Engineering manufacturing control systems using PROSA and delegate MAS

Paul Verstraete, Bart Saint Germain, Paul Valckenaers, Hendrik Van Brussel, Jan Van Belle, Hadeli

Department of Mechanical Engineering, Katholieke Universiteit Leuven, Celestijnenlaan 300B, 3001 Leuven, Belgium

Abstract: This paper presents a systematic description of a reusable software architecture for multi-agent systems in the domain of manufacturing control. The architectural description consolidates the groups' expertise in this area.

Until now, the research took a manufacturing control perspective at multi-agent systems. The research focussed on providing benefits to the manufacturing control domain by designing a novel type of control system.

This paper takes a software architectural perspective at multi-agent manufacturing control. The systematic description specifies a software product line architecture for manufacturing control. The paper describes the assets of the software product line architecture and how the assets can be combined.

Keywords: multi-agent systems, software architecture, manufacturing control


Biographical Notes: Paul Verstraete is a candidate for the PhD degree in mechanical engineering. Since 2003, he is with the Mechanical Engineering department.

Bart Saint Germain is a candidate for the PhD degree in mechanical engineering. Since 2002, he is with the Mechanical Engineering department.
1 Introduction

This paper presents a systematic description of a reusable software architecture for multi-agent systems in the domain of manufacturing control. The architectural description consolidates the groups' expertise in this area. This consolidation is the contribution of the paper.

Scope  The focus of this paper lies at the intersection of software architecture and multi-agent systems in the context of manufacturing control. The paper investigates the engineering of a multi-agent manufacturing control system.

Several characteristics of the manufacturing control domain - e.g. distribution, dynamism, etc - make applying multi-agent technology in this domain an attractive option. However, manufacturing control also poses several challenges to multi-agent technology, e.g. how to develop a responsive, predictive,... multi-agent manufacturing control system.

Until now, our team took a manufacturing control perspective at multi-agent systems (31; 21; 39; 19). The research focussed on providing benefits to the application domain by designing a novel type of manufacturing control system. The team validated the design on several industrial test cases (8; 39; 21) and developed a test-bed for manufacturing control (33).

This paper takes a software architectural perspective at multi-agent manufacturing control. The paper provides a systematic architectural description based on the existing design. This systematic description was lacking until now. In particular, combining the Product Resource Order Staff reference Architecture (PROSA) and delegate multi-agent system (MAS) into a working manufacturing control system, has not been described in a systematic architectural way.

This architectural description specifies a software product line architecture for manufacturing control. A software product line (5) is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a predescribed way. The market segment in this case is manufacturing control. The products are manufacturing control systems. A software product line architecture is a software architecture that will satisfy the needs of the software product line.

1.1 Characterizing manufacturing control

Manufacturing control applications share a set of common characteristics.

Manufacturing control applications consider a world of interest. Manufacturing control applications consider a part of reality that can be distinguished from the rest of the world, e.g. a factory. Resources (e.g. workstations) in this world of interest are capable of performing operations (e.g. drilling, milling, ...). A control system - in this case a multi-agent system - aims to direct or control the behaviour of the resources to produce goods. The control system is capable of observing the world of interest and issuing commands to the individual resources. The individual agents in the multi-agent control system coordinate amongst each other to determine which commands to send to the resources.
Manufacturing control applications are highly dynamic. Both the world of interest and the multi-agent system undergo frequent changes. Examples of such changes in the world of interest are machine breakdowns, quality deviations, the installation of new machines, delayed operations, etc. A typical change in the multi-agent system is the introduction of new decision algorithms that tackle the prevailing conditions. For instance, throughput is most important when the demand is high. On the contrary, when the demand is low, due-date compliance is a key concern.

Manufacturing control applications are heterogeneous. Manufacturing cases differ in topology types, automatization (presence of sensors, Supervisory Control And Data Acquisition (SCADA (7)) systems, computer systems, ...), processes, manufacturing technology, etc.

1.2 Overview of the paper

This paper describes a software product line architecture for manufacturing control. Section 2 describes the assets of the software product line architecture: a requirements analysis, the PROSA reference architecture, the holonic architectural style and the delegate MAS architectural pattern.

Section 3 discusses how these assets are combined. To clarify the relation between the assets, the section also specifies the internal architecture of an agent and discusses the architecture of an entity in the environment.

Section 4 discusses related work. This section compares the presented agent architecture with the Belief Desire Intention (BDI) agent architecture (20).

Finally, conclusions are given in section 5.

2 Assets of the product line

This section describes the assets of the software product line architecture. The products of the software product line share these assets. The assets are: the requirements, the holonic architectural style, the PROSA reference architecture and the delegate MAS architectural pattern.

2.1 Requirements

This section gives an overview of the common functional and quality requirements for manufacturing control applications. A common set of requirements speeds up the requirement analysis of new applications and disseminates the learned knowledge.

2.1.1 Functional requirements

Functional requirements indicate the systems’ capabilities, services and behaviours (1). They define the tasks the system is expected to perform.

In general the task of manufacturing control applications comprises controlling the behaviour of the resources in the world of interest. This task is subdivided into three subtasks: operations management, tracing and supervising execution. The next paragraphs explain each of these subtasks in more detail.
Operations management determines when and where operations have to be executed in the world of interest. Resources perform these operations. Determining when and where to execute operations is constrained by technological limitations of resources and by process requirements. Common issues are resource capacity problems and resource synchronization.

Tracing the execution of operations involves registering the starting and finishing times of all operations. All relevant data corresponding to the operations (operator, quality of the operation, etc) are monitored. This data is filtered and stored temporarily or permanently depending on the case specific needs.

