“Broadcaster”: An Architectural Description of a Prototype Supporting Real-time Remote Data Propagation in Distributed Manufacturing

V. Barot*, R. Harrison*, C.S. McLeod* and A. A. West*
*Wolfson School of Mechanical and Manufacturing Engineering
Loughborough University, Loughborough, U.K
Email: {V.Barot, r.harrison, C.S.Mcleod, a.a.west}@lboro.ac.uk

Abstract - **Globalisation of manufacturing activities tend to geographically distribute manufacturing entities, resulting into each entity adopting its own mechanism, for aggregating and analysing real-time shop floor machines’ information. The enterprise systems normally employ sophisticated and computationally expensive techniques to access this data, even if they operate remotely having limited network connectivity and system legacies. There is a need to propagate machine information in soft real-time basis to these entities regardless of their geographic locations and / or mechanisms. Authors are presenting an architectural description of a prototype system called the “Broadcaster” which efficiently distributes manufacturing machine information to a number of remotely located global engineering partners. This prototype addresses the emergent system issues like maintainability, reliability, integrity, robustness, flexibility and performance using a heterogeneous composition of “Blackboard” repository model with an event-driven invocation technique, implemented using interface-based strategy. The design and implementation assumes the control environment description to be engineered using the Component-Based system paradigm. Presently, the prototype is evaluated on a demonstration test rig provided by the Ford Motor Company, which is implemented using a fully web services distributed control device called FTB, designed by the Schneider Electric Company. Based on the evaluation from the implementation stage, authors have justified and concluded the paper highlighting the key benefits of this approach, and described any future research that is to be carried out.**

I. INTRODUCTION

In this industrial era, operation of manufacturing activities has evolved to follow the concept of a distributed paradigm [1]. One of the key factors initiating such a change is globalisation of manufacturing process. In order to provide the required flexibility and competitiveness, these manufacturing activities tend to be geographically distributed spanning various cities, countries and even continents. Every manufacturing entity possesses their individual mechanism for aggregating and analysing its machine information. Furthermore, the volume of interesting data emerging from the shop-floor machines is increasing and the management of it is becoming more complex. The manufacturing community often use sophisticated and computationally expensive techniques to access and analyse this data. This leads to the concept of satisfying various requirements identified by this distributive business nature [2]. In addition, a number of challenges and limitations in terms of availability of real-time machine information exist [3]. The scenario becomes even complicated when the enterprise systems operate in a remote computing environment with limited features such as network connectivity and / or application legacy.

In order to provide the necessary satisfaction to the requirements emerging from this distributed paradigm, a good data management and propagation strategy has to be adopted which allows efficient distribution of machine information stemming from the shop floor, in a reliable and timely manner, to a number of global engineering partners regardless of their distribution nature and / or their mechanisms. In the light of such a requirement, the authors have developed a prototype called the “Broadcaster.” It is responsible to not only efficiently collect and manage real-time information from the shop floor machines, but also propagate it to unlimited number of heterogeneous engineering remote partner resources (for example; HMI, Remote HMI, ERP systems). The prototype addresses maintainability, reliability, robustness, flexibility, data integrity and the performance issues using a heterogeneous design approach. The prototype applies a blackboard repository model and an event-driven invocation technique with clearly defined interface strategies for efficient data management and propagation.

The system requirements and the state flows within the event driven design are described by using UML and programmed using OOP techniques in dot Net framework. The triggering events in the designed system correspond to the messages transmitted between the components of the prototype. The system employs the basic publish/subscribe technique [4] for efficiently propagating the data to those clients in need, leading to avoiding unnecessary bandwidth utilisation. The paper is organised as follows: The next section details the architectural description of the prototype design. Authors have identified a number of targets and principles to achieve this desired system design. A description of the event-based approach with the blackboard repository incorporated into this system is also discussed, together with the interfaces which offer flexibility to the prototype. Section three discusses the implementation status of the prototype on a web service-enabled test rig. Based on the design and implementation, the authors have concluded this paper in the section four, which identifies some major benefits of this implementation and provides the direction of any future research work.

II. ARCHITECTURAL DESCRIPTION

A. Design Targets

The successful determination of any system is greatly affected by a good architectural design [5]. Architecture corresponds to the organisation of system functional units and the specification of their interacting interfaces [6]. Prior to the architectural description of this prototype, it is essential to specify the design targets for its justification:

- Mainly targeted at remote distributed client instances having features of varying degree and limitations.
- Seeks to minimise the time consumed in collecting, processing and propagating the machine information.
- Supports the issue of scalability, maintainability, reliability, integrity and performance in the design.
- Incorporates the debugging and monitoring functionality within the system, for local, as well as, remote propagation.
- Operates in multi-institutional and heterogeneous environments.

