Appendix A

The Pilot Systems

Our involvement in the HMS project began during the Feasibility Study, with the development of fault diagnosis systems for a steel rod cooling mill (McFarlane et al, 1998) and an automotive assembly line (the VQ Shuttle). The model-based diagnostic strategy that was developed for the assembly line is described in (Jarvis, D. and Jarvis, J., 2003). The underlying models from that project were used to automatically construct a skeletal holonic control system (Jarvis, J. and Jarvis, D., 1998). The intent was that this would provide a migration path from PLC to holonic control. However, that approach was discontinued with the development of Part Oriented Control (Gayed et al, 1998) in Phase 1 of the HMS Project.

Part Oriented Control was the first holonic execution system to be developed that utilised existing PLC controller technology and was a key outcome of the HMS Phase 1 project. The technical feasibility of the concept was demonstrated using an industrial-strength robotic assembly cell located at the University of Cambridge. It is described in Section 5.1. Part Oriented Control was evaluated for three pilot systems within the HEO control systems test bed of Work Package 3 and was implemented for the Huller-Hille Machine (Jarvis, J. et al, 2003).

The collaboration with the University of Cambridge has had a focus of manufacturing execution and was conducted outside of the HMS project. It has resulted in the development of a number of execution systems for the robotic assembly cell described in Section 5.1 and culminating with those presented in this book. Also developed was an execution system for an agent-based packing cell. Both systems were located in the Institute for Manufacturing at the University of Cambridge.

Our research has been conducted within three major research projects

1. Model-Based Diagnosis (MBD)
2. HMS WP3 (HMS)
3. Holonic Manufacturing Execution (HME)

and has involved the following organisations:

- Holden Vehicle Manufacturing Operations (HVMO)
- Holden Engine Operations (HEO)
- Commonwealth Scientific and Industrial Research Organisation (CSIRO)
- University of South Australia (UniSA)
- University of Cambridge (Cambridge)
- Central Queensland University (CQU)
- Intendico Pty. Ltd.
- Agent Oriented Software Group (AOS)

The following pilot systems were used to develop execution systems:

The individual Pilot Systems are described in Table A.1.

A.1 Model Based Diagnosis

A.1.1 VQ Shuttle

The VQ Shuttle was a low-volume vehicle assembly line. It consisted of three assembly stations (Stations 10, 20 and 30) linked by a transfer line. It had a well-defined hierarchical structure, making it ideal for an exploration of holonic diagnosis. Two types of bodies were assembled on the line (Style A and Style B). The line was controlled by a single PLC with approximately 700 I/O points. The control logic was similar to that of the larger systems within the company.

The operations performed at each station were an alternating sequence of automatic steps (known as stages) and manual steps. Automatic operation at

Table A.1. The Pilot Systems

Project	**Pilot System**	**Partners**	**Reference**
MBD	VQ Shuttle	HVMO, CSIRO, UniSA	Jarvis, D. and Jarvis, J., 2003
HMS	Huller-Hille Machine	HEO, CSIRO, UniSA	Jarvis, J. et al, 2003
HMS	Wilson Line	HEO, CSIRO, UniSA	Jarvis, J. et al, 2003
HMS	Sorting Conveyor	HEO, CSIRO, UniSA	Jarvis, J. et al, 2003
HME	Meter Box Cell	Cambridge, CSIRO, UniSA, AOS	Jarvis, D. et al, 2001; Jarvis, J. et al, 2006c
HME	Packing Cell	Cambridge, AOS	Evertsz et al, 2004; Fletcher et al, 2003
HME	Meter Box Cell	Intendico, CQU, UniSA	

a station was initiated by an operator (or operators) depressing one or more sets of palm buttons. Those buttons remained depressed for the duration of that step. The activities performed during automatic operation typically involved the opening and closing of clamps. On completion of an automatic step (indicated by a lamp being illuminated on the station control panel), the operator removed his or her hands from the palm buttons and initiated a sequence of manual activities. These typically involved the loading of panels and/or welding. When the assembly operations had been completed at a station, the operator needed to wait until assembly has been completed in the other stations. At that point, the transfer line was activated (by the operators at each station simultaneously depressing their palm buttons) and the station cycle began again with a new body. Note that the number of stages in a cycle depended on the model that was being built.

A.2 HMS WP3

HEO identified three pilot systems – the Huller-Hille machine, the Wilson line and the Engine Sorting Conveyor – as representatives for examining the feasibility of holonic execution within the HMS WP3 project. In addition, a fourth system, the meter box assembly cell, described in Section 5.1, was used for testing purposes.

A.2.1 Huller-Hille Machine

The Huller-Hille machine performed the preliminary machining for crankshafts. It consisted of a transfer line and a machining capability. From a part perspective a crankshaft was:

1. transferred into a CNC machining location
2. rotated at high speed
3. machined
4. transferred out of the machining location

The Huller-Hille Machine had 8 stations with the machining operations occurring at stations 4 and 5. The machine had buffers (in the form of a part conveyor) both at its input and output stations. To decrease latency, the system was configured to machine two parts simultaneously. Availability of a part was sensed on the input queue by a part present limit switch. The loading device then transferred the part into station 1 of the machine, where the part was automatically orientated. Stations 2 and 3 formed a two part input buffer within the machine. In station 3 there is a set of 3 proximity switches. Each proximity switch is used to identify a different part type, thus enabling a check on the part about to enter Station 4 for machining.

