Kruger, K and Basson, AH "Implementation of an Erlang-based Resource Holon for a Holonic Manufacturing Cell", Proceedings of the International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing (SOHOMA'14), Nancy, France, November 2014; in Studies in Computational Intelligence Vol. 594, pp 49-58.

Implementation of an Erlang-based Resource Holon for a Holonic Manufacturing Cell

Karel Kruger [a] and Anton Basson [a,*]

[a] Dept of Mechanical & Mechatronic Eng, Stellenbosch Univ, Stellenbosch 7600, South Africa
[*] Corresponding author. Tel: +27 21 808 4250; Fax: +27 21 866 155 206;
ahb@sun.ac.za

Abstract. The use of holonic control in reconfigurable manufacturing systems holds great advantages, such as reduction in complexity and cost, along with increased maintainability and reliability. This paper presents an implementation of holonic control using Erlang, a functional programming language. The paper shows how the functional components of a PROSA resource holon can be implemented through Erlang processes. The subjection of a case study implementation to a reconfigurability experiment is also discussed.

Keywords: Erlang/OTP·Holonic Manufacturing Systems·Reconfigurable Manufacturing Systems·Manufacturing Execution Systems·Automation

1 Introduction

Reconfigurable Manufacturing Systems (RMSs) are aimed at addressing the needs of modern manufacturing, which include [1]: short lead times for the introduction of new products into the system, the ability to produce a larger number of product variants, and the ability to handle fluctuating production volumes and low product prices.

RMSs then aim to switch between members of a particular family of products, by adding or removing functional elements, with minimal delay and effort [2, 3]. For achieving this goal, RMSs should be characterized by [4, 5]: modularity of system components, integrability with other technologies, convertibility to other products, diagnosibility of system errors, customizability for specific applications, and scalability of system capacity. RMSs thus have the ability to reconfigure hardware and control resources to rapidly adjust the production capacity and functionality in response to sudden changes [1, 6].

A popular approach for enabling control reconfiguration in RMSs is the idea of holonic control. Holons are “any component of a complex system that, even when contributing to the function of the system as a whole, demonstrates autonomous, stable and self-contained behavior or function” [7]. Applied to manufacturing systems, a

holon is an autonomous and cooperative building block for transforming, transporting, storing or validating information of physical objects.

Several experimental implementations of holonic control have been done using agent-based programming (such as in [8]), often using JADE as development tool. From our experiences with JADE, we believe there is room for improvement concerning complexity, industry acceptance, robustness and scalability.

This paper describes the implementation of holonic control using Erlang. Erlang is a functional programming language developed for programming concurrent, scalable and distributed systems [9]. The Erlang programming environment is supplemented by the Open Telecommunications Platform (OTP) - a set of robust Erlang libraries and design principles providing middle-ware to develop Erlang systems [10].

Erlang has the potential to narrow the gap between academic research and industrial implementation. This is due to several advantages offered by the Erlang language, such as increased productivity, reliability, maintainability and adaptability.

This paper describes an Erlang-based implementation of the control component for a PROSA resource holon in a reconfigurable manufacturing cell, focusing on:

- The functional components of a resource holon which must be incorporated by the Erlang implementation (section 2)
- A case study which demonstrates the Erlang-based holonic control (section 2)
- The implementation of the functional components of resource holon control through Erlang/OTP processes (section 4)
- The reconfigurability of the resource holon in reaction to changes in the holon's service specification (section 5).

2 Case Study

The case study chosen for the presented Erlang-based holonic control implementation, as shown in Fig. 1, entails the testing of circuit breakers. The station utilizes a pick-'n-place robot to move circuit breakers from an incoming fixture to specified tester slots, in a specified sequence. Upon completion of the testing, the robot removes the circuit breakers and places them in the outgoing fixture. Breakers on the same fixture may require testing in different tester slots, which differ in testing parameters and times.

The robot utilized in the case study is a Kuka KR16 robot, fitted with two pneumatic grippers at the end effector (only one of the grippers is used in this implementation). A mock testing rack with four tester slots is used – the slots are fitted with a spring-loaded clamp to hold the breakers in place during testing.

3 Holonic Control

3.1 Holonic Architecture

There are several existing reference architectures which specify the mapping of manufacturing resources to holons and to structure the holarchy (e.g. [11], [8]). Of these reference architectures, the most prominent is that of PROSA [12].

