

Chapter 2

A Reference Architecture Capturing Structure and Behaviour of Warehouse Control

Jacques Verriet and Bruno van Wijngaarden

Abstract Warehouse management and control systems are responsible for the operations in a warehouse. These systems are usually very complex due to the delivery requirements of the warehouse customers. These delivery requirements are often very specific for a group of customers, which makes it hard to reuse warehouse management and control functionality for other warehouses. In this chapter, we will introduce a method that will greatly increase the reusability of warehouse management and control functionality. We present a reference architecture that supports the development of warehouse management and control systems. This modular reference architecture is based on functional components, which can be configured using structural and behavioural parameters. This configuration is sufficient to build a warehouse management and control system. This chapter introduces the reference architecture and demonstrates how it can be used to decrease the warehouse management and control system development effort.

2.1 Introduction

The operations in a warehouse are controlled by a warehouse management and control system (WMCS). To achieve a high warehouse performance, a WMCS needs to use a warehouse's scarce resources in an efficient manner. A WMCS is commonly seen as a layered system. For instance, Ten Hompel and Schmidt [8] distinguish several layers of WMCS control. These include strategic control by an enterprise resource planning (ERP) system, functional control by a warehouse management

J. Verriet (✉)

Embedded System Institute, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
e-mail: jacques.verriet@esi.nl

B. van Wijngaarden

Vanderlande Industries B.V., Vanderlandelaan 2, 5466 RB, Veghel, The Netherlands
e-mail: bruno.van.wijngaarden@vanderlande.com

Fig. 2.1 Layers of warehouse functionality

Enterprise Resource Planning (ERP)
Planning (stock, orders & resources)
Scheduling (task lead time & resource workload balancing)
Material Flow Control (MFC)
Material Handling System (MHS)

system (WMS), section control by a warehouse control system (WCS), and process control by a material flow controller (MFC).

In this chapter, we will distinguish five layers of WMCS functionality, which are illustrated by Fig. 2.1. The top layer is the ERP system, which is responsible for the high-level management of orders and stock. The second layer, the planning layer, handles the assignment of orders to stock and equipment resources. The planning layer is also responsible for keeping sufficient stock levels by replenishing low stock levels and relocating superfluous stock. The scheduling layer is responsible for balancing the system to obtain the optimal system performance. The scheduling layer’s responsibilities include the prioritisation of the tasks that have been assigned to resources. Scheduling selects tasks and forwards them to the MFC layer, which controls the warehouse equipment and provides equipment status information to the scheduling layer. The MFC layer provides an interface to the bottom layer, the material handling system (MHS), which contains the warehouse equipment handling the actual execution of warehouse tasks.

Because of the size and complexity of modern warehouses, the development of a WMCS is very difficult and time consuming. As identified in Chap. 1, an important element of the complexity of a warehouse involves the product and delivery requirements of the warehouse customers. Another part of the complexity of WMCS development is due to the lack of WMCS development support. The layered architecture shown in Fig. 2.1 provides some design guidelines, but it does not provide sufficient support for making the design decisions needed to develop an effective WMCS. This lack of support increases the risk of developing a customer-specific WMCS, whose functionality cannot be reused for other warehouses.

In this chapter, we will address the lack of WMCS development support. We hypothesise that a WMCS reference architecture can reduce the WMCS development effort and allows WMCS functionality to be reused for different warehouses. We will test this hypothesis by defining a WMCS reference architecture that captures both the structure and the behaviour of warehouse control. We will illustrate how our WMCS reference architecture decreases the WMCS development effort and increases the reusability of WMCS functionality.

2.1.1 Decentralised Warehouse Control

Since modularity and loose coupling are important characteristics of systems built from reusable components, we let ourselves be inspired by the recent research into agent-based control systems. Such systems consist of a collection of autonomous agents, each having a limited scope. Examples from the warehousing domain include the work by Kim et al. [4] and Graves et al. [3]. They study an agent-based WMCS for warehouses with a man-to-goods and a goods-to-man picking concept, respectively. They propose a hierarchical agent-based WMCS, in which high-level agents compute a global schedule. The low-level agents constantly reschedule their work to match the real-time conditions. If they find a better schedule, they negotiate with the high-level agents for a schedule change. The high-level agents will allow the proposed schedule change if the new schedule does not deteriorate the global schedule. The results of Kim et al. [4] and Graves et al. [3] show that the constant rescheduling by the low-level agents has a positive influence on the system performance. Although the WMCS agents do not have full system knowledge, they have a similar performance as the original WMCS under normal circumstances and a better performance in case of exceptions.

Decentralised control has also been applied in domains with similarities to the warehousing domain. Other examples of decentralised control mainly involve the manufacturing domain. There are many publications describing the benefits of decentralised manufacturing systems. An example is the engine manufacturing line described by Fleetwood et al. [2], who present a decentralised control system that has a better performance and flexibility than its centralised counterpart.