Stations 4 and 5 were where the machining operations occurred. There were two levels to each of these stations. The lower level was at the same

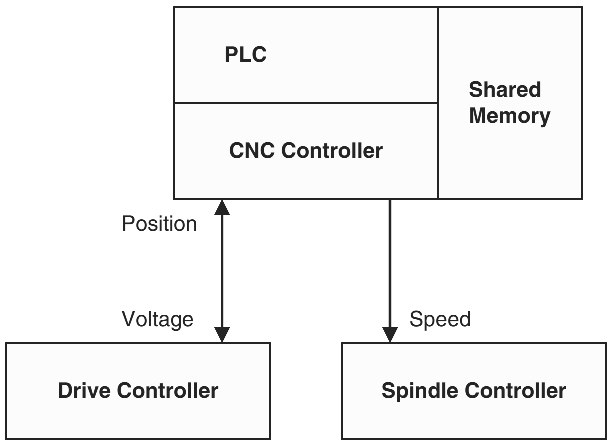

Fig. A.1. Original control architecture for the Huller-Hille machine

elevation as the other stations in the transfer mechanism. The upper level of each of these stations was directly above the transfer level station and is where the part was clamped into the spindle chucks, spun up to speed and machined. The part was raised from the lower to the upper sections using a two stage lifting hydraulic. The spindle chucks incorporated clamping mechanisms to hold the part and a counter weight to balance the rotational mass. The counter weight compensated for the off-centre spinning of the part so that vibration due to rotation is minimal. The remaining stations formed an internal output buffer.

The original control architecture is summarised in Figure A.1. Part transfer was managed by the PLC; machining was managed predominantly by the CNC controller in conjunction with the drive and spindle controllers[1]. The PLC controlled the whole process; synchronisation between the PLC and CNC was realised using shared memory.

A.2.2 Wilson Line

The Wilson line was responsible for the assembly and testing of 1.6 litre, 1.8 litre and 2.0 litre single overhead cam heads. The line was structured as a chain loop transfer with multiple assembly/test stations, some manual and some fully automatic. There was also a part repair loop associated with the final stations - faulty parts were transferred to the repair loop and then returned to the line. In total, there were 33 assembly stations, 2 test stations and 1 repair station. Two PLCs controlled the line - one controlling the assembly line and one controlling the repair loop.

[1] There is some discrete i/o associated with the drives and spindles (such as limit switches that are controlled by the PLC).

A.2.3 Sorting Conveyor

Engines produced at HEO were packaged prior to dispatch to domestic and export customers. However the engines first needed to be sorted into batches containing the same engine model. As the production process was not driven by the packaging area this function was achieved by the sorting conveyor system. Engines entered the system at the loading area. An engine was loaded onto an empty carrier; a switch was then manually set on the carrier to indicate the engine type. This setting was read at various points in the system. Carriers circulated continuously in the system, regardless of whether they were holding an engine – there were approximately 60 carriers in the system. The system was composed of a closed network of conveyor sections. Each section had a maximum capacity, which may have been less than the capacity calculated from its length. Each section had a mechanism for detecting carrier entry; most had a mechanism for controlling carrier exit.

Sorting was achieved by detecting the presence of a batch quantity in the conveyor system (5 for domestic, 4 for export) and then assigning those engines to one of 4 holding queues. The batch was then dispatched from its holding queue to the appropriate packaging station (domestic or export). There were in excess of 90 different engine models that could be produced. On any given day, approximately 10–12 of these models were produced with a volume of around 800 engines per shift. The system was controlled by a single PLC.

A.3 Holonic Manufacturing Execution

A.3.1 Meter Box Cell

The meter box cell was a robotic assembly cell designed to assemble meter boxes. It was located at the Institute for Manufacturing, University of Cambridge and is described in detail in Section 5.1.

A.3.2 Packing Cell

This cell was located at the Institute for Manufacturing, University of Cambridge. It is illustrated in Figure A.2, and was comprised of a storage unit to hold items, a Fanuc M6i robot to move items throughout the cell, a Montrack^{TM} shuttle conveyor to transport batches of raw materials (items) and the boxes into which items were to be placed around the cell. The plan view of the system is given in Figure A.3. The cell had three conveyor loops and independently controlled gates that guided shuttle navigation around the track. There were also two docking stations where shuttles were held so that the robot could pick and place items into the boxes that they carried.

