical class structure ensures code reusability through inheritance. New components may inherit properties from established classes and with the required functionality added to the derived class.

Distributed sensor and control systems are facilitated by integrating internode communications within the framework. The communications links may be implemented using any available communications channel and maintain data synchronisation between nodes, allowing connected systems to operate as a single system.

A multithreaded programme architecture ensures task exclusivity, high data throughput, prevents system delays and allows a graceful degradation in performance if the system is overloaded.

Table 1 compares the proposed PC based framework with several other platforms commonly used for control. Our system is very easily reconfigured without having to recompile the code for the system. This is achieved by breaking the operations into separate components and then specifying the connections between these components via the configuration file. In contrast, a traditional controller would require significant rework and recompilation even for relatively minor changes. The proposed framework is scalable within the CPU and bandwidth limits of the processor and where this is a limitation, the framework provides a mechanism for easily creating distributed systems. Under overload, the performance should degrade gracefully. From the table, even though platforms such as LabView™ provide a flexible framework for integrating sensing and control, there is a need for a low cost, generic platform. One potential limitation of the approach of separating the computational components is an increased overhead or latency. It is acknowledged that this is an important consideration in many control applications, so the framework is designed to minimise the latency through implementation of prioritised queues and multitasking. This prevents one slow component from degrading the performance of the rest of the system.

**Table 1** Feature comparison between the proposed framework and other platforms for controller implementation

| | *Framework using C++ on a PC* | *Traditional controller on a PC or microcontroller* | *LabView on a PC* | *PLC* |
|---|---|---|---|---|
| *Reconfigurability* | | | | |
| • Programming | Configuration specified in a text file; no programme change required | Recoding the application for each project when a change is made | Graphical icon based; flow diagrams need to be changed | Ladder logic or function blocks need to be changed |
| *Scalability* | | | | |
| • Connections | Limited by processor speed and bandwidth | Limited by processor speed and bandwidth | Limited by processor speed and bandwidth | Limited by I/O and bus bandwidth |
| • Overload performance | Prioritise certain processors, the data rate of other processors will decrease | Implementation dependant; generally processor lag will increase | Implementation dependant; processor lag if not using threads | Possible data lost or inaccurate results |

**Table 1** Feature comparison between the proposed framework and other platforms for controller implementation (continued)

| | *Framework using C++ on a PC* | *Traditional controller on a PC or microcontroller* | *LabView on a PC* | *PLC* |
|---|---|---|---|---|
| *Flexibility* | | | | |
| • Effort to add new sensor, actuator or controller | Only change is to configuration file or add new code via a DLL | Implementation dependant; add extra code to project, ensuring code fits into current resources | Use a DLL or recompile code into main application | Modify ladder logic, then recompile |
| • Range of applications | Wide range due to ease of coding and range of devices designed to be connected to a PC | Wide range due to relatively low cost of microcontroller implementation | Wide range due to ease of coding and range of devices designed to be connected to a PC | Generally limited to industrial use |
| • Control architecture | Flexible control architecture, easily altered | Inflexible | Partially flexible | Inflexible |
| *Performance* | | | | |
| • Limits to performance | Port and CPU speed | Port and CPU speed; memory and IO availability for microcontroller | Port and CPU speed | Controller speed and scan time |
| • Latency | Minimal due to prioritised queues and multithreading | Implementation and problem size dependant | Implementation dependant | Controller speed and scan time |
| • Advantages/ primary strengths | Framework can be used over multiple projects; easily reconfigured | Slightly faster execution due to less overhead | Visual language is intuitive to programme; large base of preprogrammed modules | Embedded design; small form factor |
| • Disadvantages/ primary weaknesses | Overheads from reduced coupling between components | Application needs to be designed each time; adding new devices may require major system changes | Application needs to be designed each time; expensive | No threading functionality |

The proposed framework will have wide ranging applications in the areas of home and office automation, process monitoring and control, surveillance and robotics. This has been demonstrated in this paper through a brewery temperature controller and a robot soccer system.

## References

Armstrong, D., Crane, C., Novick, D., Wit, J., English, R., Adsit, P. and Shahady, D. (2000) 'A modular, scalable, architecture for unmanned vehicles', *Proceedings of the Association for Unmanned Vehicle Systems International (AUVSI) Unmanned Systems 2000 Conference*, Orlando, FL, pp.1–15.

Bailey, D. and Sen Gupta, G. (2004) 'Error assessment of robot soccer imaging system', *Proceedings Image and Vision Computing New Zealand 2004*, Akaroa, New Zealand, pp.119–124.

Branch, J.W., Davis, J.S., Sow, D.M. and Chatschik, B. (2005) 'Sentire: a framework for building middleware for sensor and actuator networks', *Third IEEE International Conference on Pervasive Computing and Communications Workshops*, *PerCom 2005 Workshops*, pp.396–400.

Choi, J., Shin, D. and Shin, D. (2005) 'Research and implementation of the context-aware middleware for controlling home appliances', *IEEE Transactions on Consumer Electronics*, Vol. 51, No. 1, pp.301–306.

Cohen, M.A., Sairamesh, J. and Chen, M. (2005) 'Reducing business surprises through proactive, real-time sensing and alert management', *Proceedings of the 2005 Workshop on End-To-End, Sense-and-Respond Systems, Applications and Services (Seattle, Washington, 05 June, 2005). International Conference on Mobile Systems, Applications and Services,* pp.43–48.

Estrin, D., Govindan, R., Heidemann, J. and Kumar, S. (1999) 'Next century challenges: scalable coordination in sensor networks', *5th International Conference on Mobile Computing and Networking, Mobicom '99*, Seattle, USA, pp.263–270.

Hohmann, P., Gerecke, U. and Wagner, B. (2003) 'A scalable processing box for systems engineering teaching with robotics', *Proceedings of International Conference on Systems Engineering*, Coventry, pp.267–271.

Lee, K.B. and Schneeman, R.D. (2000) 'Distributed measurement and control based on the IEEE 1451 smarttransducer interface standards', *IEEE Transactions on Instrumentation and Measurement*, Vol. 49, No. 3, pp.621–627.

Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D. and Anderson, J. (2002) 'Wireless sensor networks for habitat monitoring', *Proceedings of First ACM International Workshop on Wireless Sensor Networks and Applications, WSNA'02*, Atlanta, USA, pp.88–97.

Marti, P., Yepez, J., Velasco, M., Villa, R. and Fuertes, J.M. (2004) 'Managing quality-of-control in network-based control systems by controller and message scheduling co-design', *IEEE Transactions on Industrial Electronics*, Vol. 51, No. 6, pp.1159–1167.

Meunier, R. (1995) 'The pipes and filters architecture', *Pattern Languages of Program Design*, J.O. Coplien and D.C. Schmidt (Eds). New York: ACM Press/Addison-Wesley Publishing, pp.427–440.

Sen Gupta, G., Messom, C.H. and Demidenko, S. (2004) 'State transition based (STB) role assignment and behaviour programming in collaborative robotics', *Proceedings of the 2nd International Conference on Autonomous Robots and Agents*, *ICARA 2004*, Palmerston North, New Zealand, pp.385–390.

Shafer, S.A., Stentz, A. and Thorpe, C.E. (1986) 'An architecture for sensor fusion in a mobile robot', *IEEE Conference on Robotics and Automation*, pp.2002–2011.

Sohrabi, K., Merrill, W., Elson, J., Girod, L., Newberg, F. and Kaiser, W. (2004) 'Methods for scalable self-assembly of ad hoc wireless sensor networks', *IEEE Transactions on Mobile Computing*, Vol. 3, No. 4, pp.317–331.