

REAL-TIME VIRTUAL SENSORS

Edwin Armstrong*, Nigel Hardy**

* *Department of Computer Science, Western Baptist College,
Salem, OR, USA. efa@cs.wbc.edu*

** *Department of Computer Science, University of Wales
Aberystwyth, UK. nwh@aber.ac.uk*

Abstract: Virtual sensors as software abstractions to support applications programmers, particularly in automated assembly, have been proposed and used. In earlier implementations, virtual sensors (and therefore the whole system) have not been designed to support real-time operation. Design work and an implementation are reported here which demonstrate progress towards real-time virtual sensors providing time guarantees while retaining the character of virtual sensors. Additional features required for their implementation are considered. *Copyright© 2000 IFAC*

Keywords: Architectures, Real-time, Sensor systems, Assembly robots

1. INTRODUCTION

The work reported here extends the techniques of *Virtual Sensors* (VS) to support real-time use. VS, as considered here, are software abstractions to support the use of sensed data. They have been developed to support applications programmers, particularly in automated assembly, where relatively frequent re-programming of the use of a large number of diverse sensing devices is required (Hardy *et al.*, 1992b). VS can be dynamically created and destroyed to provide structures which match the functional requirements of the task.

VS are provided and supported by a *sensor integration system* and used by a consumer module, normally called the *supervisor*. The ViSIAR idealised framework for designing and building sensor integration systems (Hardy and Aftab Ahmad, 1999; Hardy and Aftab Ahmad Maroof, 1999) provides an outline design and a number of design questions which must be answered. In addition, it describes a range of VS classes. VS provide both *control abstraction* and *data abstraction* (Hardy *et al.*, 1992a) and each VS class provides a control abstraction to support a particular type of sensor use. The set of control abstractions provided

by a particular sensor integration system should match the sensing requirements (in terms of the use made of data) of the application in hand. Common sensor classes include *state sensors* for immediate collection of values, *event sensors* for the asynchronous detection of significant changes of state and *trigger sensors* which collect the value of a state sensor on a cue from an event sensor (Rowland and Nicholls, 1995). This small set of classes illustrates the important concept that both real sensors and virtual sensors can act as input to VS, thereby establishing a hierarchy of sensing modules.

The data abstraction provided by VS is also designed to support the use of sensed data. Strong or weak typing and a rich or limited set of data types can be chosen to coincide with the programming environment of the supervisor (Aftab Ahmad, 1996).

2. RELATED WORK

VS as described grew out of the concepts of logical sensors (Henderson and Shilcrat, 1984; Henderson *et al.*, 1985; Henderson, 1985; Dekhil and

Henderson, 1998), where there was an emphasis on resilience to sensor failure, and has similarities with other contemporary work (Milovanovic, 1987; Weller *et al.*, 1990). More recently, work in mobile robots has shown an increasing tendency to similar software constructs. *Modular perception* is reviewed by Arkin (1998) where a number of higher level abstractions to support particular robot programming paradigms are discussed. Perceptual schemas (Arkin and MacKenzie, 1994) provide pertinent information in a way appropriate for the motor schema behaviours. The architecture from LAAS-CNRS (Alami *et al.*, 1998) uses a logical robot system to reduce the hardware dependency of higher level code.

RCCL (Lloyd and Hayward, 1993) provides an enhanced UNIX environment for robot control. A real-time “trajectory generator” task runs at between 50 and 200Hz. An RCCL program provides application routines which run within the trajectory generator task. These “control level” programs can involve sensor feedback. The “generalized-compliant-motion” primitive receives commands from higher levels and generates the required trajectories. RCCL uses data abstractions for sensors and actuators, dynamic creation and deletion of tasks and a layered organisation of control. Slave controllers are connected via a time deterministic network.

3. REAL-TIME VIRTUAL SENSORS

Many sensing activities which an applications programmer will specify, particularly in the fields of manipulator robots and automated assembly, do not require real-time guarantees. Sensed values can be returned at any time or at any rate convenient to the sensing mechanism and the success of the overall task will not be affected. In many cases a faster response will be advantageous, often for economic reasons, but slow or unpredictable responses will not cause failure.

Some robotic activities require real-time responses. Active compliant motion and guarded moves are typical examples (McKerrow, 1991). Real-time sensor use, without associated actuation, can be important in, for example, quality assurance and process monitoring, where regular or timely sensing is required. In typical industrial (and most research) environments, such tasks are hard-coded. Particular controllers involving specified sensors and actuators are identified at system design time and provided as “black boxes”.