Many applications of decentralised control systems are very specific for the application. However, a few reference architectures have been proposed for decentralised control systems. Examples are PROSA [9] and ADACOR [5]. The applicability of these reference architectures has been demonstrated using a variety of applications in the manufacturing domain. These reference architectures improve reusability by identifying generic roles and corresponding interaction protocols. Within the Falcon project, Moneva et al. [7] defined a WMCS reference architecture, which standardises WMCS functionality using roles and interaction protocols similar to PROSA's. These generic roles and interaction protocols allow warehouse operations to be performed by a collection of agents organised in a hierarchy matching the underlying material handling system.

2.1.2 Outline

In this chapter, we will go one step further in standardising WMCS functionality than Moneva et al. [7]. We will present a WMCS reference architecture that standardises not just the components and their interfaces, but also the components' functionalities. This is achieved by using generic behaviours with local business rule plug-ins, which can be used to make a behaviour application specific. This standardisation allows a

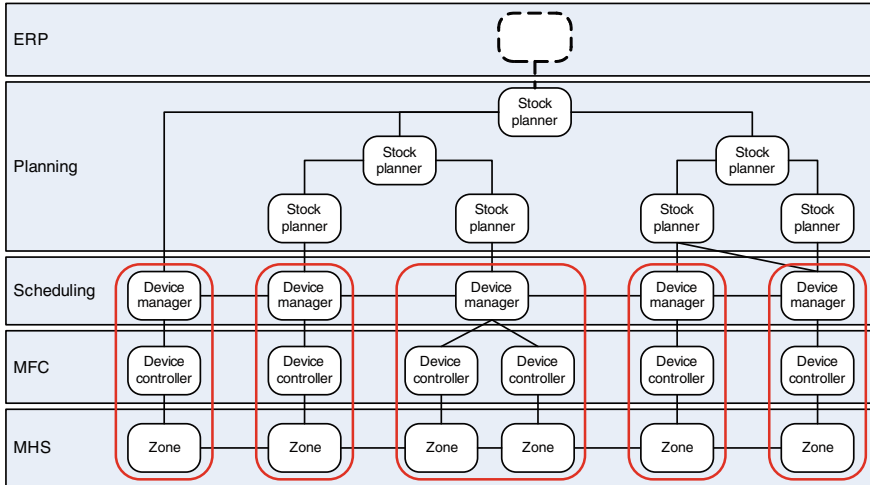


Fig. 2.2 WMCS reference architecture

WMCS to be generated automatically by identifying the components, their interconnections and behaviours, and by specifying the business rules of these behaviours.

The chapter is organised as follows. [Section 2.2](#) describes the structural and behavioural components of the WMCS reference architecture. [Section 2.3](#) introduces an automated case picking system, which will be used to illustrate the benefits of our reference architecture. A prototype implementation of the reference architecture is described in [Sect. 2.4](#). [Sections 2.5](#) and [2.6](#) present the initial results validating the new architecture. [Section 2.7](#) describes the current status of the reference architecture and an outlook into the future.

2.2 Warehouse Management and Control Reference Architecture

In this section, the WMCS reference architecture is explained in terms of its structural and behavioural components.

2.2.1 WMCS Components

Our decentralised WMCS reference architecture is a layered system, which distinguishes the same layers as [Fig. 2.1](#). The layers of the architecture and their components are shown in [Fig. 2.2](#). The WMCS reference architecture distinguishes four types of components each corresponding to a different layer of the WMCS hierarchy: stock planners, device managers, device controllers, and (material handling) zones.

The *stock planners* reside in the planning layer. The stock planners' main responsibility is the delivery of goods to its (local) customers. This delivery service is determined by its connected device managers, which define which types of goods can be delivered from a stock planner's local stock. Figure 2.2 shows that the stock planners form a tree. This tree structure allows a logical clustering of stock planners, for instance based on the type of goods they can deliver.

The *device managers* in the scheduling layer couple the abstract stock planners in the planning layer to the concrete device controllers in the MFC layer. They are responsible for sequencing the tasks assigned by the stock planners. Upon the assignment or completion of a task, a device manager selects the next task to be executed and forwards it to its underlying device controller. Figure 2.2 shows that the device managers form a network of device managers. This network is an abstraction of the warehouse topology in the MHS layer. This abstraction is used to determine how work is to be handed over in the scheduling layer in order to obtain the desired flows of goods through the warehouse. The connections in this network can be seen as producer-consumer relationships between device managers.

The *device controllers* in the MFC layer are responsible for the coordination of task execution by the material handling zones. Figure 2.2 shows that there is a one-to-one correspondence between zones and device controllers. It also shows that each device controller is connected to one device manager, but that a device manager can be connected to several device controllers. Then the device manager is responsible for dividing the work over the underlying device controllers.