PROSA (Product-Resource-Order-Staff Architecture) defines four holon classes: product, resource, order and staff. The first three classes of holons are basic holons, which interact by means of knowledge exchange. The process knowledge, which is exchanged between the product and resource holons, is the information describing how a certain process can be achieved through a certain resource. The production knowledge is the information concerning the production of a certain product by using certain resources – this knowledge is exchanged between the order and product holons. The order and resource holons exchange process execution knowledge, which is the information regarding the progress of executing processes on resources.

Staff holons are considered to be special holons which aid the basic holons by providing them with expert knowledge to reduce work load and decision complexity.

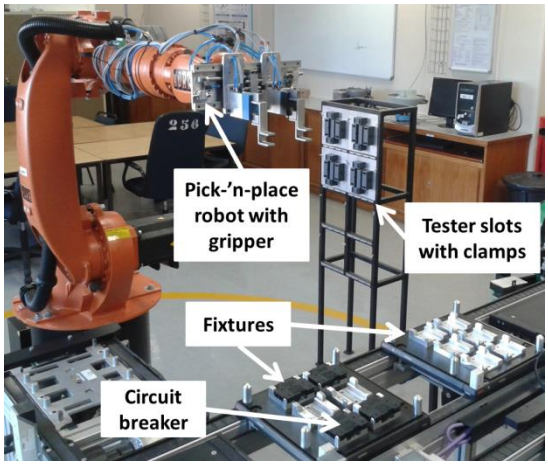


Fig. 1. Circuit breaker test station.

3.2 Resource Holon Internal Architecture

A resource holon requires several capabilities, such as communication, execution control and hardware interfacing. The resource holon model used for the implementation is described in this section.

Individual holons have at least two basic parts [13]: a functional component and a communication and cooperation component. The functional component can be represented purely by a software entity or it could be a hardware interface represented by a

software entity. The communication and cooperation component of a holon is implemented by software. This view of the internal architecture of a resource holon, as illustrated in Fig. 2 (a), is shared by [14].

The communication component is responsible for the inter-holon interaction – i.e. the information exchange with other holons in the system. The decision-making component is responsible for the manufacturing control functions, and so regulates the behavior and activities of the holon. The interfacing component provides mechanisms to access the manufacturing resources, monitor resource data and execute commands in the resource.

4 Erlang-based Control Implementation

4.1 Product, Order and Staff Holon Implementation

Though not the focus of this paper, product, order and staff holons are included in the holonic control implementation. A product holon for each type of circuit breaker is included – this holon contains the information relating to testing parameters and the sequence. For each breaker on the incoming fixture an order holon is launched to drive production. These holons acquire the resource services necessary to complete the testing process. A staff holon is included to facilitate the allocation of resource services to requesting order holons.

4.2 Resource Holon Implementation

For the presented implementation a resource holon was created for the robot and each of the tester slots. While the implementation of the robot holon is complete, the service of the tester slot holons are simulated by replacing their hardware components with timer processes.

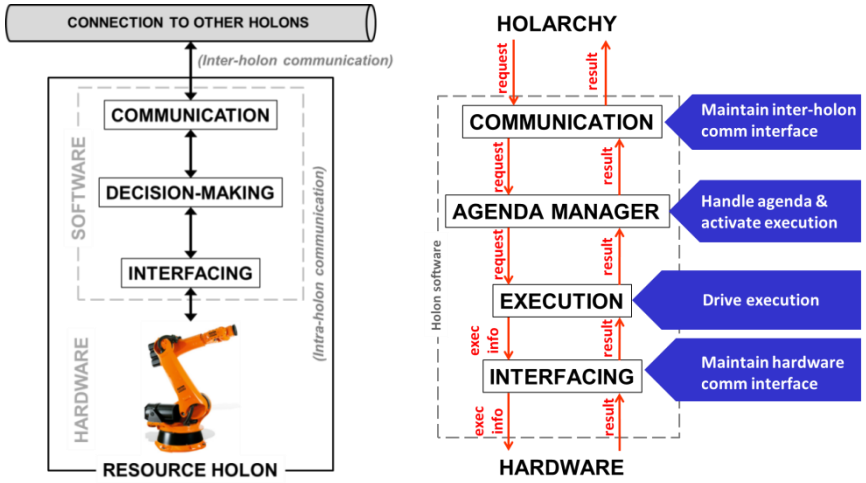
For the robot resource holon, the software components are implemented on a separate PC which interfaces with the hardware via the robot's dedicated controller. The internal holon architecture, inter- and intra-holon communication and the holon functional components are briefly discussed in this section.