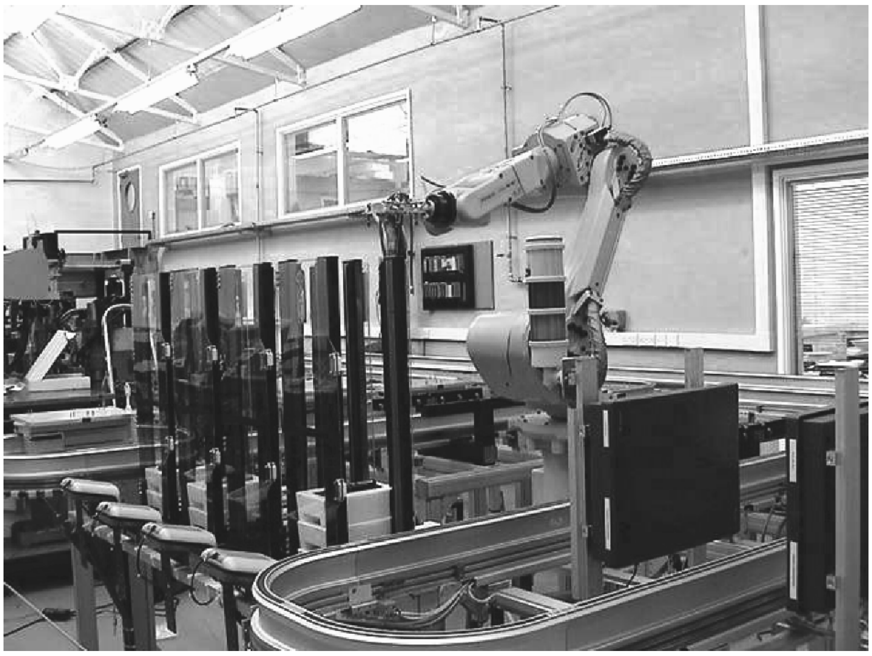

Fig. A.2. The Packing Cell

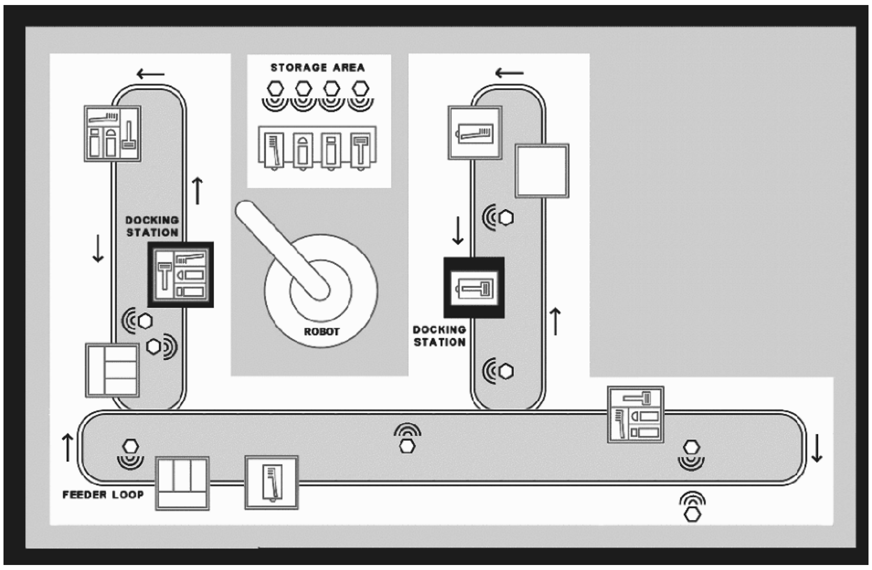

Fig. A.3. Layout of the Packing Cell

Some agile/intelligent manufacturing functions that the cell demonstrated included:

1. Self-management of resources by individual products. This included the acquisition of trays, shuttles and items.
2. Cooperation between docking stations to dynamically distribute workloads.
3. Using boxes allocated to non-urgent orders for rush orders when no free boxes were available.
4. Rescheduling arising from the filling of rush orders.
5. Routing of shuttles to minimize transit times and to give priority to those associated with rush orders.
6. Dynamic adaptation to docking station failure.
7. Pro-active storage unit organization. During otherwise idle periods, the robot would reorganise items in the chute so that required items or the most frequently requested items were at the head of each chute.

Appendix B

GORITE

There are two sides to GORITE: process modelling and belief modelling.

1. **Process Modelling** is done using a *Team Oriented Paradigm*, which means that a system is viewed in terms of the orchestrated behaviour of goal oriented teams. An application is built by defining the teams, identifying what capabilities they have / goals they can perform, enumerating the various ways in which goals may be achieved, and specifying the reasoning that is required in order to achieve those goals.
2. **Belief Modelling** is done using a *epistemological paradigm*, which means that data is generally understood as being or representing *beliefs*; the beliefs of the teams and performers. The programmer is firstly concerned with the *forms* of beliefs that can be had, and secondly with the questions of how belief structures are populated and later updated to follow the developing situation.

The key modeling classes are described below. The intent of the following sections are twofold:

1. To indicate the scope of the GORITE modelling framework and
2. To provide sufficient detail to understand the code that is presented in Section 5.4

No usage examples are given; the user is referred to (Rönnquist, 2008) for a fully worked example.

B.1 Process Modeling Classes

`Action`

The Action class is a utility base class for implementation of task goals that operate on a common entity. The created action object becomes a factory for goals that refer back to it, and that end up invoking the action's `execute()`

method as way of achieving the task goal. Generally, an action is a named definition of a function performed on data. It would occur in a process model in terms of a task goal, which is recognised as a usage point of the action, linking it to particular input and output data. The same action may be used elsewhere, with other data names. Task goals referring to the same action object will invoke the same method.