Supervising execution involves handling unforeseen events by reassigning operations. This reassignment can be automatic (the manufacturing control system takes the initiative) or on request. People interacting with the manufacturing control system are allowed to override decisions. Supervising the execution implies gathering information about the current state of the factory. The control system estimates state information when this information is not directly available.

2.1.2 Quality requirements

According to Weyns (35), quality is the degree to which a system meets the non-functional requirements in the context of the required functionality. In general quality attributes are non-functional properties of a software system such as performance, usability and modifiability.

The software product line architecture considers the following quality requirements:

Performant control refers to the influence the manufacturing control system has on the production performance. By managing the operations, the control system has a direct impact on production and thus on its performance.

The influence the control system has on the production performance is difficult to quantify on beforehand. The manufacturing control system never fully controls the manufacturing system. For instance, breakdowns and late deliveries are not under control of the manufacturing control system. The production performance is directly affected by these uncontrollable events.

Reducing the impact of changes in decision making on the multi-agent control system. Changes in decision making are defined as introducing a new decision mechanism a in the control system. The impact in this context relates to the effort that is required to adapt the software to use the new decision mechanism.

Maintain good connectivity, i.e. how tightly the manufacturing control system is (or can be) connected to the underlying world (26). In manufacturing for example, numerically controlled machines offer limited possibilities for the manufacturing control system to change its operation. This is only possible if an appropriate program is available in the control system. Also, if not enough sensory information is available, guiding the execution gets more difficult (e.g. no information when an operation has finished).

aDecisions in this context are decisions about production activities.
Improve the responsiveness of the manufacturing control system. Responsiveness (17) (sometimes also referred to as reactivity (26)) refers to the ability of a system to respond to disturbances (originating inside or outside the control system) that impact upon net profit. It is required that measures are taken that reduce this impact.

Improve the proactiveness of the manufacturing control system (26). This refers to the ability of the manufacturing control system to plan ahead. Without this ability, the control system is myopic and has limited possibilities to foresee the consequences of its actions. Note that the unpredictability of the controlled environment limits the possibilities of the control system to rely on its proactive abilities.

Depending on the manufacturing case, other requirements can exist.

2.2 Holonic architectural style

An architectural style is a set of constraints on an architecture - on the element types and their patterns of interaction - and these constraints define a set or family of architectures that satisfy them (1).

A holonic architecture is a multi-leveled hierarchy of semi-autonomous sub-wholes, branching into sub-wholes of a lower order. Sub-wholes on any level of the hierarchy are referred to as holons.

The idea of a ‘holon’ was introduced by Arthur Koestler in 'The Ghost in the Machine' (16). The idea was based on an observation of how social and biological systems are organised, based on the 'parable of the two watchmakers' from Herbert Simon (23).

The section first motivates the choice for the holonic architectural style in the manufacturing area. Next it presents a reference model for holons. Finally, it defines the term holon in the context of manufacturing control.

2.2.1 Motivation

Holonic architectures were introduced to provide an answer to some shortcomings in existing control system architectures. Two architectural styles are frequently used: the hierarchical and the heterarchical architectures.

- Hierarchical architectures (9) introduce levels of control and have a pyramidal structure. There are strict master-slave relationships between the elements. Control decisions are operated top-down and status is reported bottom-up. Hierarchical control claims to have a high and predictable performance. Local controllers are helpless when cut-off from their directing supervisors. This makes realizing robustness very difficult and increases the coupling between the different elements. Also, in hierarchical control, decision making is often based on obsolete information.

- Heterarchical architectures (9) allow for full autonomy between the elements in the architecture. The cooperation between elements is arranged via an explicit negotiation procedure (e.g. contract-net protocol (24)) or via indirect
coordination. Full local autonomy is maintained during the cooperative process. This architectural style claims to increase robustness. Because elements function autonomously, they should not fail when other elements malfunction. The disadvantages of this architectural style are reduced predictability of the control system and possible incompatibility issues.

The holonic architectural style contributes to the responsiveness of the control system. The architectural style combines the high and predictable performance promised by the hierarchical architectural style with the robustness against disturbances and the agility of the heterarchical architectural style. A holon functions on its own, which increases robustness (as in the heterarchical architectural style). At the same time, it also functions as part of a bigger whole, forming a hierarchy with other holons for a period of time (similar to the hierarchical architectural style). Higher level holons advise the lower level holons, which is an advantage in handling disturbances.

2.2.2 A reference model for holons

Figure 1  A reference model for holons

This section presents a reference model for holons. A reference model is a division of functionality together with data flow between the pieces. It is a standard decomposition of a known problem into parts that cooperatively solve the problem.

A software architecture maps functionality to a system decomposition. The reference model helps software architects to build concrete software architectures, by offering a standard functional decomposition.

Arising from experience, reference models are a characteristic of mature domains (1). Holons have been introduced in 1967 (16), and have been applied to
The reference model identifies three functionalities that are an essential characteristic of every holon. These functionalities are self-assertion, integration and maintaining its canon. Figure 1 shows the holon reference model. This section first describes these functionalities in general. Next the section describes each of the functionalities from a software architecture point of view. Finally, the section describes the data these functionalities exchange.

General description of the responsibilities Every holon has a dual tendency to preserve and assert its individuality as a quasi-autonomous whole; and to function as an integrated part of an (existing or evolving) larger whole. The polarity between the Self- Assertive (S-A) and Integrative (INT) (16) tendencies is characteristic for a holon. The S-A tendencies are the dynamic expression of the holon’s wholeness, the INT tendencies of its partness.