The *zones* in the MHS layer are responsible for the actual execution of warehouse tasks. Examples of material handling zones are miniloads, order-picking workstations, and transport loops. Figure 2.2 shows connections between the zones; these correspond to the physical connections between the warehouse equipment.

Figure 2.2 shows a similarity to the organisation-based WMCSs in Chap. 3: a device manager and its underlying device controllers and zones can be seen as the presences of a single MASQ agent in the scheduling, MFC, and MHS communication spaces. This has been illustrated by the rounded rectangles in Fig. 2.2.

Both the zones and the device controllers are equipment-specific, and therefore reusable, components. In our WMCS reference architecture, the stock planner and device managers are generic components. Both types of components have parameters from which a structure as shown in Fig. 2.2 can be constructed. The stock planners' parameters include their parent and children in the tree of stock planners and the connected device managers. The device managers' parameters include the neighbours in the network of device managers and the capabilities of the underlying device controllers.

2.2.2 WMCS Behaviours

In Sect. 2.1, we have defined the components of our WMCS reference architecture. A collection of these components needs to behave in such a way that the underlying

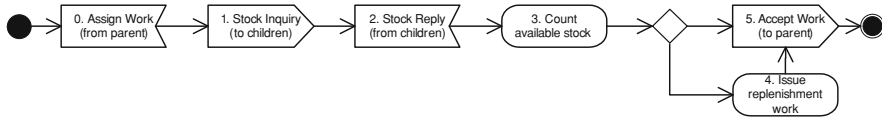


Fig. 2.3 AssignWork behaviour activity diagram

warehouse equipment operates in an efficient manner. The main challenge in designing an effective WMCS is designing the behaviours of its components that together provide the desired system behaviour. We will focus on the behaviour of the stock planners and device managers, as their high-level behaviour has the largest influence on system behaviour.

To alleviate the system design effort, we have standardised component behaviour. We have defined a collection of generic behaviours for stock planners and device managers: their interaction has been fixed with respect to the interacting agents and the interface object types. Each of these behaviours is triggered by the reception of messages matching generic interfaces. In this section, we will describe some examples of our reference architecture's behaviours implementing the generic interaction protocols; this includes the object types being communicated.

2.2.2.1 Delivery Behaviour

The main goal of the stock planners is the delivery of goods to its customer stock planners. Several generic behaviours have been defined to control this delivery. An example is the AssignWork behaviour shown as a UML activity diagram in Fig. 2.3. This behaviour is triggered by the reception of an AssignWork message; this is a standardised interface specifying the goods to be delivered and the container type in which to deliver them. After receiving the message, the AssignWork behaviour sends a StockInquiry message to its child stock planners to ask for the availability of the requested goods. It then waits for the corresponding StockReply messages and computes the available stock level. If the stock level does not suffice, the behaviour issues replenishment work. This involves sending and receiving several messages, but this has not been detailed in Fig. 2.3. The AssignWork behaviour ends by sending an AcceptWork message to acknowledge the original AssignWork message.

The completion of the AssignWork behaviour will asynchronously trigger another delivery behaviour, the ForwardWork behaviour, which is shown in Fig. 2.4. This behaviour starts by checking whether all stock needed for a work assignment is available in its children's local stock. If sufficient stock is available, it will send a SupplyCostInquiry to its child stock planners. The children will answer with a SupplyCostReply specifying the cost of delivering parts of the work to be forwarded. Based on the answers, the ForwardWork behaviour assigns work to its children using an AssignWork message. The ForwardWork behaviour will continue until it has completely forwarded the work assignment to its child stock planners.

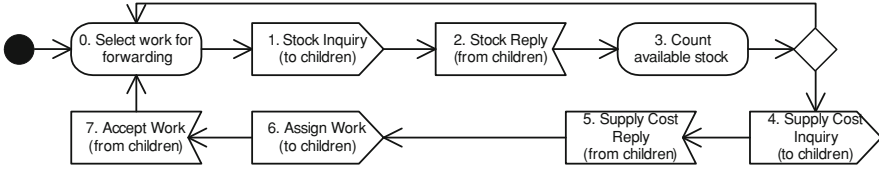


Fig. 2.4 ForwardWork behaviour activity diagram

Note that the ForwardWork behaviour and the AssignWork behaviour trigger each other: the ForwardWork behaviour of a stock planner is triggered by the AssignWork behaviour of its parent and triggers the AssignWork behaviour of its children. The combination of these behaviours allows work to be forwarded from the root of the stock planner hierarchy to the leaf stock planners. From there, it is forwarded to the associated device managers. The latter involves a ForwardWork behaviour that is similar to the one shown in Fig. 2.4.