B. Component-Based Paradigm

The achievements of these targets assume that the machine control system architecture corresponds to the Component-Based (CB) system paradigm. Thus the Broadcaster is intended to be implemented in a distributed manufacturing environment which has been engineered using the CB approach. The fundamental concept of this paradigm is that new systems are developed from existing components within the system libraries. This reduces the costs and the efforts for developing a system, as well as, reduces the time to market the product to meet new demands of customers. The major issue of reconfigurability and reuse of manufacturing systems is also addressed by this paradigm [7]. Harrison et al [8] describes the Component-Based paradigm by decomposing the "System" into "Sub systems." The sub systems are further decomposed into "Components" which comprises of a number of "Elements" with "States."

C. Design Principles and Interface Specification

The above identified targets (i.e. requirements) act as a major factor that influences the design of this prototype. The output of this design process describes the prototype's system architecture [9]. In order to achieve the design targets, a number of principles are considered when making design decisions. These principles, that drive the design of the prototype, are implemented via a set of control interfaces shown as a simplified representation in the Fig. 1. These are:

· **Mechanism independence:** The architectural design incorporates the possibility of having independence from any low-level, as well as high-level mechanism, which one may have to generate, store, transfer and / or analyse machine data. This is achieved via the "Mechanism Manager Control" that encapsulates the engineering definition associated with specific parties.

· **Geographic uniformity:** Regardless of the geographic location or the nature of the distribution adopted at higher-levels or lower levels, the information is encoded and propagated in a uniform XML format which can be decoded easily from one schema to the other, as and when required by the distribution partners. The main reason for adopting an XML-based representation is to support the legacy structure of enterprise resources as it may be inconvenient and/or expensive to discard such systems. In addition, XML has various technologies such as XSD for schema definition, which can be useful for extension and manipulation of XML descriptions. It also plays a major role in the description and provision of web services [10]. By wrapping legacy systems as web services, one can preserve the investments in a valuable system. The uniformity feature is achieved within the "Dictionary Control" of the prototype.

· **Information availability:** All the generated machine information is stored and time-stamped efficiently in a circular memory buffer, acting as a blackboard, which implements complex multi-threading queue mechanism. The storage and management is done via the "Blackboard Management Control." The access to real-time data, which is published in this buffer, is also managed via this control interface. The publishing mechanism is carried with the help of "Mechanism Manager Control."

· **Information access and security:** All the remote partners connecting to the prototype can be authenticated and validated for information access. This is implemented via the "Registry Control." Environment configurations can also be managed via this control.

An important aspect of any design process is the specification of the interfaces between all the components (i.e. objects) of the system. As the interface provides an abstract model of a system component, it does not reveal any details to other parties concerned. The prototype has precisely defined and implemented a number of interfaces (as shown in the Fig. 1.) via one or more classes defined in the system using dot Net programming language. The class (es) can be instantiated and thus can be allowed to communicate with it through an interface reference. The main reason for implementing an interface is to provide encapsulation and abstraction to the class's implementation details, hence avoiding dependencies of clients on the interface. This leads to the design to be more maintainable, flexible and powerful. In the Fig. 1, "Broadcaster Control" is the main control shell which manages and schedules all the activities in the prototype. Each top-level interface control, e.g. "Dictionary," has a number of interfaces, e.g. "GlobalDictionary" and "Control Parser", which provides a specific functionality to the system. Table I is an example of a procedural interface listing definition implemented in the Broadcaster for controlling various aspects of

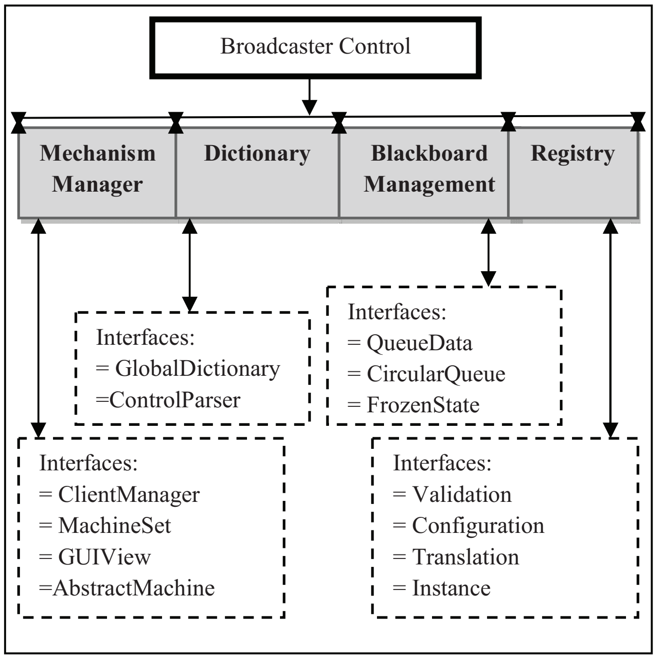


Fig. 1. The "Broadcaster Control" managing all the interfaces

the prototype such as debugging information, status events, corrupt messages, error messages and slots used from the buffer. This procedural interface is part of the "GUIView" interface.