3.1 Requirements

What might real-time VS mean to a supervisor? The principle requirement is for it to be possible

to dictate that a sensing event takes place at a specified time. This requirement implies two further features.

First, any computer system has resource limitations and it may not be possible to satisfy a particular real-time sensing request. There may, for example, be inadequate processing power, communications bandwidth or speed in the basic sensing hardware. A real-time VS integration system must, therefore, have the ability to determine that a request cannot be met and then to refuse it. This allows a simple supervisor to avoid potential failure and may offer the opportunity for a sophisticated supervisor to re-plan, using slower speeds or an otherwise less demanding method.

Second, it is inevitable that timing failures will occur, even if they rarely do so. Bugs and hardware failures are always with us. Standardised error reporting is an important aspect of the VS abstractions. Timing failures represent a new type of VS failure and their detection and reporting will require new facilities.

Specifying sensing at a particular time is a basic requirement. More generally it is necessary to co-ordinate activities. This implies both timing guarantees and a scheduling method to administer tasks. Timing guarantees cannot be effectively added on to an existing system; they must be designed into a system from the start. Earlier work has concentrated on virtual sensors, and neglected the actuation side. In the work reported here, simple virtual actuators have been employed. The usage abstraction of these is limited to a simple create-execute-destroy life-cycle (c.f. the more sophisticated abstractions for sensing (Hardy and Aftab Ahmad Maroof, 1999)) but serves to provide a consistent interface and schedulable actions with known timings.

3.2 Our Understanding of Real-Time

The objective of a real-time system is to meet the individual timing requirements of each task. These requirements are each expressed as a *Critical Time Frame* (CTF) within which a particular event must occur. We recognise 3 types of event and use them to characterise systems:

- Failure to be within the CTF for a **hard** real-time event constitutes a system failure. This may result in an immediate physical problem, but often it will be appropriate to initiate a shut-down procedure. This must be supported by reporting of timing failures.
- A system is expected to be tolerant to occasional missed CTF for **soft** real-time events. It will provide a mechanism, such as a default value or action, to handle them. At some pre-

defined number or frequency of missed CTF, the mechanism will consider that the system has failed. Again, timing failure detection and reporting will be necessary to support this.

- A **non** real-time event is one with no CTF.

A CTF requires a start and stop time. In the demonstration system described below, these are represented as a start time plus a duration in clock ticks. This can be represented as $CTF(s, d)$. In addition, it is convenient to define periodic events as $CTF(s, d, p, n)$ with a period of p ticks occurring n times. n can be considered to default to 1 and may take a special value *forever*.

4. THE ELEMENTS OF A REAL-TIME VIRTUAL SENSOR INTEGRATION SYSTEM

The following general design decisions about the elements of a real-time VS integration system are made. Alternative decisions are possible, but have not been investigated.

4.1 A Distributed System

A distributed multi-processor system is a common decision in real-time applications. Tindell (1993) gives 3 reasons: i) fault tolerance based on replication of resources including processors; ii) processing requirements, including the need for simultaneous execution of tasks; iii) the distributed nature of the real world application. These all apply to VS systems.

In addition, it is clear that some elements of a VS based system will not require real-time guarantees, and indeed it would be hard to provide them. Sophisticated supervisors (perhaps involving planning and/or user interaction) should not be in a real-time environment. Some sensing will not require or be appropriate for real-time operation. It is clear, therefore that the final system will be mixed – real-time and non real-time – and a multiprocessor system provides the obvious framework for this. Some nodes will be real-time while others are not. Real-time tasks, often specified by non real-time nodes, can be delegated to real-time nodes.

4.2 A Time Deterministic network

Given distributed processors in a real-time system, all delays must be known in order to guarantee task completion within the required time frame. It is, therefore, necessary to be able to determine and guarantee *message latency* - the time from the initiation of a message to its receipt. Note

that non real-time nodes in the distributed system must play their part and support this network.

4.3 A Stable and Fault Tolerant Execution Platform

To support real-time tasks a predictable execution environment which detects errors, reports them and degrades gracefully is required. It must also support concurrency and timing constraints.

4.4 Scheduling

A scheduler should sequence tasks to meet all deadlines and possibly maximise system utilisation. In a real-time system there may be no sequence which allows all CTF to be met. A scheduler is therefore required to detect and reject impossible task loads.