Holons are governed by fixed sets of rules. The rules - referred to as the system’s canon - determine its invariant properties, its structural configuration and/or functional pattern (16).

Self-Assertion refers to the responsibility of the holon to achieve its goals and to maintain its functioning. A sample goal for a holon who represents a customer order in the context of manufacturing is ‘get produced’. An example of maintaining functionality - in the case of a holon which represents a workstation in a factory - is requesting repair.

Integration refers to the awareness and the evaluation of the holons functioning in a larger whole. The functionality also involves changing the position of the holon in the current holonic hierarchy (= holarchy). An example in manufacturing is a customer order holon observing that its current batch is no longer serving its interest. Changing the position of the holon in the holarchy in this context means joining another batch of orders.

Maintaining canon refers to maintaining the knowledge on the structural properties of the holon and its invariants. For a holon representing a workstation the canon refers to knowledge on the machine specifications and the machine state. For instance, the canon contains knowledge on how fast the workstation should perform an operation and how fast it currently performs the operation in reality. The holon also knows which actions and operations are not possible.

Information exchange between the functionalities Holons on the higher level advise holons at lower levels. For example, a holon representing a factory division advises the workstation holons on what operations they should perform.

The self-assertion and integration functionality influence each other. The self-assertion functionality influences the integration functionality to change the holons place in the holarchy, because the holon cannot function properly as a whole at the moment. The integration functionality influences the goals of the self-assertion functionality. The holons place in the holarchy may change or modify the goals of
the holon. For instance, if a last part of an order is missing to complete the order, a holon representing this missing part can increase its priority to get produced.

Both the self-assertion and the integration functionality use the knowledge maintained by the ‘Maintaining canon’ functionality.

### 2.2.3 Holons in manufacturing

In the context of manufacturing, a holon is defined as an autonomous and cooperative building block of a manufacturing system for transforming, transporting and storing physical and information objects (4). It is software that corresponds to a physical part. The software is responsible for controlling the physical part (3). Remarkable in this definition is that a holon, and thus a holonic manufacturing system, involves both the world of interest and the software. Every holon corresponds to a physical entity in the world of interest. Therefore the architecture is directly influenced, though not necessarily identical, to the system architecture of the physical manufacturing system.

### 2.3 PROSA

#### 2.3.1 Motivation

Using the holonic architectural style requires an identification of the different types of holons. Responsibilities need to be assigned to each holon type. Also relationships between the holons need to be defined clearly.

PROSA is a reference architecture for holonic architectures that reduces the impact of changes in decision making by separating concerns. For instance, changes in decision making in the resource allocation aspect in manufacturing should not affect decision making in the logistics aspect of manufacturing.

The PROSA reference architecture was originally developed in the manufacturing domain. It has also been applied in other application domains (e.g. railway systems (8)).

#### 2.3.2 Architecture

This section presents the PROSA holonic reference architecture (38). The reference architecture defines the various holons in the architecture and the data they share. These holons are composed in a bottom-up manner to create new holons (aggregation). The result of this composition process is the holonic manufacturing system, which is also a holon. In its simplest form, there is one factory holon (a resource holon) that aggregates a number of basic PROSA holons. However, multiple aggregation levels may exist.

The basic holons in the reference architecture represent three separate concerns in the world of interest: logistic management, technological planning and process execution. These holons are called the order, product and resource holon respectively. A fourth optional holon, the staff holon, allows for integration with legacy systems. A module decomposition view of the reference architecture is shown in Figure 2.
The section now discusses each of the basic holons, their responsibilities and the data they share between each other. Finally, there is a short discussion of the reference architecture.

**Resource holons** correspond to a physical part - e.g. a production resource in the manufacturing system - and contain a software part that controls their resource. Resource holons share process execution knowledge with the order holons. For instance, the resource holon share data about processing capacity and processing functionality with the order holons. A resource holon provides the following functionalities.

- A resource holon abstracts the production means. In manufacturing control this holon generalizes machines, conveyors, pallets, raw materials, tool holders, material storage, personnel, floor space, etc.
- A resource holon reflects a physical resource, i.e. contains information about the physical part, the present state, future states and what-if functionality (for more information see Section 2.4)
- A resource holon controls the resource, i.e. starting and stopping an operation, supervising and tracing the execution.
- A resource holon keeps the reflection of the resource state synchronized with the actual resource state.

**Order holons** corresponds to orders in the world of interest. An order generalizes customer orders, make-to-stock orders, prototype-making orders, orders to maintain and repair resources, etc. The order holon is responsible for performing the assigned
work correctly and on time. The order holon is also responsible for reflecting the order.

**Product holon**  Product holons hold the process and product knowledge to assure the correct making of a product with sufficient quality. As such, the product holon is responsible for maintaining the product-model of its product type, not the product-state-model of one order being produced.

**Staff holon**  The staff holons correspond to legacy systems. They are optional.

**Data shared between the holons**  The product and resource holon share process knowledge. Process knowledge contains the information and methods on how to perform a certain process on a certain resource. It is knowledge about the capabilities of the resource, which processes the resource can perform, the relevant process parameters, the process quality, possible outcomes of a process, etc.

The product and order holon share production knowledge. Production knowledge contains the information and methods on how to produce a certain product using certain resources. It is knowledge about the possible sequences of processes to be executed on the resources, about the data structures to represent the outcome of the processes, about the methods to access information of process plans, etc.

Order and resource holons share process execution knowledge. This knowledge contains the information and methods regarding the execution of processes on resources. It is knowledge about how to request the start of a processes on a resource, how to reserve a resource, how to monitor the execution of a process on a resource, how to interrupt a process, the consequences of interrupting a process, how to suspend and resume a processes on a resource, etc.