Capability

A `Capability` is a container of `Goal`s, which are accessible by their names, and understood to represent alternative ways in which that named goal may be achieved. It is also typically used as base class for containing belief structures when required.

Context

This is an interface that in particular is implemented by the `Plan` class. It is recognised by the BDI type `Goal` processing for providing the `Query` that defines alternative invocation contexts for a `Goal`.

Data, Data.Element

The `Data` class is a container class for the data elements (of type `Data.Element`) that goals use and produce. The underlying modelling concept is that these elements are accessible to all sub-goal executions for a particular goal performance. In this regard, the elements can be viewed as providing an evolving situational context for goal execution. Element references are made available to a task goal via two parameters in its `execute()` method – one parameter specifies the list of input elements to be used in goal achievement and another the output elements produced by goal achievement. Data elements can be multi-valued – in parallel execution (such as `Repeat` goals) different values for the same data element can then be bound to each branch. Individual performers are free to maintain their own private data or beliefs, using whatever representation is most appropriate. GORITE provides support for a relational belief model that can be used for belief management.

Executor

This is a special class, which implements an infinite loop for running the `Performer.TodoGroups` of a collection of `Performers`.

Goal, Goal.Types, Goal.States

The `Goal` class is used for modelling the decomposition of a goal as an abstraction of sub goals. Goals are performed by means of performing their sub goals in certain ways, with the exceptions of task goals, which are leaf nodes in goal hierarchies. The types of goals, enumerated by `Goal.Types`, are listed in Table B.1.

`Goal.States` is an `enum` for the values that the `Goal` execution method may return, apart from a thrown `Goal.LoopEndException`. Its values are listed in Table B.2.

Performer

The `Performer` class is intended to be base class for team member implementations. Conceptually it is considered the agent for goal executions, and it contains the belief structures, if any, that agent reasoning may refer to. The class extends `Capability` by including `Performer.TodoGroup` objects, and a method `performGoal()` for performing goals synchronously.

Performer.TodoGroup

`Performer.TodoGroup` is a representation class keeping goal executions according to their registered groups. To this end, a goal has an optional `group` attribute that names which todo group the goal belongs to. The todo group is seen as a stack of goal executions, where by default new executions are added to the end (first in, first out), and only the first execution of the stack is progressed. When the first execution terminates (`PASSED, FAILED` or `CANCEL`) it gets removed from the stack, allowing another execution to be progressed. When executions are added to or removed from the todo group, a user provided *meta-goal* can be invoked in order to review the current stack, and possibly promote any pending execution to be the one to progress next.

Plan

This is a representation class for goals that implement `Context` and `Precedence. Plan` objects typically occur only as top level objects of goal hierarchies.

Precedence

This is an interface that is implemented by the `Plan` class. It is recognised by the BDI type goal processing as providing a weight for its associated goal hierarchy. Higher precedence goal hierarchies are attempted before lower precedence goal hierarchies. Goals without explicit precedence weight get a default value of `Goal.DEFAULT_PRECEDENCE`, which is 5.

Table B.1. Goal types

Value	Description
`Sequence`	A sequence goal is achieved by achieving its sub goals in order. The goal fails when any sub goal fails.
`Condition`	A condition goal is achieved by achieving any of its sub goals, which are attempted in sequence. The goal fails only if all sub goals have been attempted and have failed.
`Fail`	A fail goal is achieved by means of failing any of its sub goals, which are attempted in sequence. The goal fails when and if all sub goals have been attempted and succeeded. In particular, a fail goal without sub goals always fails immediately, and is a way to induce failure. Failure may also be induced by tasks, which can succeed or fail.
`Parallel`	A parallel goal is achieved by achieving all its sub goals in parallel. The goal fails if any of its sub goals have failed fail, once all sub goals have terminated.
`Repeat`	A repeat goal is achieved in the same way as a sequence goal, but by instantiating it over a multi-valued data element, so that each instantiation gets a separate data context focusing on one of the values for that data element. The instantiations are processed as parallel branches, and the repeat goal succeeds when all branches have succeeded. If a branch fails, then the repeat goal fails, and all other branches are cancelled.
`Loop`	A loop goal is achieved by repeating its sub goals as a sequence goal again and again until somewhere in a sub goal hierarchy an end goal breaks the loop. The loop goal fails immediately if a sub goal fails, and it succeeds when an inner end goal breaks the loop.
`End`	An end goal has loop control semantics in addition to normal success and failure. If all its sub goals succeed when attempted one by one in sequence, like a sequence goal, then the end goal breaks an enclosing loop goal. Otherwise, when a sub goal fails, then the end goal succeeds without breaking the loop. An end goal never fails.
`Control`	A control goal has a control semantics similar to an end goal, but for parallel goal executions. Its sub goals are performed in sequence, and if all succeed, then the control goal causes the enclosing parallel execution to cancel all branches and succeed. The parallel executions include the parallel goal, the repeat goal and the implicit parallel branches of a multi-filled role.
`Task`	A task goal is achieved by invoking its `execute()` method, which is Java code that returns a value of type `Goal.States`.
`BDI`	A BDI goal is a task goal that is achieved by finding and performing a goal hierarchy as named by the BDI goal. The BDI goal expects a `Data.Element` object to provide the Capability in which to find one or more goal hierarchies of the given name, and these are performed one by one until one succeeds. The `control` attribute of the BDI goal is used for naming the data element concerned, which would be either be an allocated performer or the name of a role.