Schedulers may be static or dynamic. A static scheduler uses complete *a priori* knowledge of the task set and its constraints, such as deadlines, computation times and precedences. This assumption is realistic for many real-time systems (Stankovic *et al.*, 1995). It produces an order of execution release times as a table, often called a calendar, which is stored. A calendar typically loops. It is fixed and the system cannot respond to run-time information. It may be produced manually or using algorithms such as Earliest Deadline First, followed by application of heuristics (Xu and Paras, 1991) or one of a range of standard optimisation techniques (Altenbernd and Hansson, 1997).

A dynamic scheduler uses a complete knowledge of current tasks and changes the schedule over time as new tasks arrive. It may result in a high overhead but can be flexible and responsive. A range of techniques is available, including Earliest Deadline first, Rate Monotonic (Liu and Layland, 1973) and Deadline Monotonic (Tindell and Hansson, 1995) which offer schedulability calculations.

4.5 A Distributed Clock

The presence of a system-wide clock by which the timing of events can be synchronised or otherwise co-ordinated is considered essential to a distributed real-time system's function (Gergeleit *et al.*, 1994; Iyengar and Jayasimha, 1994). CTF then have a common meaning across all nodes.

4.6 Modifications to a VS System

The following general additions to a system, such as ViSIAr are expected.

- integration with a scheduling and control strategy.
- temporal constraint variables in the VS abstractions;
- an exception mechanism to handle timing failures;

To meet tight CTFs, it may be necessary to design and build the system to minimise resource usage and contention. This is a practical, rather than theoretical, requirement. More fundamentally, all real-time components must be built with deterministic timings. Physical sensor and VS routines must all execute in known times.

4.7 General Architecture

A corollary of the decisions above is the following division of the final system. The **supervisor** (and possibly some other elements of the system) will be *non real-time*. This leaves the applications programmer free of real-time design and implementation demands. The **VS and the scheduler** must be real-time. To support scheduling, VS operations must have known execution times. Use of VS may therefore be seen as a mechanism for restricting, to a known and timed set, the routines which may be called in a real-time task. **Physical sensors**, and supporting low level components (the fixed part of ViSIAR (Hardy and Aftab Ahmad Maroof, 1999)) must be time deterministic. The concept of a Device Specification File (DSF) supports this. The operation of each physical device can be timed and appropriate parameters recorded in a DSF for use in scheduling. The latency between requesting a value and its return should be recorded. The maximum frequency at which values can be requested may also be relevant.

5. A DEMONSTRATION ENVIRONMENT

A demonstration system has been built to explore and validate the ideas presented above. It has the following features.

5.1 A Distributed System

The supervisor software is run on the main processor of an IBM PC compatible machine. Slave processors, on physically distant boards, are Motorola 68HC11 based boards.

5.2 A Time Deterministic Network

CAN, the *Control Area Network* was chosen. It has the required real-time features, a well defined

standard and multiple implementation sources. There is also a significant user community and published material on its use. RS485 serial serial communications over a shielded twisted pair provides the physical layer. A CAN chip (AN82527) provides the protocol layer.

5.3 A Stable and Fault Tolerant Execution Platform

The μ C/OS real-time kernel (Labrosse, 1992) was chosen to run on each of the 68HC11 processors. This is a small, open-source, system which is easily modified and ported to new platforms. For this work, a 68HC11 port was required and the latencies of each kernel routine were measured. Minor modifications were necessary, including the addition of a real-time task to schedule the dispatch of tasks on pre-defined system clock ticks. This was done in a way similar to that included in later versions of the kernel (Labrosse, 1995).

5.4 Scheduling

For the purposes of demonstration, a static scheduling mechanism is used. Start times of all tasks are determined before run-time and represented in a table which is then used by a dispatcher on each board to initiate tasks.

5.5 A Distributed Clock

The dispatchers all operate on synchronised system-wide clock ticks. This is implemented by broadcasting a command to all nodes which causes them to bring their local clocks into synchronisation. Each real-time kernel then handles a 50 Hz interrupt which invokes a pre-emptive priority based dispatcher. Synchronisation of event initiation is thereby guaranteed at this resolution.

5.6 Modifications to a VS system

All real-time VS interactions must execute in scheduled real-time tasks. Each VS class provides operations appropriate to its abstraction (Hardy and Aftab Ahmad, 1999). In a real-time environment, the simple **Interrogate** operation which returns a current sensed value is the crucial one. It applies to state sensors, to condition sensors (which test a state sensor value against a threshold and return a boolean) and to logging sensors (which collect values on a specific stimulus and store them for later analysis). Event sensors detect and report on changes in condition sensor values. These can be active, in which case they provide asynchronous notification of the change,

or passive, in which case an **Interrogate** must be performed to determine whether the event has occurred. Clearly, only passive event sensors can be scheduled in a simple way.