The staff holons advise the basic holons in their basic operations.

**Discussion**  By placing these three concerns at the same level, the reference architecture differs from traditional manufacturing control. The reference architecture allows to (30):

- Separate the resource allocation aspects from process specific aspects. For instance, in traditional manufacturing control, the process specific aspects are considered before execution. Once execution is started, this information is no longer available. By including this knowledge in the product holon, this information is hidden from the other holons and is still available at execution time. This offers greater flexibility to search for new solutions in the event of production changes or production disturbances.

- Separate the necessary modules, which are generic, from the optional modules, which can be domain specific: order, product and resource holon all hide the specific technical details from each other. For instance, resource holons can easily be specialized to represent manufacturing specific resources (e.g. a rare type of machine). As another example, the introduction of new types of processes does not require modification of the existing holons either (introducing a new product holon). This improves the configurability.
• Separate the structural aspects of the architecture from the algorithmic aspects: existing scheduling and planning algorithms can be integrated without affecting the basic architecture. Scheduling algorithms can be introduced in a distributed way, or in a centralized way by introducing a staff holon (for more information, see (31; 32; 34)).

2.4 Delegate MAS

Delegate MAS is an architectural pattern that allows an agent to delegate a responsibility to a swarm of lightweight agents to support this agent in fulfilling its functions (14). The issuing agent can delegate multiple responsibilities, each of them applying the delegate MAS pattern. The agent may use a combination of delegate multi-agent systems to handle a single responsibility. The delegate MAS may also provide services to other agents.

Delegate MAS is a more generic description of an approach inspired by ant food foraging (25) behaviour used in the research team the past few years (27; 12; 11; 10). The delegate MAS pattern is related to the polyagent approach. A good comparison between delegate MAS and the polyagent approach is described in (18).

The section first motivates the choice for the pattern. Next, the section gives an architectural description of the pattern.

2.4.1 Motivation

As mentioned in the introduction, manufacturing control applications are characterised by their distributed and dynamic nature. The world of interest is complex, i.e. the world of interest contains a lot of specialised and detailed information.

The agent needs to be capable of making decisions that are adapted to the events happening in the world of interest. The delegate MAS pattern allows an agent to exploit the detailed and up-to-date information in the world of interest to adapt his intentions\(^b\) to these events. Furthermore, the pattern allows multiple agents to cooperate in creating this intention. This cooperation reduces the agent complexity and makes the pattern a good solution for the manufacturing control applications.

2.4.2 The architectural pattern

An architectural pattern is a description of element and relation types together with a set of constraints on how they may be used.

The delegate MAS pattern consists out of three elements: the agent, the ant and the environment. The section discusses each of these elements together with the relations between the elements.

Figure 3 shows a module view of the delegate MAS pattern.

---
\(^b\) An intention is a option to which the agent is committed. An option is a plan that contains a number of actions that are performed on a resource in the world of interest. The plan also specifies when the actions start and end.
**Environment** The environment is a software representation of the world of interest. The environment of the targeted applications contains a directed graph, which may change over time. The nodes in the graph represent the resources in the environment and the edges represent the connections between different resources. The environment also contains representative entities for the orders in the factory. An order entity resides on a node in the graph and can communicate with the resource entity on this node. When an order entity is on a particular node, the corresponding resource entity can perform an action of the task of this order entity.

The environment also contains a pheromone infrastructure. Ant agents can deposit, observe and modify pheromones in the pheromone infrastructure. These pheromones can also be transformed by the environment (e.g., ants can insert bookings in the planning scheme of a resource by depositing pheromones).

**Agent** An agent delegates a concern to a swarm of lightweight agents. This delegation involves two main responsibilities: maintaining the swarm of ants and interpreting the results made available by the ants in the environment.

The agent is responsible for maintaining the swarm of ants. This responsibility involves maintaining the population size and the diversity of the swarm. An individual ant is not aware of these swarm properties.

Interpreting the results involves fetching the pheromones on the node on which the agent resides and interpreting these pheromones. The interpretation depends on the responsibility that is delegated by the agent.

Note that delegation is not the same as total dependency. The swarm of ants supports the agent by providing a service. If this service is not delivered properly, the agent should do a best effort to cope on its own.

**Ant** An ant is responsible for executing a task that serves the responsibility of the issuing agent. Typically the ant performs one possible scenario during his life to fulfill the task. For instance, the ant explores one possible solution, the ant follows one possible path to find food, etc.

The ant is created and initialized from the agent and travels autonomously. The ant has its own lifecycle. Each ant may only perform a bounded computational effort within its bounded lifetime and has a bounded footprint (memory).
Relations between the elements  The issuing agent creates and sends ants at regular times. The agent initializes the ant properly. The creation frequency and the initialization is determined for every ant type separately. In this way the population size and the diversity of the swarm is maintained. The ant agents are born at the location where the issuing agent resides and travel over the environment.

The environment enables indirect communication between the issuing agents and the ants. Ants interact with the environment by means of pheromones. Ants from different originating agents or from other generations are able to drop, modify and observe pheromones in the environment. Agents also observe the pheromones in the environment. Some of these pheromones contain the results of the different ants. The agents interpret and adapt their behaviour according to these results.

2.5 Rationale

This section summarizes the qualities described in section 2.1.2 and evaluates how the assets of the product line contribute to these qualities.

Performant control  Both the holonic architectural style and the delegate MAS pattern contribute to this quality requirement.

The holonic architectural style combines the high and predictable performance of the hierarchical architectural style with the robustness against disturbances and the agility of the heterarchical architectural style.