2.2.2.2 Replenishment Behaviour

The description of the AssignWork behaviour already showed the existence of replenishment work triggered by the forwarding of work through a hierarchy of stock planners. These replenishment assignments are handled by creating delivery work that is forwarded using the ForwardWork and AssignWork behaviours described in Sect. 2.2. Order-driven replenishment may take a long time to be fulfilled, because the required goods need to be transported from one part of the warehouse to another. This may not be acceptable for popular products; these fast movers have to be delivered so often, that they should be in stock all the time. This is achieved by the MinMaxReplenishment behaviour. This behaviour gets triggered when goods have been delivered by a stock planner. The behaviour will check whether the stock level for a certain product has dropped below a minimum. If so, the behaviour will issue replenishment work, which will be handled in the same manner as the order-driven replenishment issued by the AssignWork behaviour. This will ensure that the stock level will be replenished to a specified maximum level.

2.2.2.3 Execution Behaviour

The delivery and replenishment behaviours involve standardised behaviours of the stock planners. The behaviours of the device managers have been standardised as well. An example of a device manager behaviour is the Execution behaviour, which is triggered by an event from a device controller signalling the completion of a task. This behaviour will select the next task to be executed. This involves sending a TaskInquiry message to the consumer device managers; this list contains all tasks that can be executed. The consumers will select a task from this list and send it as

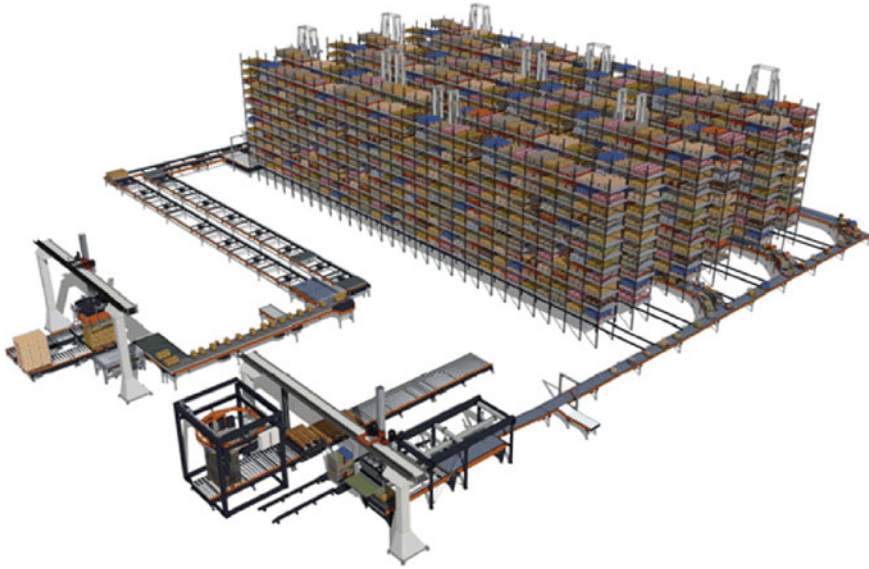


Fig. 2.5 Automated case picking module

a reply. The producer device manager will select one task from the collection of replies and assign it to the signalling device controller, which is responsible for the execution by the material handling system.

2.3 Case Study

We will use the system shown in Fig. 2.5 to illustrate the main ingredients of our reference architecture. It shows an automated case picking module with a palletiser and three case picking cells. Each cell consists of two pick fronts, a reserve, two case pickers, and a tray miniload.

The corresponding component structure is shown in Fig. 2.6. The module, pick fronts, and reserves are stock planners that are part of planning layer; the palletiser, case pickers and tray miniloader are device managers, which reside in the scheduling layer. Besides these, there is an initiation component that plays the role of the ERP system. We do not consider the MFC and MHS layers explicitly.

The module's responsibility is delivering ordered pallets. If it is asked to deliver a pallet, it asks its children, the pick fronts and the reserves, for the corresponding cases. The pick fronts' local stock consists of trays, from which the associated case picker can pick cases. The reserves also store trays of cases, but they cannot be used to deliver cases. If a pick front cannot deliver a certain product, it can be replenished from a reserve: the corresponding tray miniload can pick entire trays and place them in the pick front's local stock.