D. Data Structure

As the information is required to be propagated to consumers in soft real-time basis, there is always a risk of collecting data faster than actually processing and propagating it further, or vice versa. To smooth out these speed differences, authors have implemented a circular data structure which acts as a buffer queue where data can be saved and retrieved easily using multithreading techniques. The table II shows a simple algorithmic implementation listing for enqueuing and dequeuing the machine data within the structure. The "Mx" is the maximum preset capacity of the queue items which can be reconfigured using a configuration tool available within the prototype. "S" corresponds to various possible states in the algorithmic implementation and FV / RV are the position values of the front / rear pointers in the structure respectively.

Table I
EXAMPLE PROCEDURAL INTERFACE DEFINITION
IN THE PROTOTYPE

```
Interface IBroadcasterView {
void debug_strs(string debug_txt)
void status_strs(string status_txt)
void errors_strs(string error_txt)
void corrupt_strs(string corrupt_txt)
void mem_used(int slot)
 }
//The above procedural interface can be implemented by a class.
//Objects instantiate a class, thus accesses the interface via an
//interface reference declaration in any language implementation.
```

Table II
ENQUEUE / DEQUEUE ALGORITHM IN THE PROTOTYPE

```
<Queue>=Enqueue
S3:
   If! =S1 then
        If ($RV_{<queue>} \rightarrow Mx-1$) then
$RV_{<queue>} = FV_{<queue>}$
 S4
        Else
$RV_{<queue>}$ ++
 If ($FV_{<queue>} \rightarrow RV_{<queue>}$ AND Queue [$FV_{<queue>}$! =NULL] then
ST: OVERFLOW
         Else
Queue [$RV_{<queue>}$] $\leftarrow Q_{value}$

<Queue>=Dequeue
S4:
  If !S2 then
      If ($FV_{<queue>} \rightarrow RV_{<queue>}$) then
$FV_{<queue>}$ =0 AND $RV_{<queue>}$= -1)
      Else If ($FV_{<queue>} \rightarrow Mx-1$) then
$FV_{<queue>}$ =0
      Else
$FV_{<queue>}$ ++
```

Mutual exclusion mechanism is implemented which prevents the writing and the reading threads to access the same data slot at the same time.

E. Heterogeneous Composition

The architecture of the Broadcaster is based upon heterogeneous composition of the conventional "Blackboard" repository model with an event-driven invocation technique. In the prototype, the central repository represents the current state of the machine at a given time, and any change in this state triggers the selection of various execution processes. The event-based invocation is a system design method which is based on an event-driven control. This control is not embedded in an individual component but it is determined by an externally generated trigger called an event [9]. Traditionally, the event handlers allow the system to receive events and invoke the corresponding operation(s). The same approach is adopted in the design of this prototype where a number of event-handlers are designed for handling variety of events that are triggered in the system. The system requirements of the prototype are described using Unified Modelling Language (UML) [11] and the programming design of the system is based on Object-oriented techniques, where the events correspond to the messages shared between the objects of the system. Though event driven approach is a very popular technique for memory-constrained embedded systems [12], it can also be applied to the architectural design of higher level systems [13, 14]. In this design, the timing of events entirely depends on its environment, thus the system must be able to cope with the events when they occur. As the events are irregularly occurring, it causes the system to move from one state to another. For this reason, the event-driven approach implements the tasks through the system as a finite state machine (FSM). The FSM is coded as objects to represent the flow of execution through the system. The UML has been used to support the description of the FSM. The event-driven design allows the author to avoid any scalability issues which may arise in processing messages [15]. The authors have allowed the operating system scheduler to schedule the control of threads. This enables one to manage the resources and retain the performance of the system throughout its execution.

As described by Lee [16], any control system implemented using the CB paradigm has control elements whose behaviour is represented using finite state machines. The description of such behaviours can be carried out via a set of states. A transition process from one state to the other is triggered by the machine event. This leads to a number of critical messages to be received by the prototype. The prototype collects and stores the machine information in a circular data structure and controls the propagation of this information to various remote clients. As the data is collected, it is being published in a reconfigurable buffer by queuing up mechanism with the help of a dedicated interface (i.e. "Blackboard Management Control,") so that clients can subscribe to the data regardless of their geographic locations or implementation mechanisms (via "Mechanism Manager Control.") The subscription of clients is handled by this interface which in turn can interact with another interface (i.e. "Registry Control") to authenticate remote connections.

Fig. 2 shows a general architectural description of events in the design of this system. As events can either be triggered by the machine or remote clients, the corresponding events are detected by the prototype. The immediate asynchronous event listener performs this duty by concurrently handling events and forwarding the control to the classification event handler. The classification event handler has its own set of top-level event handlers which discriminates as to which type of events are generated. For example, if the event handler detects a machine event, it refers into its own registry which details all the different event types that it can handle together with the corresponding interested components (i.e. objects) for further propagation. When a particular component generates an event, for example, the availability of a state change machine message, the event handler identifies the interested party (i.e. object) by consulting its registry. This allows the handler to pass the events further to those registered components.