Table B.2. Goal states

Value	Description
`PASSED`	Goal execution terminated successfully.
`FAILED`	Goal execution terminated in failure.
`STOPPED`	Goal execution seemingly was interrupted, and is pending.
`BLOCKED`	Goal execution seemingly was suspended, and is pending.

Team

`Team` is a base class for team classes. The class extends `Performer`, and thereby `Capability`, by being a container of `Performer` objects and providing notions (`Team.Role` and `Team.TaskTeam`) for defining orchestrated operations for its performers.

Team.Role

The `Role` class is a base class for representing team members in a team. `Role` extends `Capability`, so as to hold `Goal` methods that are tied to the role within the context of a team. It further contains an attribute `required`, which indicate goals that are required by a `Performer` of the role.

Team.TaskTeam

`Team.TaskTeam` is a representation class for defining the set of roles required to achieve a coordination goal. A `TaskTeam` object is configured with `Role` objects that define the positions involved. It is then populated with `Performer` objects according to how they fill the roles in the task team. Thereafter coordination goals deploy the task team by establishing the role filling within their `Data` objects.

B.2 Belief Modelling Classes

And

This is a representation class for a conjunctive form of `Query` objects. It makes a `Query` object that provides a left-to-right short-cut conjunctive evaluation of its member `Query` objects.

Condition

This is a base class for defining a query that involves a boolean computation.

Distinct

This is a representation class of a query to recognise distinction in the values of two or more `Ref` objects, i.e., they must all have different values.

Or

This is a representation class for a disjunctive form of `Query` objects. It makes a `Query` object that provides a left-to-right short-cut disjunctive evaluation of its member `Query` objects.

Query

This is an interface for belief query processing separately from the belief sources. A `Query` object is seen as providing a series of value combinations for its collection of `Ref` objects, namely successive combinations for which the query is true. The `Query` interface includes support for establishing observers on belief sources, to allow the set up of processing a query on demand, and it includes support for belief update via the query.

QueryBase

This is a utility class for simplifying the implementation of the `Query` interface as an inline class or a separate class, by providing default implementations.

Ref

This is a representation class for carrying the values obtained by query processing.

Relation

This is a representation class for relational belief modelling. A `Relation` is created for a given set of columns, by indicating the types of the values in the columns. It is then set up with *key constraints* as needed, where a key constraint indicates a sub set of columns such that all rows stored must differ in those columns. As a row is added, all existing rows that are the same according to some key gets removed. The `Relation.get` method returns a `Query` for obtaining the matching rows of a given combination of inputs.

Snapshot

This is a representation class for a query that takes snap shots of a given query, and only tells about new results since the prior snap shot. A `Snapshot` object tracks a given query by collecting the valid combinations of `Ref` object values, and remembering these combinations when it is reset.

When

This is a representation class for a query that suspends the calling thread instead of failing. A `When` object is given a query to monitor, and will resume the calling thread when there are new results from the monitored query.

References

Agent Oriented Software. (2008a) "AOS Autonomous Decision-Making Software: About", http://www.agent-software.com/index.html.

Agent Oriented Software. (2008b) "AOS Autonomous Decision-Making Software: Products", http://www.agent-software.com/products/index.html.

Anderson, J. and Lebiere, C. (1998) *The Atomic Components of Thought.* Lawrence Erlbaum Associates.

Bass, L., Clements, P. and Kazman, R. (2003) *Software Architecture in Practice (2nd Edition)*, Addison-Wesley.

Beach, L.R. and Connolly, T. (2005) *The Psychology of Decision making: People in Organizatiions 2nd. Edition*, SAGE Publications.

Bratman, M.E. (1987) *Intention, Plans, and Practical Reasoning*, Harvard University Press, Cambridge, MA (USA).

Bratman, M.E., Israel, D.J. and Pollack, M.E. (1988) "Plans and resource-bounded practical reasoning", Computational Intelligence, Vol. 4, No. 4.

Bratman, M. (1999) *Faces of Intention: Selected Essays on Intention and Agency*, Cambridge University Press.

Brennan, R. and Norrie, D. (2003) "From FMS to HMS", in Deen, M. (Ed), *Agent Based Manufacturing. Advances in the Holonic Approach.* Springer, 2003.

Brooks, R. (1999) *Cambrian Intelligence: The Early History of the New AI*, MIT Press.

Buchanan, B. and Shortliffe, E. (1984) *Rule Based Expert Systems*, Addison-Wesley.

Bussman, S. and Sieverding, J. (2001) "Holonic Control of an Engine Assembly Plant – an Industrial Evaluation" in Proceedings of the 2001 IEEE International Conference on Systems, Man and Cybernetics, Tucson AZ, USA, October 2001.