Internal Architecture and Communication. For the Erlang/OTP implementation, the internal architecture of a resource holon (Fig. 2(a)) is adapted to that shown in Fig. 2(b). Though the *Communication* and *Interfacing* components are present in both models, the *Decision-making* component in Fig. 2(a) is split into two components, namely the *Agenda Manager* and *Execution* components.

The communication within the Erlang implementation can be classified as either inter- or intra-holon communication. Inter-holon communication is the interchange of messages between the different holons in the system, while intra-holon communication refers to the messages sent between the holon's software and hardware components.

Messages follow the tuple format *{Sender, Message}*, where *Sender* holds the address of the process sending the message and *Message* holds the payload of the message. The payload, for messages received by a resource holon, is in the form of a record.

In addition to the inter-holon communication, Fig. 2 (b) also shows intra-holon communication in terms of *requests*, *results* and *execution information*. As the *Communication* component receives messages from other holons requesting a service, *request* messages are formulated and forwarded to the *Agenda Manager* component. The *Agenda Manager* processes the request and responds to the *Communication* component, which in turn formulates and sends a reply to the requesting holon. The *Agenda Manager* can also send a message to the *Execution* component to activate execution. The *Execution* component parses the message to extract the *execution information* which is passed on to the hardware. The hardware, and subsequently the *Execution* component, gives feedback in the form of *result* messages.



(a) (b)

Fig. 2. (a) A generic (adapted from [14]) and (b) the adapted resource holon model.

Holon Functional Components. The resource holon model of Fig. 2 (b) has four components: *Communication*, *Agenda Manager*, *Execution* and *Interfacing*.

The *Communication* component of the resource holon is responsible for maintaining the inter-holon communication interface. It receives request messages from other holons in the system, evaluates the message type and content and forwards the message to the appropriate holon component. The holon component then returns a result message, which the *Communication* component then sends to the requesting holon.

The component is implemented as a single Erlang process running a *receive-evaluate* loop. This recursive process receives messages from other holons and, by means of pattern matching, identifies relevant messages and then forwards the necessary information to the appropriate holon component. The *Communication* compo-

nent's process also receives intra-holon messages – by the same means the messages are forwarded to the corresponding holon.

The *Agenda Manager* component manages the service of the resource holon. The component manages a list of service bookings by order holons and triggers the *Execution* component, with the necessary execution information, according to the agenda.

The *Agenda Manager* component is implemented through two processes - a *receive-evaluate* loop, for receiving messages, and a generic finite state machine (FSM) behavior (using the OTP *gen_fsm* library). Through pattern-matching, received messages are related to events which cause state transitions in the FSM.

The *Execution* component of the holon drives the hardware actions which constitute the service(s) of the resource holon. It activates a sequence execution of hardware functions, with the necessary execution information.

The *Execution* component is also implemented using a *receive-evaluate* loop, for receiving messages, and a generic FSM behavior. The required sequence of hardware actions is formulated into this FSM. With each execution state, the necessary activation and information messages are sent to the hardware via the *Interfacing* component. The process receives feedback regarding the execution status from the hardware – these messages are then used as events to trigger the transitions between the states. When execution is completed, the execution result is forwarded to the *Agenda Manager* and *Communication* components and ultimately replied to the order holon.

Fig. 3 (a) shows the execution state diagram for the pick-'n-place robot holon. This example shows three states: "ready", "picking" and "placing" – each representing an execution state of the robot. The FSM switches between states in accordance with received messages from the *Agenda Manager* and the hardware.

The *Interfacing* component maintains the communication interface between the Erlang control processes and the program on the robot controller. This component isolates the hardware-specific communication structures from the *Execution* logic.

This component is implemented using a *receive-evaluate* loop for receiving messages and a process for TCP communication. For TCP communication, the process utilizes communication functions from the OTP *gen_tcp* and XML functions from the XMErL libraries [15].