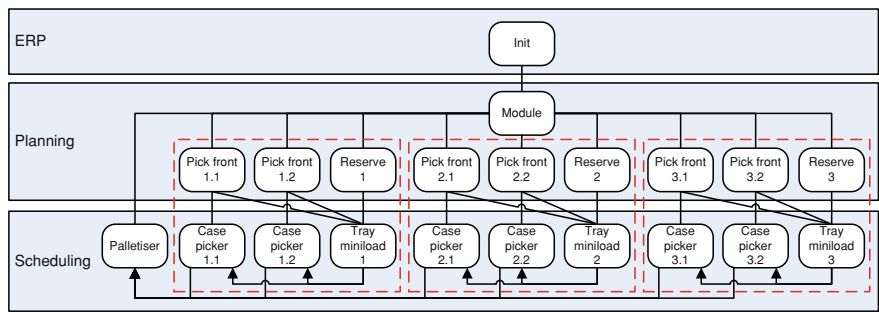


Fig. 2.6 Automated case picking WMCS components

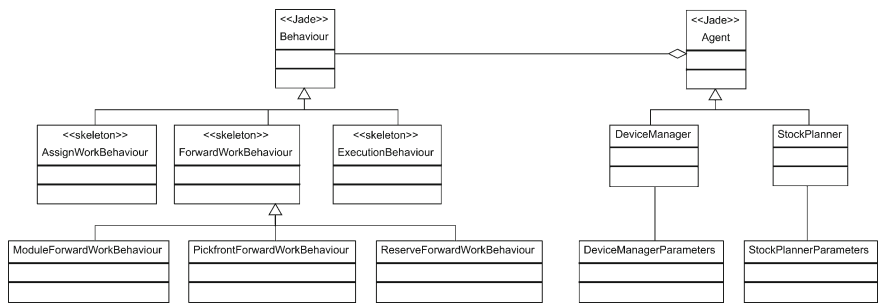


Fig. 2.7 WMCS reference architecture class diagram

Figure 2.6 shows that there need not be a one-to-one relationship between stock planners and device managers. The tray miniloads are associated with two pick fronts and one reserve. It also shows that the device managers are connected in a network, which describes their producer-consumer relationships. Since a tray miniload can replenish two case pickers, it is connected to two case pickers. Similarly, the case pickers are all connected to the palletiser, as they deliver cases for pallet building by the palletiser.

2.4 Implementation

To assess the validity of the WMCS reference architecture introduced in Sect. 2.2, we have implemented a prototype in Java. This prototype implementation only covers the planning layer and the scheduling layer, since these layers cover the most important component behaviours. The actual execution by the MFC and MHS layers is replaced by a simulation, which is included in the scheduling layer. A partial class diagram for the implementation is shown in Fig. 2.7.

The implemented prototype is an agent-based system built upon Jade middleware [1]. A Jade application consists of a number of agents, each having a collection of behaviours. Figure 2.7 shows Jade’s Agent class and its Behaviour

class, which both have warehouse-specific specialisations in our reference architecture. We distinguish two types of parametrised agents: stock planner agents and device manager agents. These correspond to the components described in [Sect. 2.1](#). The parameters of the stock planners specify the parent and child stock planners, the connected device managers, the initial stock level, and a collection of behaviours. The main parameters of the device managers are its behaviours and the consumer device managers in the network of device managers.

The agents' behaviours are the main ingredients for obtaining the desired system behaviour. As described in [Sect. 2.2](#), the behaviours of the stock planners and device managers have been standardised. This means the interaction between the agents has been fixed, both with respect to the interacting agents and the interface object types. Examples of these generic interaction protocols are described in [Sect. 2.2](#). However, the detailed content of the exchanged messages has not been specified. We will refer to these interaction protocols as *skeleton behaviours*.

Figure 2.8 shows an excerpt of the implementation to illustrate the concept of skeleton behaviours. It shows parts of the code of three classes: Main, ForwardWorkBehaviour, and ModuleForwardWorkBehaviour. The class Main defines the WMCS by specifying the agents and their behaviours. Figure 2.4 shows a behaviour ModuleForwardWorkBehaviour being added to the module agent. This behaviour is a specialisation of the abstract class ForwardWorkBehaviour, which is a skeleton behaviour implementing the activity diagram shown in Fig. 2.4 as a state machine. State 0 of the activity diagram corresponds to state 0 in the code in Fig. 2.8. This state calls an abstract method `selectNextWorkAssignment`, which corresponds to an empty placeholder function for a business rule in the abstract ForwardWorkBehaviour class. This method is defined in the concrete ModuleForwardWorkBehaviour class.

The notion of skeleton behaviour is further illustrated in Fig. 2.7. It shows the abstract ForwardWorkBehaviour class and three of its application-specific specialisations for the module, the pick fronts, and the reserves. These three behaviours share the interaction protocol defined by the skeleton behaviour ForwardWorkBehaviour, but implement the corresponding business rules differently.

2.5 Experimental Validation

The architecture described in the previous sections is being validated using a series of experiments of the implemented prototype described in [Sect. 2.4](#). The first experiments of the architecture involve the automated case picking module described in [Sect. 2.3](#). The first experiment has been described by Verriet et al. [10]; this experiment focused on stock planner delivery behaviour and device manager execution behaviour. It did not involve replenishment, because the pick fronts were assumed to have sufficient stock to fulfil all orders.