In the Fig. 2, the flow of events from one level to the other consumes concurrent threads to avoid performance degradation. At each major control level in the design, a small thread pool is allocated [15], which is scheduled by the operating system itself, thus simplifying the software engineering concept. The threads within the thread pool are allocated using their respective thread manager for each interface within the prototype. In the system, the events are arbitrary triggered when the values of variables within objects change. There is a dedicated interface (i.e. instance control interface in the "Registry Control") within the prototype which monitors these values and raises the corresponding events. The approach of adopting this design strategy allows the system to be decomposed down into smaller components which are represented as objects, and thus potentially enhancing the prototype reusability and maintenance.

III. IMPLEMENTATION STATUS

A. *Experimental Setup and Evaluation*

In order to establish confidence in the adopted approach, and ensure that it delivers the targets identified, evaluation of the prototype is a must. The prototype is currently evaluated using a demonstration Festo test rig provided by the Ford Motor Company. The rig is implemented using a fully web services distributed control device called FTB (Field Terminal Block) designed by the Schneider Electric Company [17]. FTB provides a truly distributed architecture that allows devices to communicate with one another in a fully autonomous environment. In the test rig, seven components are implemented using four web services-enabled FTB's fulfilling the necessary orchestration and choreography system requirements. The best way of evaluating any system is to see how well it meets the requirements identified [9]. A number of simulation test cases were created to assist in discovering any system defects and show that the system conforms to its requirements. The system had to perform correctly using a given set of simulated test cases, in order to demonstrate that the targets were delivered. The evaluation of one such machine state propagation setup is shown in the Fig. 3. Fig. 3 shows that the Broadcaster propagated machine information from the web services-enabled test rig to two remote end-user client resources such as HMI and VRML model of the engineering toolset. The Fig. 4 shows the test rig used for this setup, where the control environment description has been engineered using the CB system paradigm. The test cases simulated the actual shop-floor machine data propagation scenario whilst the rig was live (just like a real production line). Successful soft real-time machine data propagation justified the usefulness of the architectural design adopted by the authors.

In the experimental setup, machine information was generated as a stream of data when the finite state mechanism of a particular element was triggered [16]. The frequency of data generation ranged from tens to hundreds of events per second, resulting into the propagation of such information. The Broadcaster collected these data and efficiently propagated them to the connected higher-level engineering resources such as a Human Machine Interface, as shown in the Fig 3. Real-time machine transmitted information was asynchronous and soft-time in nature, that is, it was transmitted as a stream of packetized TCP/IP messages. In this context, event corresponds to the process of propagating any state change, mode change and generation of time-critical machine error for example, pairs check error.

B. *Client_instance Tool*

In order to accurately analyse a large number of client instances connecting to the prototype on real time basis, the authors developed a client_instance tool. It was authors main intention to evaluate the emergent properties of the system, therefore this tool simulated random remote connections of various clients (having different performance mechanisms) to the prototype, to evaluate the robustness, reliability and the