The delegate MAS pattern contributes to the system responsiveness. This responsiveness reduces the effect on the performance by uncontrollable events.

Reducing the impact of changes in decision making  The PROSA reference architecture is designed to provide separation of concerns (cf. section 2.3). For instance, changes in decision making in the resource allocation concern (which is managed by the resource agent) do not affect decision making of the order agent (e.g. in choosing amongst alternative options).

Maintaining good connectivity  The separation between the environment and the agents assures good connectivity. Indeed, no additional constraints are imposed on the agent than constraints that are present in reality. The environment reflects the entities in the world of interest as good as possible. The environment does not enforce decision making strategies; this is the responsibility of the agents. For example, the order agent has the possibility to consider all plans that are technically sound, even if they are not the standard policy of the company. Of course, the decision algorithm of the order agent may take company policies into account.

System responsiveness  The food foraging behaviour of ants is generally recognised to be responsive to events that occur in the environment (2). Since Delegate MAS is inspired on this behaviour, it contributes to the responsiveness of the system to changes in the environment.

System proactiveness  The delegate MAS pattern makes use of a swarm of ants that spread information through the environment. In the software product line this
pattern is used to spread the intentions of the agents in the environment, and thus create short-term local forecasts (27; 29) (see section 2.4 and 2.3). This forecast allows the agents to foresee what is going to happen in the near future.

3 Combining the assets of the product line architecture

This section describes how the assets of the product line architecture are combined to meet the needs of manufacturing control applications.

The section specifies how agents are used together with holons and how delegate MAS relates to the PROSA reference architecture. Next, it specifies the internal architecture of an agent. Finally, it discusses the architecture of an entity in the environment.

3.1 Holon versus Agent

PROSA is a reference architecture for holonic architectures while Delegate MAS is an architectural pattern for MAS. Holons and agents are similar concepts, however they are not equal. Since this paper combines both assets in one architecture, this section clarifies the relation between both.

In section 2.2 a reference model for holons is presented. This section specifies how the functionalities in this reference model map to software architecture.
As mentioned in section 2.2.3, a holon in manufacturing corresponds to a physical part. The reference model identifies three responsibilities: self-assertion, integration and canon maintenance.

The canon of a physical entity in the word of interest determine the rules, invariants, structural properties and the functional pattern of the entity. The canon functionality - when mapped to software architecture - is responsible for estimating the canon of a physical entity as good as possible, and making it available to the rest of the software. This functionality is provided by the environment entity software module.

According to (37) an agent is a computer system that is situated in some environment and that is capable of flexible autonomous action in order to meet its design objectives. The agent module is responsible for the self-assertion and integration functionality of the reference model.

Figure 4 shows the relations between the holon, agent and environment modules. The environment is the aggregate of all environment entities. The agent is situated in the environment. The aggregate of the agents and the environment is called the manufacturing control system.

The agents share information with the environment. The agents can perform actions in this environment (e.g. starting an operation on a machine), but these actions are not guaranteed to succeed. The agents can also perceive the resources in the environment on which they are located (e.g. an order agent can see the order environment entity the order agent reflects, and the resource on which the order environment entity is located). The agent can perceive information about the current state of the resource (e.g. how many orders are at the machine) and about the capabilities of the resource (what happens if I perform an operation with this machine).

The agents can also exchange information by sending messages to each other, mediated by the environment (36).

3.2 Combining the PROSA reference architecture and the Delegate MAS architectural pattern

This section explains the mapping between the PROSA reference architecture and the Delegate MAS architectural pattern.

The PROSA reference architecture has three basic holon types: the Resource holon, the Order holon and the Product holon. Each holon has a corresponding agent. Each of the three basic PROSA agents is a candidate to apply the Delegate MAS pattern. Note that the pattern can be applied more than once on the same agent type.

The product line has three types of delegate multi-agent systems: feasibility delegate MAS, exploring delegate MAS and the intention delegate MAS. The feasibility delegate MAS provides the agent feasibility information, i.e. a path through the environment, a sequence of resources which can be reached through this path. The exploring delegate MAS provides the agent options to achieve its goal. An option is a sequence of operations which make use of the resources of the environment. The intention delegate MAS allows the agent to communicate its intention to the other agents by making use of the environment. Several manufacturing control applications used this set of services (39; 8; 31).
The section specified the set of possible delegate MAS types. It now specifies which base PROSA agent type makes use of which delegate MAS services.

Figure 5 shows one possible mapping between the Delegate MAS services and the PROSA base agent types that is applied in several manufacturing applications (27; 22; 13; 21). The mapping is most suitable for an order driven production system with a flow-shop layout. A flow-shop layout is a combination of different production lines, with switches in between. Therefore early choices in routing may hinder product completion later on, since certain operation may not be available on all production lines. The configuration is suitable for order-driven production since only order agents send out intention ants. Therefore, they drive the decision making (although resource and product agents are involved in the decision process, they do not initiate this process).

When the mapping in figure 5 is used, the cooperation between the Delegate multi-agent systems and the basic PROSA agents has a strong resemblance with how a tourist books a hotel room. The feasibility delegate MAS takes care of the marketing. The hotel (resource) spreads word of its existence on other resources so that other ants can find the information. The exploration delegate MAS originates from the Tourist Order Agent. The exploring ants search through these 'advertisements' and report the results back to the Tourist Order Agent. Out of these, the order agent makes a choice. The difference with people, is that the agent does not make this choice once, but over and over again. The intention delegate MAS propagates the chosen hotel rooms through the environment. Hotel Resource Agents have a ‘reservation department’ (e.g. a blackboard) which keeps track of the reservations of the order agents. There is no explicit cancellation strategy. Reservations disappear after a short time if no confirmations are received anymore. For more detailed information, see (14; 27).