VS can be created in a non real-time environment but interrogation operations can be performed in real-time tasks. This requires that they be time deterministic. That is, the operation, the operations on any VS involved in the hierarchical definition of the VS, and the physical sensing activities at the base of that hierarchy must all execute in known times.

The demonstration system can perform interrogations within real-time tasks. Such a task is scheduled using a CTF. This provides a basis for an abstraction where standard VS creation parameters plus the CTF parameters can be presented for the dynamic creation of a real-time VS.

The system detects timing failures. Each scheduled task has a time, specified by a CTF, by which it must be completed. If it is not, the dispatcher detects this and reports a failure. The failure might have been caused by something physical (nothing can be done about this and it is a real problem which should be reported). Alternatively, it might have been caused by a scheduling failure. If the scheduling algorithm or the mechanism for determining how long a task should take is faulty or if any of the fundamental timing of routines are incorrect then the estimate of how long the task would take and/or whether there is enough processor time to do it may be wrong.

A high priority task, called MkSafe, is associated with the timing failure detection. Each device can be set to a previously defined safe state by MkSafe, thereby attempting a controlled shutdown.

6. DEMONSTRATION SYSTEM EXAMPLES

6.1 *Scheduled Observations*

Regular sampling of a simple state sensor can be set up in the demonstration system as follows. The system is coded in C but is presented here in pseudo code for simplicity and clarity. The interrogation operation is assigned to a task.

```
Task10:
{
    result = Interrogate(S1)
}
```

The virtual sensor S1 is assumed to be previously defined. In this simple example, the returned value is stored in a global variable. Other uses, such as storing in a log or making the result available to another task via a more sophisticated mechanism are possible.

Before real-time operation commences, or as part of another real-time task, the CTF for the task must be set.

```
CTF(Task10) = (now+500, 10, 100, forever)
```

This provides (for use by the dispatcher) the information necessary to make the task runnable (in 500 ticks from now and at 100 tick intervals after that) and to detect timing failures. If, at time `now + 510` the task has not completed, the dispatcher can report this.

6.2 *Reported Priority Based Failure*

The major example for the system involves three output devices to provide visual demonstration. Three 68HC11 slave boards are used. Each controls a fan which is able to lift a ping-pong ball up a plastic tube. Three light sensors are situated on each tube to detect the passing ball. Real-time guarantees are necessary so that no sensing event is missed. In the demonstration, a real-time task raises a ball to one of the sensors and then allows it to fall. Such tasks must not be pre-empted lest a sensing event be missed.

Two of the boards also control a laser drawing device. Each board controls a stepper motor which rotates a mirror. Co-ordinated motion of the two mirrors allows a shape to be drawn in laser light on a wall. To draw a triangle, as in the demonstration, start times for each edge must be synchronised on the two boards. This involves use of the distributed clock to schedule synchronous tasks. Crucially, the tasks which control the two mirrors must have sufficient time resource to operate on time and to completion of the motion. The system can guarantee this before the motion commences.

Finally, each board has a pair of lights. These are flashed by a low priority task which is, effectively, the idle process.

The calendar for this demonstration was generated manually using observed timings. This allows timing faults to be introduced for testing. As an example, on one board a simple task to flash the lights (at a higher than normal rate) for a fixed period was scheduled at the same time as raising the ping-pong ball. It was given a higher priority and therefore was consistently dispatched until it has completed. This left insufficient time for the ball to be raised to the sensor before that task's CTF had expired. This failure was detected and reported. The same board successfully controlled one of the mirrors throughout; demonstrating that detection and reporting of failure do not affect other real-time tasks.

7. CONCLUSIONS

The demonstration system provides for interrogation of virtual state sensors within a specified CTF and will report any missed real-time deadlines to the supervisor. A simple virtual actuator framework provides the same facilities for actuation. Timings are manually calculated and scheduling is done off-line. A DSF table and use of a restricted set of routines as suggested and supported under the VS framework offer a way forward to automatic calculations of timings for a VS hierarchy. Dynamic scheduling algorithms are to be found in the literature. The demonstration focuses on state sensor interrogation. Timed log sensor use is a natural extension. Condition and passive event sensor interrogation are also believed to fit this model but have yet to be demonstrated.