Bussmann, S., Jennings, N. and Wooldridge, M. (2004) *Multiagent Systems for Manufacturing Control. A Design Methodology.* Springer.

Card, S., Moran, T. and Newell, A. (1983) *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum.

Chirn, J. (2002) "A Holonic Component Based Approach to Manufacturing Control Systems", PhD Thesis, University of Cambridge.

Chirn, J. and McFarlane, D. (2000) "Application of the Holonic Component Based Approach to the Control of a Robotic Assembly Cell", in Proceedings of IEEE Conference on Robotics and Automation, San Francisco.

Christensen, J. (2003) "HMS/FB Architecture and its Implementation", in Deen, M. (Ed.), *Agent Based Manufacturing. Advances in the Holonic Approach.* Springer.

Cirocco, L., Jarvis, D., Jarvis, J., Lovrenich, R. and Matson, J. (1999) "Zone Logic: A Declarative Approach to the Agent Based Control of Discrete Part Manufacturing Systems", in H. Van Brussel and P. Valckenaers, Eds., Proceedings of 2nd International Workshop on Intelligent Manufacturing Systems, Leuven, Belgium.

Cohen, P. and Levesque, H. (1991) "Teamwork", Nous, Vol. 25, No. 4, pp. 487–512.

Connell, R., Lui, F., Jarvis, D. and Watson, M. (2003) "The Mapping of Courses of Action Derived from Cognitive Work Analysis to Agent Behaviours", in *Proceedings of Agents at Work: Deployed Applications of Autonomous Agents and Multi-agent Systems Workshop*, Second International Joint Conference on Autonomous Agents and Multi Agent Systems, Melbourne.

Coram, R. (2002) *Boyd: The Fighter Pilot Who Changed the Art of War*, Little, Brown and Company.

Decker, K. (2004) "A Vision for Multi-agent Systems Programming", in Dastani, M., Dix, J. and El Fallah Seghrouchni, A. (Eds.), Lecture Notes in Artificial Intelligence, 3067. Springer.

d'Inverno, M., Kinny, D., Luck, M. and Wooldridge, M. (1997) "A formal specification of dMARS", in Singh M.P., Rao A.S., and Wooldridge M., editors, Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag LNAI 1365 pp. 155–176.

Evertsz, R., Fletcher, M., Jones, R., Jarvis, J., Brusey, J., and Dance, S. (2004) "Implementing Industrial Multi-agent Systems Using JACK$^{\text{TM}}$", in Dastani, M., Dix, J. and El Fallah Seghrouchni, A. (Eds.), Lecture Notes in Artificial Intelligence 3067, Springer.

Fletcher, M., Lucas, A., Jarvis, D., Jarvis, J., Rönnquist, R., McFarlane, D., Brusey, J. and Thorne, A. (2003) "JACK-based Holonic Control of a Gift Box Packing Cell", Technical Report of the Centre for Distributed Automation and Control, Institute for Manufacturing, University of Cambridge.

Forgy, C. and McDermott, J. (1977) "OPS, a Domain-Independent Production System Language", Proceedings of the Fifth International Joint Conference on AI, pp. 933–939.

Gayed, N., Jarvis, D. and Jarvis, J. (1998) “A Strategy for the Migration of Existing Manufacturing Systems to Holonic Systems”, In Proceedings of SMC 98, San Diego.

Georgeff, M.P. and Lansky, A.L. (1986) “Procedural Knowledge”, in Proceedings of the IEEE Special Issue on Knowledge Representation, pp. 1383–1398.

Georgeff, M.P. and Ingrand, F.F. (1989) “Monitoring and Control of Spacecraft Systems using Procedural Reasoning”, in Proceedings of the Space Operations Automation and Robotics Workshop.

Grosz, B. and Kraus, S. (1996) “Collaborative Plans for Complex Group Action”, Artificial Intelligence, Vol. 86, pp. 269–357.

Gruver, W., Kotak, D., van Leeuwen, E. and Norrie, D. (2003) “Holonic Manufacturing Systems – Phase II”, in Proceedings of the International Conference on the Industrial Applications Of Holonic and Multi-Agent Systens (HoloMAS 2003), Prague, September 2003.

Heikkilä, T., Rannanjärvi, L., Sallinen, M. and Rintala, M. (2003) “A Holonic Shot-Blasting System”, in Deen, M. (Ed.), *Agent Based Manufacturing. Advances in the Holonic Approach.* Springer.

Heinze, C., Goss, S., Josefsson, T., Bennett, K., Waugh, S., Lloyd, I., Murray, G. and Oldfield, J. (2002) “Interchanging Agents and Humans in Military Simulation”, AI Magazine, Vol. 23, No. 2, pp. 37–48.

Holonic Manufacturing Systems Consortium, (2001) “Holonic Manufacturing Systems Overview”, http://hms.ifw.uni-hannover.de.

IEEE Computer Society (2006) “The Foundation for Intelligent Physical Agents”, http://www.fipa.org.

IMS International (2006). “Intelligent Manufacturing Systems”, http://www.ims.org

JADE, (2006) “Java Agent Development Framework”, http://jade.tilab.com.