The following list shows the mappings that have already been implemented in various applications.

- Flowshop configuration (e.g. (27; 22; 13; 21)).
  - Order agent: explore and intention delegate MAS
  - Resource agent: feasibility delegate MAS

- Jobshop configuration (e.g. (39)).
  - Order agent: explore and intention delegate MAS
– Resource agent: feasibility delegate MAS

– Jobshop configuration with flow control (e.g. (31)).

– Order agent: explore and intention delegate MAS

– Resource agent: intention delegate MAS

3.3 Agent architecture

The agent architecture considers ‘autonomy’ and ‘being situated in an environment’ the core characteristics of an agent. Furthermore, the agent software module is responsible for providing the integration and the self-assertion functionality of a holon, as explained in section 2.2.2. To provide these functionalities, agents delegate responsibilities to their respective delegate multi-agent systems (section 2.4).

More concrete, the agent architecture considers the following responsibilities of an agent:

• Performing actions in the environment. An example of an action is drilling a hole in a piece of metal at a particular time.

• Acting on changes in the environment. Events in the environment can influence the decisions of a holon. Breakdowns, execution delays, order arrivals are examples of events happening in the environment.

• Instantiating and maintaining the delegate MAS services.

• Communicating with other agents.

This section shows how the internal architecture of the agent realizes these functionalities. The section explains the different layers and modules of the agent architecture and discusses their relations. To illustrate the behaviour, the section describes an example scenario. The scenario illustrates how an agent works together with a delegate MAS.

3.3.1 Software architecture

Figure 6 shows a module layer view of the agent architecture. The view shows two layers, the consciousness layer and the activity layer. The activity layer contains the interaction and delegation module.

The consciousness layer encapsulates the autonomy characteristic of the agent. Autonomy refers to the capability of the agent to take initiative. An initiative triggers an activity that is performed by the agent at a certain point in time. Example activities are perceiving the environment or placing a pheromone. The consciousness layer is responsible of assigning these activities to time.

These assignments to time can be either slots or points in time.
The activity layer is responsible of performing the activities that are activated by the consciousness layer. These activities may require new initiatives from the agent. The activity layer can request these activities to the consciousness layer.

For example, sending out an intention ant requires an action to be carried out in the environment, i.e. creating an intention ant. The activity layer requests this activity to the consciousness layer, which schedules the creation of the intention ant. When the time is reached at which the creation activity is scheduled, the consciousness layer triggers an activity in the activity layer. This layer then executes the activity.

The activity layer has two modules: the interaction module and the delegation module.
The interaction module is responsible for enabling interaction both with agents and the environment. The module is composed out of three modules (as shown in figure 7).

The communication module encapsulates the concrete direct communication mechanism. Agents have connections to each other, which reflect the connections between the entities in the environment. E.g. when a resource entity in the environment has a connection to an order entity (indicating the order is located at the resource), the corresponding order agent has a connection to the corresponding resource agent. Agents connected to each other can communicate by exchanging messages through the environment.

The action module is responsible for executing actions modifying the environment. The outcome of an action is not guaranteed, it is merely an attempt to execute the action. Example actions are creating an ant agent, depositing pheromones, etc.

The perception module is responsible for perceiving the environment. An agent cannot perceive the whole environment. Its perception is limited to the resource where it is located and to the orders and products located on that resource.

Figure 8 Delegation: module view

Delegation module The delegation module is responsible for managing the delegation of responsibilities to swarms of other agents, i.e. delegate multi-agent systems. The architecture considers three different types of delegation: belief management delegation, plan management delegation and intention management delegation.

Managing the delegation involves two responsibilities: maintaining the swarm of agents and optionally interpreting the results they bring back to the agent.

Maintaining the swarm of agents involves the creation of ant agents. The module is responsible for deciding at which rate the ant agents are generated and which types of ant agents are created (there can be different types of ant agents for one delegate MAS; for example exploring ants guided by the planning and exploring
ants that are not guided (see (34)). There are three types of delegate MAS: the feasibility delegate MAS, the exploration delegate MAS and the intention delegate MAS.

Interpreting the results involves fetching the pheromones from the blackboard of the resource where the agent resides and interpreting these pheromones. The interpretation of these pheromones is different for each type of delegation. For the belief management, delegation interpretation does not happen by the PROSA agent itself (i.e. the order, product or resource agent) but by the ant agents of the exploration delegate MAS. For the plan management delegate MAS, this interpretation involves maintaining a set of interesting options (e.g. the five best performing plans) in the agent. For the intention delegate MAS this interpretation involves selecting amongst the most interesting options and determining whether to keep the current intention or to switch to another option.

3.3.2 Example

This example illustrates how an agent and a Delegate MAS cooperate. The example describes how the order agent uses an exploring delegate MAS and intention delegate MAS during its life.

For simplicity reasons, the example considers a job shop layout. In a job shop layout, all resources are accessible from all other resources. Therefore, the presence of feasibility pheromones indicating the presence of other resources on the blackboards, are unnecessary in this case. Therefore the order agent makes use of a combination of an exploring and intention delegate MAS.

**Creation of the order** As soon as the production order is created (e.g. by the Material Requirements Planning system), a corresponding order holon is created and added to the manufacturing control system. The order holon has an environment entity reflecting the state of the order. The state of the order is initialized at the entrance of the factory, which in turn is connected to a transporting unit. The order holon also has an order agent that has access to the order environment entity.