The second experiment focuses on delivery and replenishment behaviour. The experiment starts with pick fronts without any stock and reserves having sufficient stock for all orders. The components of the corresponding WMCS and their behaviours are listed in Table 2.1. There are ten stock planner agents: one module, six pick

```

public class Main {
    ...
    public static void main(String[] args) {
        ...
        moduleAgent.addBehaviour((Behaviour) Class.forName(
            ModuleForwardWorkBehaviour.class.getName()).newInstance());
        ...
    }
    ...
}

public abstract class ForwardWorkBehaviour extends Behaviour {
    ...
    public final void action() {
        ...
        switch (state) {
            case 0:
                retrieveWork();
                break;
            case 1:
                ...
        }
    }
    ...
    private void retrieveWork() {
        ...
        activeWorkAssignment = selectNextWorkAssignment(deliveryWip);
        ...
    }
    ...
    protected abstract WorkAssignment selectNextWorkAssignment(
        Collection <WorkAssignment> workAssignments);
    ...
}

public final class ModuleForwardWorkBehaviour extends
    ForwardWorkBehaviour {
    ...
    protected WorkAssignment selectNextWorkAssignment(
        Collection <WorkAssignment> workAssignments) {
        ...
        return workAssignment;
    }
    ...
}

```

Fig. 2.8 WMCS source code excerpts

fronts, and three reserves. Associated with these stock planners are ten device manager agents: six case pickers, three tray miniloads, and one palletiser. These agents and their connections correspond to those in Fig. 2.6. It shows that each pick front has two device managers: a case picker and a tray miniloader. The former allows the delivery of cases; the latter is needed for replenishment. Besides the stock planner and device managers, there is an initiator agent playing the role of the ERP system.

The implementation of the experiment involves a total of 27 skeleton behaviours: 21 stock planner and 6 device manager behaviours. These behaviours have been made concrete for the different agents in Fig. 2.6. This involves a total of 38 ACP-specific behaviours (see Table 2.1). Together these behaviours have 83 business rules overwriting the placeholder functions in the skeleton behaviours. Each behaviour

Table 2.1 Prototype behaviours per component

Component type	# behaviours	# components	# threads
Initiator	3	1	3
Module	5	1	5
Pick front	12	6	72
Reserve	6	3	18
Case picker	6	6	36
Tray miniload	4	3	12
Palletiser	2	1	2
Total	38	21	150

has been deployed as a separate thread using Jade’s threaded behaviour factory; this makes a total of 150 behaviour threads running on a single pc. Despite the large number of threads, there is no performance issue, because the threads are all simple programs and only a few threads are active simultaneously.

Figure 2.9 shows a partial trace of the agent conversations in Jade’s Sniffer [1]: it shows the messages that are sent between one module, two pick fronts, one reserve, two case pickers, and one tray miniload. We have limited the number of agents to be able to visualise all messages. Figure 2.9 shows three conversations. The first conversation (messages 2.x) involves the module’s AssignWork behaviour (see Fig. 2.3). It starts when the module receives an AssignWork message from the initiation agent, which plays the role of the ERP system. As one can see in Fig. 2.9 the AssignWork behaviour causes several messages of different types to be sent, the last one being an AcceptWork reply to the initiation agent. After the first pallet has been assigned to the module, two conversations start running in parallel. The initiation agent forwards the second pallet to the module (messages 3.x) and the module starts to forward this pallet to the pick front and the reserve (messages 4.x).

Figure 2.10 shows a Gantt chart for the execution of the tasks corresponding to two identical pallets consisting of 40 cases each of a unique product. Each colour represents a product. It clearly shows that the pick fronts do not have any stock: before a case picker can pick a case for a pallet, the tray miniload has to replenish the pick front. The case for the second pallet can be picked directly after the first one, because the trays delivered by the tray miniloads contain more than one case. This means that replenishment is needed only once. Figure 2.10 also shows that the work is divided equally over the different case pickers and tray miniloads.

2.6 Architectural Validation

The previous section showed the functional validation of our reference architecture: we were able to forward work through a hierarchy of stock planners and device managers and perform the necessary replenishment. This section considers the validation of the reference architecture in terms of usability aspects. One of the limitations of