Jarvis, B., Corbett, D. and Jain, L. (2005) “Beyond Trust: A Belief-Desire-Intention Model of Confidence in an Agent’s Intentions”, in Proceedings of KES 2005, Melbourne.

Jarvis, D., Jarvis, J., Lucas, A. and Rönnquist, R. (2001) “Implementing a Multi-agent Systems Approach to Collaborative Autonomous Manufacturing Operations”, in Proceedings of the 2001 IEEE Conference on Systems, Man and Cybernetics, Tucson AZ, USA, October 2001.

Jarvis, D. and Jarvis, J. (2003) “Holonic Diagnosis for an Automotive Assembly Line”, in Deen, M. (Ed), Agent Based Manufacturing. Advances in the Holonic Approach. Springer.

Jarvis, D., Jarvis, J. and Rönnquist, R. (in press) “Using Agent Teams to Model Virtual Enterprises”, Multiagent and Grid Systems.

Jarvis, J. and Jarvis, D. (1998) “Design Recovery for PLC Controlled Manufacturing Systems”, In Kopacek, P. (Ed.), *Manufacturing Systems: Modelling, Management and Control*, Elsevier.

Jarvis, J., Jarvis, D. and McFarlane, D. (2003) “Achieving Holonic Control – an Incremental Approach”, Computers in Industry, Vol. 51, pp. 211–223.

Jarvis, J., Jarvis, D., Rönnquist, R. and Fletcher, M. (2006a) “Provision of Robust Behaviour in Teams of UAVs – a Conceptual Model”, in Proceedings of 1st SEAS DTC Technical Conference, Edinburgh.

Jarvis, J., Jarvis, D., Rönnquist, R. and Fletcher, M. (2006b) “Towards a Reference Architecture for Agent-Based Power Management for Unmanned Vehicles”, in Proceedings of 1st SEAS DTC Technical Conference, Edinburgh.

Jarvis, J., Rönnquist, R., McFarlane, D., and Jain, L. (2006c) “A Team-Based Approach to Robotic Assembly Cell Control”, Journal of Network and Computer Applications, Vol. 29, pp. 160–176.

Jones, R., Laird, J., Nielsen, P., Coulter, K., Kenny, P. and Koss, V. (1999) “Automated Intelligent Pilots for Combat Flight Simulation”. AI Magazine. Vol. 20, No. 1, pp. 27–41.

Jones, R.M. and Wray, R.E. (2006) “Comparative Analysis of Frameworks for Knowledge-Intensive Intelligent Agents”, AI Magazine, Vol. 27, No. 2, pp. 57–70.

Karim, S. and Heinze, C. (2005) “Experiences with the design and implementation of an agent-based autonomous UAV controller”, in Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi-Agent Systems, Utrecht, The Netherlands.

Klein, G. (1996) *Sources of Power: The Study of Naturalistic Decision Making*, Lawrence Erlbaum.

Koestler, A. (1967) *The Ghost in the Machine*, Arkana, London.

Kollingbaum, M. and Norman, T. (2004) “Norm Adoption and Consistency in the NoA Architecture” in Dastani, M., Dix, J. and El Fallah Seghrouchni, A. (Eds.): Lecture Notes in Artificial Intelligence, 3067. Springer.

Laird, J.E, Newell, A. and Rosenbloom, P. (1987) “Soar: An architecture for General Intelligence”, Artificial Intelligence, Vol. 33, No. 1, pp. 1–64.

Lui, F., Connell, R., Vaughan, J., Jarvis, D. and Jarvis, J. (2002) “An Architecture to Support Autonomous Command Agents in OneSAF Testbed Simulations”, in Proceedings of SimTectT 2002, Melbourne.

Mathieson, I., Dance, S., Padgham, L., Gorman, M., and Winikoff, M. (2004) “An Open Meteorological Alerting System: Issues and Solutions”, in Proceedings of the 27th Australasian Computer Science Conference. The University of Otago, Dunedin, New Zealand.

Maturana, F., Shen, W. and Norrie, D. (1999) “Metamorph: An Adaptive Agent-Based Architecture for Intelligent Manufacturing”, International Journal of Production Research, Vol. 37, No. 10, pp. 2159–2174

McFarlane, D. and Bussmann, S. (2000) “Developments in Holonic Production Planning and Control”, Production Planning & Control, Vol. 11, no. 6, pp. 522–536.

McFarlane, D., Marett, B., Elsley, G., Jarvis, D. and Wilbers, P. (1995) “Application of Holonic Methodologies to Problem Diagnosis in a Steel Rod Mill”, in Proceedings of the 1995 IEEE Conference on Systems, Man and Cybernetics, IEEE.

Newell, A. (1980) “Physical Symbol Systems”, Cognitive Science, Vol. 4, pp. 135–183.

Newell, A. (1982) “The Knowledge Level”, Artificial Intelligence, Vol. 18, No. 1, pp. 87–127.

Newell, A. and Simon, H. (1972) *Human Problem Solving* Prentice-Hall.

Padgham, L. and Winikoff, M. (2004) *Developing Intelligent Agent Systems: A Practical Guide*. Wiley.

Parunak, H.V.D. (1999) “Industrial and Practical Applications of DAI”, in Weiss, G. Ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press.