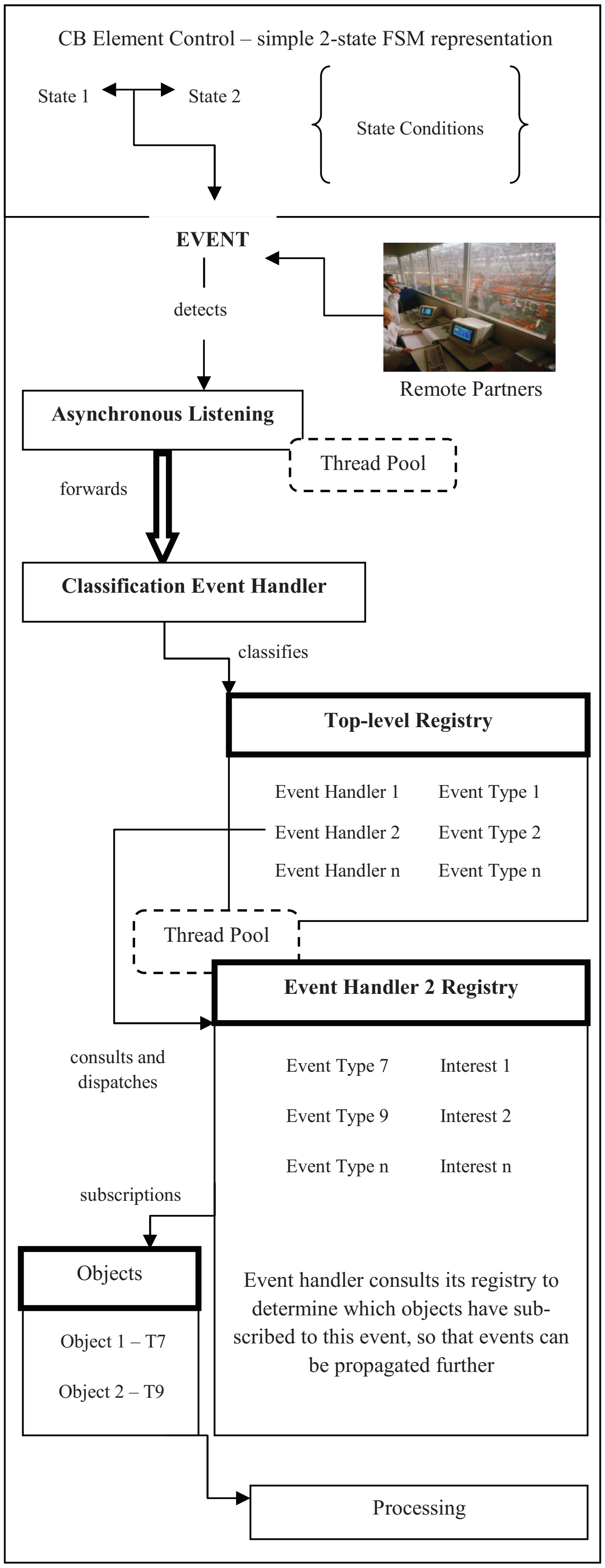
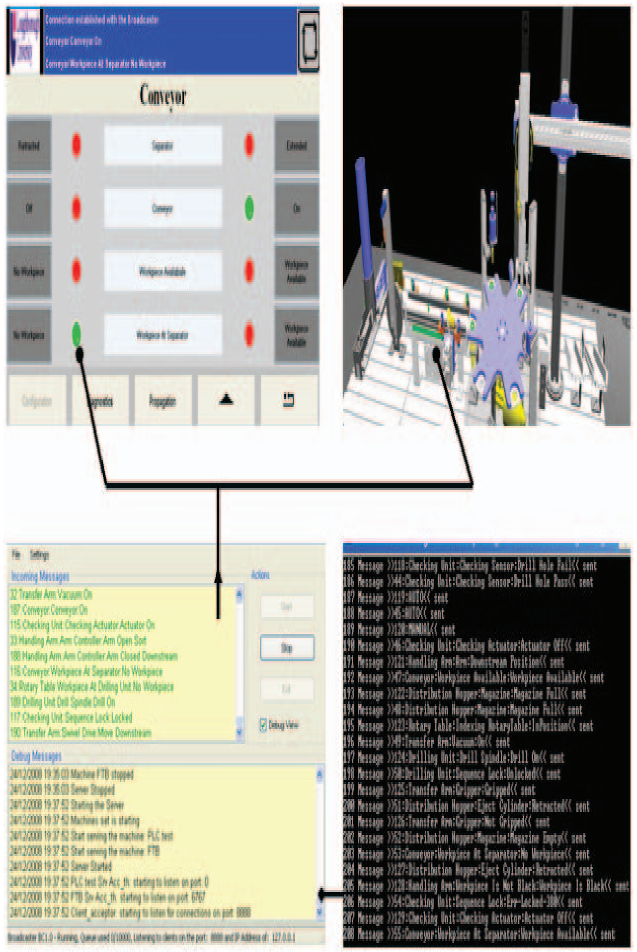


Fig. 2. Broadcaster's event-driven invocation



Incoming Messages

```
32 Transfer Arm Vacuum On
187 Conveyor Conveyor On
115 Checking Unit Checking Actuator Actuator On
33 Handling Arm Arm Controller Arm Open Sort
188 Handling Arm Arm Controller Arm Closed Downstream
116 Conveyor Workpiece At Separator No Workpiece
34 Rotary Table Workpiece At Drilling Unit No Workpiece
189 Drilling Unit Drill Spindle Drill On
117 Checking Unit Sequence Lock Locked
190 Transfer Arm Swivel Drive Move Downstream
```

Debug Messages

```
24/12/2008 19:35:03 Machine FTB stopped
24/12/2008 19:35:03 Server Stopped
24/12/2008 19:37:52 Starting the Server
24/12/2008 19:37:52 Machines set is starting
24/12/2008 19:37:52 Start serving the machine: PLC test
24/12/2008 19:37:52 Start serving the machine: FTB
24/12/2008 19:37:52 Server Started
24/12/2008 19:37:52 PLC test Srv Acc_th: starting to listen on port: 0
24/12/2008 19:37:52 FTB Srv Acc_th: starting to listen on port: 6767
24/12/2008 19:37:52 Client_acceptor: starting to listen for connections on port: 8888
```

```
Broadcaster BC1.0 - Running, Queue used 0/10000, Listening to clients on the port: 8888 and IP Address of: 127.0.0.1
```

```
185 Message >>118:Checking Unit:Checking Sensor:Drill Hole Fail<< sent
186 Message >>44:Checking Unit:Checking Sensor:Drill Hole Pass<< sent
187 Message >>119:AUTO<< sent
188 Message >>45:AUTO<< sent
189 Message >>120:MANUAL<< sent
190 Message >>46:Checking Unit:Checking Actuator:Actuator Off<< sent
191 Message >>121:Handling Arm:Arm:Downstream Position<< sent
192 Message >>47:Conveyor:Workpiece Available:Workpiece Available<< sent
193 Message >>122:Distribution Hopper:Magazine:Magazine Full<< sent
194 Message >>48:Distribution Hopper:Magazine:Magazine Full<< sent
195 Message >>123:Rotary Table:Indexing RotaryTable:InPosition<< sent
196 Message >>49:Transfer Arm:Vacuum:On<< sent
197 Message >>124:Drilling Unit:Drill Spindle:Drill On<< sent
198 Message >>50:Drilling Unit:Sequence Lock:Unlocked<< sent
199 Message >>125:Transfer Arm:Gripper:Gripped<< sent
200 Message >>51:Distribution Hopper:Eject Cylinder:Retracted<< sent
201 Message >>126:Transfer Arm:Gripper:Not Gripped<< sent
202 Message >>52:Distribution Hopper:Magazine:Magazine Empty<< sent
203 Message >>53:Conveyor:Workpiece At Separator:No Workpiece<< sent
204 Message >>127:Distribution Hopper:Eject Cylinder:Retracted<< sent
205 Message >>128:Handling Arm:Workpiece Is Not Black:Workpiece Is Black<< sent
206 Message >>54:Checking Unit:Sequence Lock:Err-Locked-30K<< sent
207 Message >>129:Checking Unit:Checking Actuator:Actuator Off<< sent
208 Message >>55:Conveyor:Workpiece At Separator:Workpiece Available<< sent
```