The creation of the order agent triggers the consciousness module to periodically activate the plan management and intention management modules in parallel. Both modules perform two basic tasks: maintaining the swarm and interpreting the results (as explained in section 3.3.1). The intention management module fetches the pheromones provided by the plan management module. It uses a submodule comparing the various explored paths with the current intention of the order holon. If the explored path is better than the intention, the order holon in this example takes this path as its intention with a chance of 60% if the order holon already had an intention and 90 % otherwise (see (31) for more details).

The exploring and intention ants travel through the environment. On their journey, they request information of the resource environment entities. The resource environment entities store these requests. The resource agent observes and schedules them. The ants fetch the results from this scheduling in the environment entity and travel to the next resource. The bookings from the intention ants create a local schedule on the environment entity that is used to steer production.

**Release of the order into production** The resource environment entities monitor the local schedules that are deposited by the resource agents. Once the start time of
the next operation equals the current time, the resource environment entity sends a command to its respective physical entity (the actual machine). They also receive sensory information on the progress of the operation, information on eventual breakdowns and the finishing of the operation.

The order agents observe the operation start and stop by monitoring their respective environment entity. They keep sending exploring and intention ants during execution. The local schedules are continuously refreshed. If something goes wrong, the ant agents discover this and make this information available. The order agent then decides if he changes his intention or not. If he switches, the old intention is discarded and evaporates automatically.

Removal of the order out of the system Once the order has completed the production process, the order agent is discarded (archived). The order environment entity is now located at the exit of the factory, where it can be fetched for transportation.

3.4 Environment entity

A holon module contains two modules: the agent module and the environment entity module (section 3.1). This section discusses the environment entity module in more detail.

The environment entity module is responsible for the software representation of the physical entity in the external world of interest. Environment entities also serve as a place to deposit pheromones (section 2.4). The pheromones need to be observable by other agents. The environment entity can modify the content of the pheromones (e.g., merging two pheromones). The environment entity is also responsible for the evaporation of the pheromones. To support this functionality, current applications apply the blackboard pattern.

This section first discusses the blackboard pattern. Next, the section gives an architectural description of the blackboard pattern together with a list of characteristics motivating the choice for this pattern. Finally, it documents the architecture of an environment entity.
3.4.1 Blackboard pattern

Figure 10 shows a module view of the blackboard system (6). It describes three modules:

- Knowledge sources (KSs) are independent modules that contain the knowledge needed to solve the problem. KSs can be widely diverse in representation and inference techniques.

- The blackboard is a module containing input data, partial solutions, and other data that are in various problem-solving states.

- A control module makes runtime decisions about the course of problem solving and the expenditure of problem-solving resources. The control module is separated from the individual KSs. Several implementations are possible. Triggering knowledge sources when new information is available is one alternative. Another approach uses polling at regular frequencies in order to sense new information.

The blackboard system has some important characteristics with respect to manufacturing control applications (6):

- Independence of expertise and diversity in problem-solving techniques. Each holon is able to contribute independently using his own area of expertise and decision making techniques.

- Flexible representation of blackboard information. There are no prior restrictions on the interaction language. Not all agents need to understand all information on the blackboards. Information is used when understandable.

- Positioning metrics. One position metric is to use multiple blackboards. Each holon has a blackboard attached. This makes information locally available.

- Ordering metrics. The pheromones evaporate after some time. The architecture uses a discrete ordering, i.e. the pheromone is available or is not available.
3.4.2 Environment entity architecture

Layers and modules First this section documents the different layers and modules of the environment entity architecture. Figure 11 shows a module layer view of the environment entity.

The module view of the environment entity has two layers: the reflection layer and the blackboard layer.

The blackboard layer corresponds to the blackboard module of the blackboard pattern (see section 3.4.1). Partial solutions correspond to the pheromones placed by the ants. E.g. an intention pheromone contains information on the plan an order is committed to execute.

State reflection and what-if functionality are the responsibilities of the reflection layer. These two responsibilities are assigned to two modules, the state module and the what-if module.

The state module reflects the state of the physical entity, i.e. it is designed to contain all properties of the physical entity. E.g. a conveyor belt environment entity has a speed property, a property holding the current entities being transported, the positions of these entities, etc. Synchronization with the physical entity is the core responsibility of this module. Synchronization is both updating the state according to changes from the physical entity as well as controlling the physical entity (e.g. executing an operation on a machine).

The what-if module extrapolates the state from one time to a given time depending on the known information. There are two typical uses of this module, current state determination and short-term forecast creation.

Up-to-date sensor information of the current state is not always available. Sensors registering status updates are not everywhere available and in some cases sensors operate with a slow frequency. By using (historical) knowledge on the behaviour of the entity the module estimates the current state using the last measured state. This shields the agents from discontinuous sensor updates. Short term fore-
casts answer the questions from ant agents. A short example clarifies this. An exploring ant tries to find a particular plan in the factory. In order to build such a plan the ant queries the resources (laying on the path of the ant) about the possible starting and finishing times for the respective operation. To respond to the ants, resources need to know their state at the time the order would arrive on the resource. Knowing this state, resources know whether setups need to be done before starting the operation. Finally, after virtually executing the operation the resource can give the expected finishing time to the ant.

Information about the expected states of the environment entity in the near future can be stored as pheromones in the blackboard module of the environment entity. This information combined with current state information can be requested by agents or other entities.

Relations between modules In the blackboard pattern, the blackboard depends on the knowledge sources and on the control module.

The knowledge sources correspond to the agents in the system. Agents hold the knowledge to fulfill the goal of the control system. The consciousness layer of an agent takes the initiative for the decision taking part in a holon. In the decision taking process, agents put pheromones on the blackboards via their corresponding delegate MAS. In particular, ant agents deposit the pheromones.