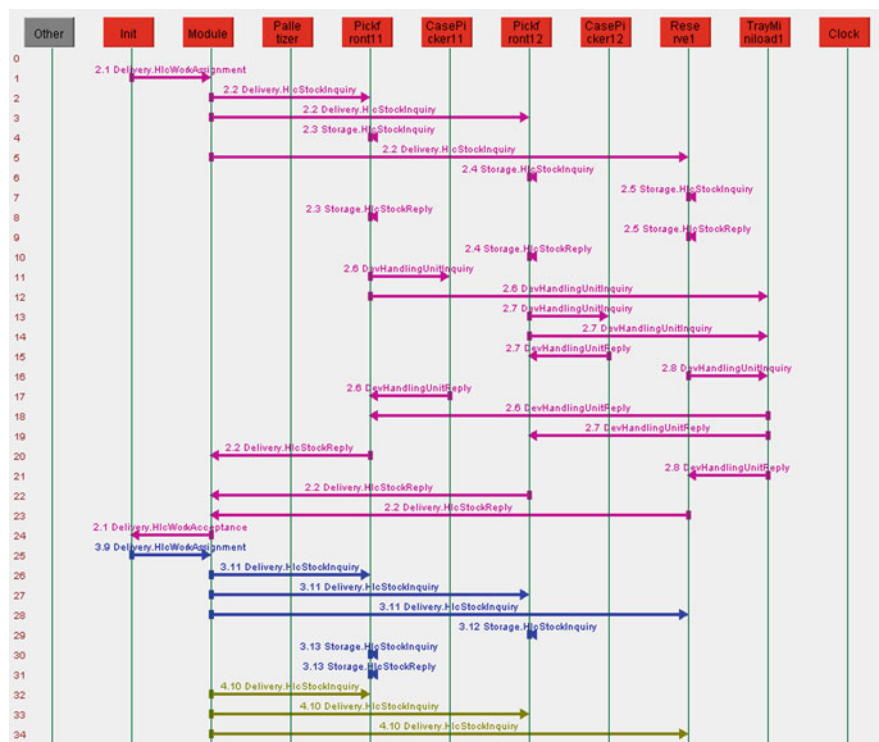


Fig. 2.9 Agent conversations in Jade’s Sniffer

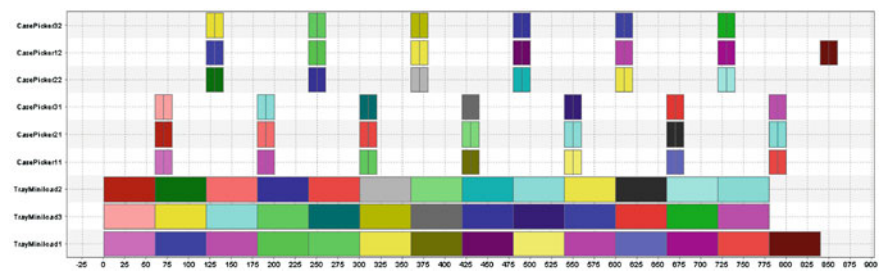


Fig. 2.10 Execution Gantt chart

traditional WMCSs was the limited reusability. This has clearly been addressed by our reference architecture, which is built around generic component types and a collection of generic skeleton behaviours. These generic components can be used and configured for any WMCS.

Reusing the generic components and behaviours has a large positive effect on the WMCS development effort. This effort is limited to the configuration of the system. This involves specifying the connections between the components and defining

Table 2.2 Prototype implementation details

Category	Description	# classes	% classes	# lines	% lines
Generic code	Stock planner skeleton behaviours	21	14	6,064	38
	Device manager skeleton behaviours	6	4	1,072	7
	Initiator behaviours	3	2	271	2
	Miscellaneous	83	56	4,932	31
ACP-specific code	Initialisation code	1	1	814	5
	Stock planner behaviours	23	15	2,115	13
	Device manager behaviours	12	8	612	4
Total		149	100	15,980	100

the business rules to be filled in the abstract architecture, which is built around generic component types and a collection of generic skeleton behaviours. For the experiment described in the previous section, the application-specific business rules and the specification of the agent parameters account for 17 and 5 % of the code (see Table 2.2). The remaining 78 % of the code needs to be written once and can be used for other warehouses.

This 22 % application-specific code is an increase compared to the 15 % of the first experiment described by Verriet et al. [10]. The reason for the increase is the complexity of the second experiment. In the first experiment, each skeleton behaviour had only one application-specific specialisation, whereas the behaviours of the pick fronts and the reserves are specialisations of the same skeleton behaviour in the second experiment. This suggests that the amount of application-specific code strongly depends on the number of WMCS components.

There is more to the WMCS development effort than just the amount of code. Currently, warehouse architects define a warehouse architecture and communicate this architecture to the WMCS developers. This communication has not been formalised, meaning that it is highly ambiguous. The misinterpretations caused by the ambiguity necessitate rework by the designers and hence involves a larger WMCS development effort. Our reference architecture can eliminate a large part of the ambiguity of the architectural communication, because it provides a clear structure for the WMCS architecture. This structure will leave less room for misinterpretation, because the architectural description will contain local business rules with a clearly defined scope and interface.

Because of the standardised interfaces, the components of our architecture are also exchangeable. A change in the stock planner hierarchy can easily be accommodated by altering the parent-child relationships in the WMCS configuration. The new structure may require an alteration of business rules, but this is limited to the components affected by the structural change. Similarly, a change of the network of device managers does not involve a large redesign effort.