Pokahr, A., Braubach, L. and Lamersdorf, W. (2005a) “A Flexible BDI Architecture Supporting Extensibility”, In Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2005).

Pokahr, A., Braubach, L. and Lamersdorf, W. (2005b) “Jadex: a BDI Reasoning Engine”, in Bordini, R., Dastani, M., Dix, J., and Seghrouchini, A. (Eds.) *Multi-Agent Programming*, Kluwer Book.

Pynadath, D., Tambe, M., Chauvat, N. and Cavedon, L. (1999) “Toward Team-Oriented Programming, LNCS, Vol. 1757, pp. 233–247.

Rao, A. and Georgeff, M. (1995) “BDI Agents: from theory to practice”, in Proceedings of the 1st Int. Conf. on MAS (ICMAS’95).

Ritter, A., Braatz, A., Höpf, M., Schäffer, C. and Westkämper, E. (2003) “Transport Agents – Specification and Development”, in Deen, M. (Ed.), *Agent Based Manufacturing. Advances in the Holonic Approach.* Springer.

Ritter, F. and Norling, E. (2006) “Including Human Variability in a Cognitive Architecture to Improve Team Simulation”, in Sun, R. (Ed.) *Cognition and multi-agent interaction. From cognitive modeling to social simulation.* Cambridge University Press.

Roberts, R. (1989) *Zone Logic – A Unique Method of Practical Artificial Intelligence*, COMPUTE! Books, Radnor, Pennsylvania.

Rönnquist, R. (2008) “GORITE” http://users.tpg.com.au/rrq/gorite/index.html

Searle, J. (1995) *The Construction of Social Reality*, Allen Lane.

Smith, R. (1980) “The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver”, IEEE Transactions on Computers, Vol. 29, pp. 1104–1113.

Suda, H. (1990) “Future Factory Systems Formulated in Japan”, Techno Japan, Vol. 23, No. 3.

Tamura, S., Seki, T. and Hasegawa, T. (2003) “HMS Development and Implementation Environments”, in Deen, M. (Ed.), *Agent Based Manufacturing. Advances in the Holonic Approach.* Springer.

University of Michigan (2007) “Soar”, http://sitemaker.umich.edu/soar/home/index.html.

Valckenaers, P. and Van Brussel, H. (2005) “Holonic Manufacturing Execution Systems”, CIRP Annals 2005, pp. 427–432.

Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L. and Peeters, P. (1998) "Reference Architecture for Holonic Manufacturing Systems: PROSA", Computers in Industry, Vol. 37, pp. 255–274.

Vaughan, J., Connell, R., Lucas, A. and Ronnquist, R. (2003) "Towards Complex Team Behaviour in Multi-Agent Systems using a Commercial Agent Platform", in *Innovative Concepts for Agent-Based Systems*, Springer, LNCS Vol. 2564, pp. 175–185.

Vicente, K. (1999) *Cognitive Work Analysis: Toward Safe, Productive and Healthy Computer-Based Work*. Lawrence Erlbaum Associates.

Walker, S., Brennan, R. and Norrie, D. (2005) "Holonic Job Shop Scheduling Using a Multiagent System". IEEE Intelligent Systems, Vol. 20, No. 1, pp. 50–57.

Wikimedia Foundation, Inc. (2007) http://www.wikipedia.org.

Wooldridge, M. (2002) *MultiAgent Systems*. Wiley.

Wyns, J. (1999) "Reference Architecture for Holonic Manufacturing Systems", PhD Thesis, Katholieke Universiteit Leuven, Leuven, Belgium.

Yoshikawa, H. (1993) "Intelligent Manufacturing Systems . . . " in Yoshikawa and Goossenaerts (Eds.) *Information Infrastructure Systems for Manufacturing* (IFIP Transactions B-14), p. 19. Elsevier.

Glossary

AGV	Automated Guided Vehicle
API	Application Programming Interface
BBS	BlackBoard System
BDI	Belief, Desire, Intention
C2	Command and Control
CNC	Computer Numerical Control
dMARS	distributed Multi-Agent Reasoning System
FSM	Finite State Machine
FIPA	Foundation for Intelligent Physical Agents
GOMS	Goals, Operators, Methods, Selections
GORITE	Goals, Roles, Intentions and TEams
HCBA	Holonic Component Based Architecture
HCI	Human Computer Interaction
HMS	Holonic Manufacturing Systems
IDE	Integrated Development Environment
IMS	Intelligent Manufacturing Systems
JACOB	JACOB^{TM} Object Modeller
JADE	Java Agent Development Framework
JACK	JACK^{TM} Intelligent Agents
JACK Teams	JACK^{TM} Teams
JAL	JACK^{TM} Agent Language
JACK Sim	JACK^{TM} Sim
JVM	Java Virtual Machine
OODA	Observe Orient Decide Act
PLC	Programmable Logic Controller
POC	Part Oriented Control
PROSA	Product Resource Order Staff Agents
PRS	Procedural Reasoning System
RFID	Radio Frequency Identification

SME	Subject Matter Expert
TOP	Team Oriented Programming
UDP	User Datagram Protocol
UAV	Unmanned Air Vehicle