The control module is embedded in the sensing module of the agent (3.3.1).

The representation layer depends on the blackboard layer. The entity extrapolates the current state depending on the available information (what-if). Intentions (section 3.2) give an indication of which actions are going to happen into the future. The intentions are available in the blackboard layer.

4 Related work

This section compares the presented agent architecture with the Belief-Desire-Intention (BDI) agent architecture (20). The section first discusses the main functionality of the BDI architecture. Next, the section discusses how this functionality is obtained by using the agent architecture presented here. Finally, the section illustrates the differences between both architectures by means of an example.

4.1 Main functionality in the BDI architecture

The BDI architecture has a main module, a BDI interpreter, which makes use of other modules to deliberate which reaction the agent is intending to undertake given the events it receives in its event queue. The agent then executes these intentions when the execution preconditions are fulfilled. To perform this reaction deliberation, the reaction deliberation module makes use of three other modules, the belief, plan and intention management module. These modules are now discussed.

Management of beliefs The agent needs to have information about its environment. The architecture does not assume that the agent can obtain this information in one sensing action. Therefore this state information is stored internally in the agent
and is updated appropriately after each sensing action. This information is called the beliefs of the agent.

**Plan management** Plans are abstract specifications of both the means for achieving certain desires and the options available to the agent. An invocation condition specifies when a plan can be chosen. The agent stores all plans in a data structure and checks if certain plans become available each time beliefs are updated.

**Intention management** The agent keeps track of the intentions it adopts in a runtime stack of hierarchically related plans. Intentions can run in parallel or can be suspended until a condition occurs.

4.2 Comparing both architectures

The main functionality of both the BDI agent and the agents presented in this paper is to create intentions that take the events that occur in the environment into account. To determine their intentions, both agent types use three types of information: information about facts (beliefs), information about strategies to achieve a desire (plans) and their current intentions. This main functionality is addressed by the interpreter module in the BDI architecture and by the consciousness module in the agent architecture presented in this paper.

The main difference between both agent architectures is the strategy the agents uses to manage the beliefs, desires and intentions. A (classical) BDI agent explicitly manages this information in an internal data structure. The agents in the product line architecture store this information in the environment by using delegate MASs and shares this information with other agents. The belief, plan and intention management modules of the BDI agent correspond to the belief, desire and intention delegation modules in the agent architecture presented here.

4.3 Example

A paper of Rao and Georgeff (20) presents an air traffic management application. This section briefly describes how this application can be realised using the architecture described in this paper.

In the classical BDI approach there is one local agent - the aircraft agent - and four global agents: the sequencer agent, the wind modeller, the coordinator and the trajectory checker.

In the approach presented here, there are three agent types: the resource agent, the product agent and the order agents. The aircrafts are order agents. The airport itself is a resource agent, which is connected to other airport agents. All resource agents are situated in the environment. Note that this is one possible design where the environment has a low level of detail. A more detailed version would also model the landing strokes and assign a resource agent to each landing stroke. The airport holon then aggregates the landing stroke holons.

The wind is not an agent but a physical law present in the environment. A global coordinator would be a staff agent (this is optional), which gives advice to the order agents. The trajectory checker functionality is obtained by the resource
agents feasibility delegate multi-agent systems which spread information about the presence of the resources in the environment (comparable to road signs). The sequencer functionality emerges out of the sequencing functionality of the individual resource agents. A global sequencer agent is not required (although it is possible to introduce an optional staff sequencer holon).

The aircraft agent delegates exploration to the exploration delegate MAS. The ant agents in the MAS return possible plans to the aircraft agent. When choosing an intention, the aircraft agent uses a second delegate MAS, the intention delegate MAS, to spread this information in the environment.

The airport agent uses a local scheduling strategy to determine the sequence of aircrafts. It gives answers to the exploring and intention ant agents when they explore a slot and book a slot respectively.

5 Conclusion

This paper gives a systematic architectural description of a product line for manufacturing control applications. This product line represents a set of manufacturing control systems that share a common, managed set of features that satisfy the specific needs of this manufacturing control applications. The products are developed from a common set of core assets: a shared analysis of requirements, a reference architecture (PROSA), an architectural style (holonic architectures) and an architectural pattern (delegate MAS). These assets have proven their value over several manufacturing control applications. There are strong indications that these assets are useful in a broader domain (e.g. traffic control (15), railway applications (8)).

Five important qualities are considered by the architecture: performant control, reducing the impact of changes, maintaining good connectivity and improving the responsiveness and proactiveness of the system. The general functionalities the control system needs to achieve are the management of operations, tracing the execution and supervising the execution.

A first core asset, the PROSA reference architecture transfers experience from the manufacturing domain into the architecture. It places three separate and essential concerns (i.e. logistics, technological aspects and resource allocation) in manufacturing at the same level. This is in contrast with traditional hierarchical control systems, and allows to:

- separate the resource allocation aspects from process specific aspects
- separate the necessary generic modules from the specific modules
- separate the structural aspects of the architecture from the algorithmic aspects

These characteristics reduce the impact of changes in decision making.

A second core asset, the delegate MAS architectural pattern, reduces the complexity of the agent by delegating the functionality of the base agents to a swarm of lightweight ant agents. These ant agents receive information from the PROSA agents and make it available in the environment. This pattern improves the responsiveness, reactiveness and the proactiveness of the manufacturing control system.
The paper specifies how the core assets are combined to create a manufacturing control system architecture. This systematic description supports software architects in their architectural design for new applications in the manufacturing control application domain.

References