Fig. 3. Top-left: Remote HMI instance, Top-right: Remote VRML representation of the test rig, Bottom-left: The "Broadcaster" prototype and Bottom-right: Real-time machine events generation status

performance of the adopted approach. The robustness was evaluated based on the percentage of events causing a system failure or the amount of data corrupted whilst transmission. The reliability corresponded to the availability of data and the robustness of the system.

The performance of the system was evaluated using the scalability metric. The prototype was tested for approximately five days at defined time intervals to determine whether the events generated by the rig were actually propagated to all the connected clients using the prototype in a soft real-time basis. Obviously, the remote connections were simulated within the laboratory, so the network connection limitations were not taken under consideration at this stage. The confidentiality of the data being compromised by unauthenticated connections was controlled via the "Registry Control" interface. This functionality implemented a gateway for controlling the access to the message structure (i.e. Blackboard repository) in the prototype. The reliability and robustness of the prototype clearly indicated that the approach was powerful enough as the data was propagated to the connected remote clients in soft-real time basis. The maximum number of simulated clients to be evaluated depended on the scalability of the system. In terms of the performance, authors had to ensure that the prototype was scalable enough to handle as many client

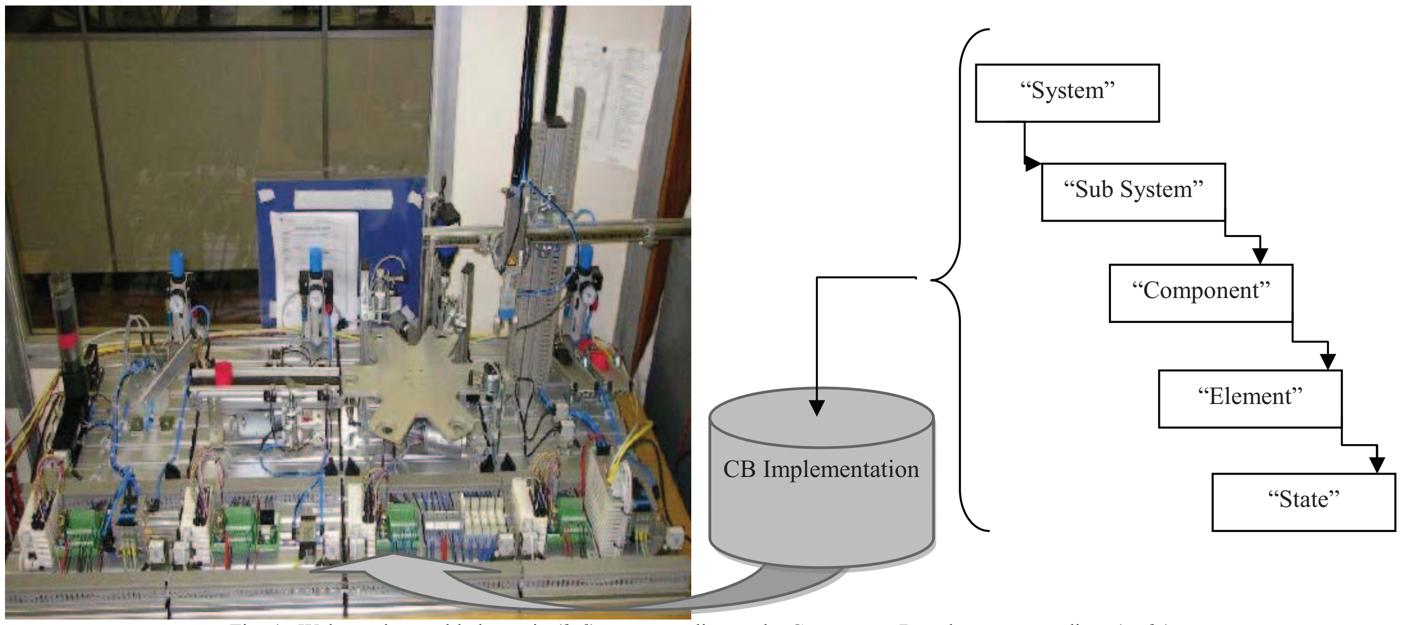
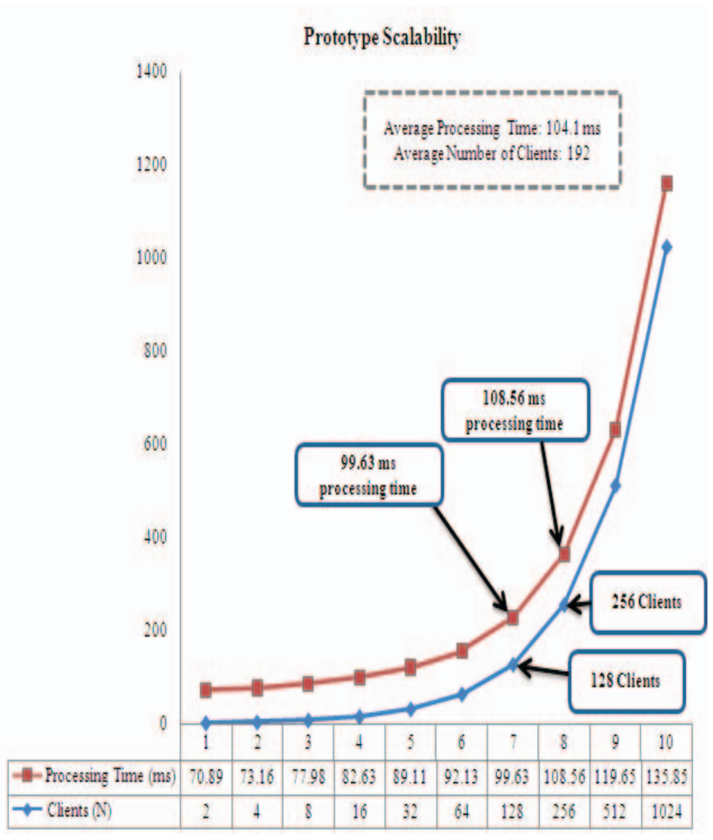


Fig. 4. Web-service enabled test rig (*left*) corresponding to the Component-Based system paradigm (*right*).

instances as possible without compromising the system functionality. Authors monitored the performance of the prototype by gradually incrementing the number of remote connections (while persisting the existing connections waiting for new events from the test rig). The clients' connections were generated using the random structure functionality of the dot Net framework and the speed of accessing data for each client varied from 10ms to 3∧10^3ms. None of the system events failed and all the connected clients downloaded data without any corruptions or complications. Throughout this phase, the size of the structure was kept constant at around 5000 items per cycle, where each item consumes up to 4 bytes of the assigned buffer slot.