To improve the WMCS development even further, we are currently developing a graphical editor (see Fig. 2.11), in which the warehouse architect can select the required warehouse components and their interfaces [6]. Moreover, he will be able to select the components' behaviours and specify the appropriate application-specific

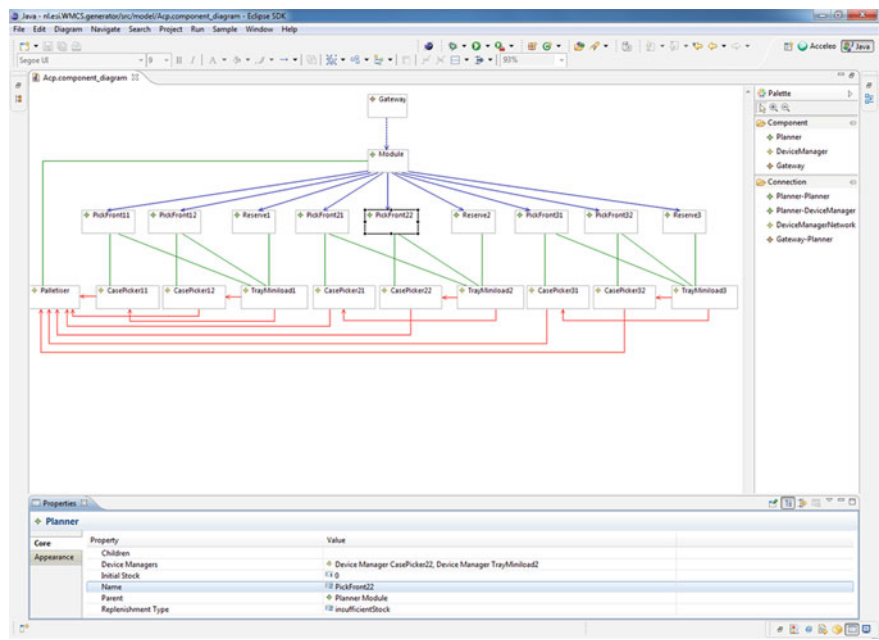


Fig. 2.11 Prototyp WMCS editor

business rules. A large part of a WMCS code can be generated automatically from the architect’s specification using a model-driven software engineering approach (see Chap. 4). The generated code includes the code needed to define the components and their parameters. The remaining code, the application-specific business rules, is to be written by the WMCS designers.

2.7 Conclusion and Outlook

In this chapter, we have described a WMCS reference architecture. This architecture defines the generic WMCS components, their connections, and their generic behaviours. A WMCS can be designed by specifying the parameters of the components and behaviours. The behaviours’ parameters are application-specific business rules, which create concrete behaviours from abstract skeleton behaviours. First experiments successfully demonstrated forwarding of work through a hierarchy of components and order-driven replenishment of local stock.

The experiments have also shown that the reference architecture can greatly reduce the WMCS development effort. The architecture provides a communication framework, which reduces the chance of ambiguity by limiting the scope to local business rules. Moreover, a lot of generic code can be reused: in our experiment, only 22 %

of the code was application specific. The remaining code can be reused for other warehouses by configuring its components, their connections, and their behaviours. Using the editor we are developing, part of the application-specific code and all of the reusable code can be generated automatically.

References

1. Bellifemine F, Caire G, Greenwood D (2007) *Developing multi-agent systems with JADE*. Wiley, Chichester
2. Fleetwood M, Kotak DB, Wu S, Tamoto H (2003) Holonic system architecture for scalable infrastructures. In: *IEEE international conference on systems, man and cybernetics 2003*, vol 2, pp 1469–1474
3. Graves RJ, Wan VK, van der Velden J, van Wijngaarden B (2008) Control of complex integrated automated systems—system retro-fit with agent-based technologies and industrial case experiences. In: *Proceedings of the 10th international material handling research colloquium*
4. Kim BI, Graves RJ, Heragu SS, St. Onge A (2002) Intelligent agent modeling of an industrial warehousing problem. *IIE Trans* 34:601–612
5. Leitao P, Restivo F (2006) ADACOR: A holonic architecture for agile and adaptive manufacturing control. *Comput Ind* 57:121–130
6. Liang HL (2011) A graphical specification tool for decentralized warehouse control systems. SAI Technical Report, Eindhoven University of Technology, Eindhoven
7. Moneva H, Caarls J, Verriet J (2009) A holonic approach to warehouse control. In: *7th international conference on practical applications of agents and multi-agent systems (PAAMS 2009)*, *Advances in intelligent and soft computing* vol 55. Springer, Berlin, pp 1–10
8. Ten Hompel M, Schmidt T (2006) *Warehouse management: Automation and organisation of warehouse and order picking systems*. Springer, Berlin
9. Van Brussel H, Wyns J, Valckenaers P, Bongaerts L, Peeters P (1998) Reference architecture for holonic manufacturing systems: PROSA. *Comput Ind* 37:255–274
10. Verriet J, van Wijngaarden B, van Heusden E, Hamberg R (2011) Automating the development of agent-based warehouse control systems. In: *Trends in practical applications of agents and multiagent systems*, *Advances in intelligent and soft computing*, vol 90. Springer, Berlin, pp 59–66



<http://www.springer.com/978-0-85729-967-3>

Automation in Warehouse Development

Hamberg, R.; Verriet, J. (Eds.)

2012, XII, 244 p., Hardcover

ISBN: 978-0-85729-967-3