8. REFERENCES

- Aftab Ahmad (1996). A Framework for the Design of Software for Robotic Sensing. PhD thesis. University of Wales, UK.
- Alami, R., R. Chatila, S. Fleury, M Ghallab and F. Ingrand (1998). An architecture for autonomy. *Int. J. Robotics Res.* **17**(4), 315–337.
- Altenbernd, Peter and Hans Hansson (1997). The slack method: A new method for static allocation of hard real-time tasks. Technical Report C-Lab Report 19/97. University of Paderborn. Paderborn, Germany.
- Arkin, R.C. and D. MacKenzie (1994). Temporal coordination of perceptual algorithms for mobile robot navigation. *IEEE Trans. on Robotics and Automation* **10**(3), 276–286.
- Arkin, Ronald C. (1998). *Behaviour-based robotics*. MIT Press.
- Dekhil, Mohamed and Thomas C. Henderson (1998). Instrumented sensor system architecture. *Int. J. of Robotics Res.* **17**(4), 402–417.
- Gergeleit, M., J. Kaiser and H. Streich (1994). DIRECT: Towards a distributed object-oriented real-time control system. In: *Proc. Workshop on Concurrent Object-based Systems*.
- Hardy, Nigel and Aftab Ahmad (1999). Decoupling for re-use in design and implementation using virtual sensors. *Autonomous Robots* **6**(3), 265–280.
- Hardy, Nigel and Aftab Ahmad Maroof (1999). ViSIAR - a virtual sensor integration architecture. *Robotica* **17**(6), 635–647.
- Hardy, N.W., H.R. Nicholls and J.J. Rowland (1992a). The design of sensing commands in the InFACT assembly machine. In: *Proc. 23rd Int. Symp. Industrial Robots (ISIR '92)*.
- Hardy, N.W., H.R. Nicholls and J.J. Rowland (1992b). Supervisory System Software. In: *InFACT: Project, Concepts, Machine*. (C. Loughlin, Ed.). MCB University Press. Bradford.
- Henderson, T., C. Hansen and B. Bhanu (1985). The specification of distributed sensing and control. *J. Robotic Systems* **2**(4), 387–396.
- Henderson, T.C. and E. Shilcrat (1984). Logical sensor systems. *J. Robotic Systems* **1**(2), 169–193.
- Henderson, Tom (1985). The specification of logical sensors. In: *Workshop on Intelligent Control*. (TH0146-1/86/0000/0095, © IEEE). pp. 95–101.
- Iyengar, S.S. and D.N. Jayasimha (1994). A versatile architecture for the distributed sensors integration problem. *IEEE Trans. Computing* **43**(2), 175–185.
- Labrosse, Jean J. (1992). *Micro-C/OS The Real-Time Kernel*. R & D Publications. Lawrence, Kansas. ISBN 0-13-031352-1, Distributed by Prentice Hall.
- Labrosse, Jean J. (1995). *Embedded Systems Building Blocks*. R & D Publications. Lawrence, Kansas. ISBN 0-87930-440-5.
- Liu, C.L. and J.W. Layland (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM* **20**(1), 46–61.
- Lloyd, John and Vincent Hayward (1993). Real-time trajectory generation in Multi-RCCL. *J. Robotic Sys.* **10**(3), 369–390.
- McKerrow, P.J. (1991). *Introduction to robotics*. Addison-Wesley. Sydney.
- Milovanovic, R. (1987). Towards sensor-based general purpose robot programming language. *Robotica* **5**(4), 309–316.
- Rowland, J.J and H.R. Nicholls (1995). A virtual sensor integration implementation for a flexible assembly machine. *Robotica* **13**(2), 195–199.
- Stankovic, John A., Marco Spuri, Marco Di Natale and Giorgio Buttazzo (1995). Implications of Classical Scheduling Results for Real-Time Systems. *IEEE Computing* **28**(6), 16–25.
- Tindell, K. (1993). Fixed Priority Scheduling for Hard Real-Time Systems. PhD thesis. Dept. Computer Science, University of York, UK.
- Tindell, Ken and Hans Hansson (1995). Real-time systems and fixed priority scheduling. Technical Report Tech. rept. DoCS 95/notes. Uppsala University. Uppsala, Sweden.
- Weller, G.A., F.C.A. Groen and L.O. Hertzberger. (1990). A sensor processing model incorporating error detection and recovery. In: *Traditional and Non-traditional Robotic Sensors*. (T.C. Henderson, Ed.). NATO. Springer-Verlag, Berlin. pp. 351–363.
- Xu, J. and D.L. Paras (1991). On satisfying timing constraints in hard real-time systems. In: *Proc. ACM SIGSOFT 91 Conf. Software for Critical Systems*. ACM.