Fig. 5 shows a graphical representation of the scalability evaluation carried out using the client_instance tool for the prototype. From previous research experiments, authors had identified that the average processing time taken by the prototype for propagating each client's event was approximately 74milliseconds [18]. Considering this value as a benchmark, authors evaluated the performance of the Broadcaster by incrementing remote client instances in the multiples of 2^N ($0<1<=N, N+=1$) until the overall performance of the prototype degraded to around -35% of the benchmark value. This gave the authors an uppermost boundary value of around 100milliseconds as a reference. This is quite an acceptable value as the average response time requirements for data propagations to any client application is around 500ms (or 1000ms for remote clients) in Powertrain assembly applications [19, 20]. It has to be noted that this is the maximum propagation delay expected when a machine triggers an event, which in turn has to be propagated to the clients. It can be seen that the performance of the prototype was around 100ms when the number of clients were approximately 192. Any further increment led to almost 20% growth in the processing time of the prototype. Therefore, the authors propose that the scalability of the prototype remains stable for almost approximately 200 simultaneous remote clients' connections to download shop-floor machine data. For any increment in the number of instance connections, problems may arise in terms of response times to process events in the prototype and thus the performance may suffer.



	1	2	3	4	5	6	7	8	9	10
🟥 Processing Time (ms)	70.89	73.16	77.98	82.63	89.11	92.13	99.63	108.56	119.65	135.85
◆ Clients (N)	2	4	8	16	32	64	128	256	512	1024

Fig. 5. "Broadcaster" performance evaluation

IV. CONCLUSION

The authors have discussed an architectural description of a prototype called the "Broadcaster" in this paper. This prototype offers an approach where by shopfloor machine information can seamlessly be propagated to a number of remote clients regardless of their distribution nature and / or implementation mechanisms. The prototype assumes that the description of the control environment is engineered using the Component-Based system paradigm. The architecture of the Broadcaster is based upon heterogeneous composition of the conventional "Blackboard" repository model with an event-driven invocation technique implemented using interface-based strategy. This offers the required flexibility, reconfigurability, maintainability, reliability, integrity, security and a well accepted performance. Authors have implemented a circular data structure which acts as a buffer queue where data can be saved and retrieved easily using multithreading techniques.

The prototype has been evaluated on a demonstration web services-enabled test rig which mimics a real-world manufacturing production line. A client_instance tool has been developed for evaluating the performance of the prototype in a distributed manufacturing scenario. It has been observed that on average, the prototype can propagate machine information reliably to approximately 200 client resources with an acceptable performance. The current implementation only allows TCP based communications due to the strong integration of TCP/IP with event notifications. The evaluation of the above solution to a PLC-based control system is under progress. Furthermore, a substantial amount of research work is to be carried out when implementing such a solution to a real world manufacturing practice such as the Powertrain assembly line.

Such an implementation to a manufacturing enterprise is beneficial as it provides new ways to support machines regardless of their locations. It also establishes new relationships between supply-chain CB implementation partners and makes information access more efficient. The key aspect of such an implementation stems from the provision of open vendor independent solutions, supporting wide array of mechanisms in a distributed manufacturing environment.

ACKNOWLEDGMENTS

The authors would like to acknowledge EPSRC, Loughborough IMCRC and EU SOCRADES framework 6 research project and other collaborating companies and colleagues for their kind support.

REFERENCES

[1]. Harrison, R., A.A.West., R.H.Weston, and R.P. Monfared, *Distributed engineering of manufacturing machines.* Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 2001. **215**(2): p. 217-231.

[2]. Hao, Q., W. Shen, and L. Wang, *Towards a cooperative distributed manufacturing management framework.* Computers in Industry, 2005. **56**(1): p. 71-84.

[3]. Wang, L., P.Orban, A. Cunningham, and S.Lang, *Remote real-time CNC machining for web-based manufacturing.* Robotics and Computer Integrated Manufacturing, 2004. **20**(6): p. 563-571.

[4]. Linthicum, D.S., *Enterprise application integration*. 2000.

[5]. Shaw, M. and D. Garlan, *Software architecture: perspectives on an emerging discipline*. 1996: Prentice-Hall, Inc. Upper Saddle River, NJ, USA.

[6]. Bosch, J., *Software Architecture: The Next Step.* LECTURE NOTES IN COMPUTER SCIENCE, 2004: p. 194-199.

[7]. Harrison, R., A.W.Colombo, A.A.West, and S.M.Lee, *Reconfigurable modular automation systems for automotive power-train manufacture.* International Journal of Flexible Manufacturing Systems, 2006. **18**(3): p. 175-190.

[8]. Harrison, R., S.M. Lee, and A.A. West, *Lifecycle engineering of modular automated machines.* 2nd IEEE International Conference on Industrial Informatics, 2004: p. 501-506.

[9]. Sommerville, I., *Software Engineering*. 8th ed. 2007, Reading, Massachusetts: Addison-Wesley Publishing Company.

[10]. Erl, T., *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. 2004: Prentice Hall PTR Upper Saddle River, NJ, USA.

[11]. Barot, V. and J.Carter, *Design and Development of a Judicial Advisory Expert System (JAES) to Resolve Legal SGA Ownership Dispute Cases*, in *Proceedings of the 2008 UK Workshop on Computational Intelligence (UKCI08)*. September 2008: Leicester, UK.

[12]. Dunkels, A., O. Schmidt, T.Voigt, and M.Ali, *Protothreads: simplifying event-driven programming of memory-constrained embedded systems*. 2006: ACM Press New York, NY, USA.

[13]. Garlan, D., G.E. Kaiser, and D. Notkin, *Using tool abstraction to compose systems.* IEEE Computer, 1992. **25**(6): p. 30-38.

[14]. Philip, G.C., *Software design guidelines for event-driven programming.* The Journal of Systems & Software, 1998. **41**(2): p. 79-91.

[15]. Welsh, M., D. Culler, and E. Brewer, *SEDA: An Architecture for Well-Conditioned, Scalable Internet Services.* 2000.

[16]. Lee, S.M., R. Harrison, and A.A. West, *A component-based control system for agile manufacturing.* Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 2005. **219**(1): p. 123-135.

[17]. Colombo, A.W., *SOCRADES. http://www.socrades.eu/Documents/objects/file1224780946.72*, Accessed on: Jan 2009.

[18]. Barot, V., C.S.Mcleod., R.Harrison, and A.A.West, *Efficient real-time remote data propagation mechanism for a Component-Based approach to distributed manufacturing.* in *International Conference on Manufacturing Systems Engineering*. April 2009: Rome, Italy, in press.

[19]. COMPAG, *COMponent Based Paradigm for AGile Automation.* Loughborough University research project.

[20]. Lee, L.J., *A Next Generation Manufacturing Control System.* PhD dissertation, MSI Research Institute, Loughborough University (internal), 2003.