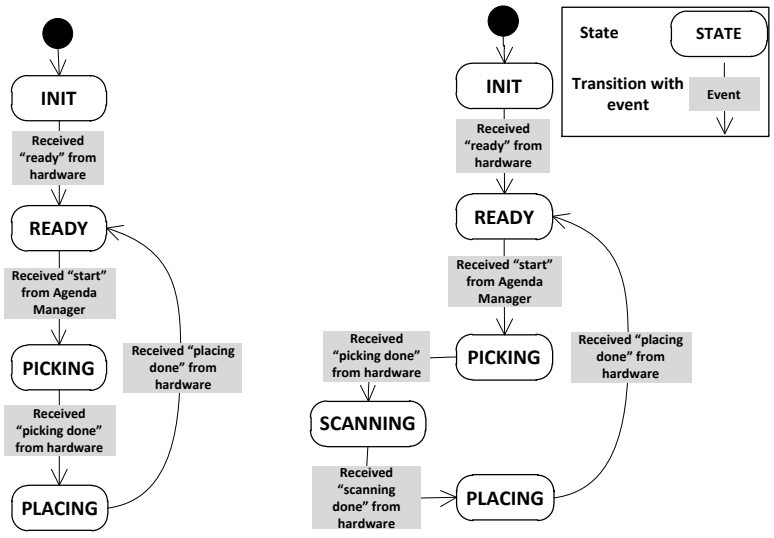


Fig. 3. *Execution* state diagrams (a) before and (b) after adding the scanning function.

In addition to the OTP functionality used in the holon implementation described above, more tools offered by Erlang/OTP are available for enhancing the implementation. Two tools which can be very useful are the *Supervisor* and *Logging* modules. For this implementation, a *Supervisor* process for all the discussed components is included. The *Supervisor* process launches and shuts down the processes in a specified order and restarts the components if they fail. Erlang/OTP includes an *error_logger* module [16] which is used to output error, warning and information reports to the terminal or to file. The format of these reports can be customized according to the needs of the application.

5 Reconfiguration Experiment

A reconfiguration experiment was performed on the case study implementation to demonstrate the reconfigurability of the Erlang-based resource holon. The experiment entailed a change to the service that is provided by the robot holon – more specifically, the service was adjusted to include a scanning operation. The pick-'n-place robot must then, prior to placing, bring the circuit breaker to the vicinity of a scanner.

The added scanning function is only intended for diagnostic purposes and does not entail a change to the product information. The scanning function must be included in the *Execution* component of the robot holon. This means that an additional state must be added to the FSM. The state diagrams of the FSM before and after the addition of the scanning function are shown in Fig. 3.

The following code snippet shows the code for the FSM prior to the addition of the scanning operation:

```
1) init(_) -> {ok,ready,[]}.
2) %STATE: ready
3) ready(Msg=#service{message_type=start,info=#coords{}},_) ->
4)   robot_pi ! {robot_exec,
5)   #service{message_type=start, info=Msg#coords.pick_coords}},
6)   {next_state,picking,[Msg#service.info]}.
7) %STATE: picking
8) picking(Msg=#service{message_type=done,result=true},
9) Coords) ->
10)   robot_pi ! {robot_exec,
11)   #service{message_type=start, in-
    fo=Coords#coords.place_coords}},
12)   {next_state,placing,[]}.
13) %STATE: placing
14) placing(Msg=#service{message_type=done,result=true}, _) ->
15)   robot_am ! {robot_exec,
    Msg#service{message_type=done,result=true}},
16)    {next_state,ready,[]}.
```

The states are defined as function heads (e.g. lines 3, 8 and 14) – the functions take two input arguments: a transition event and the state data. When the transition event occurs (e.g. a message is received), actions are performed and the new state is specified. Here the actions involve sending messages to other processes using the "!" operator (e.g. lines 4, 10 and 15). The new state to transition to is specified by `{next_state, StateName, StateData}`, as is shown in lines 6, 12 and 16. The following code snippet shows the inserted code for the additional scanning operation:

```
7) %STATE: picking
8) picking(Msg=#service{message_type=done,result=true},
    Coords) ->
9)   robot_pi ! {robot_exec,
    #service{message_type=start, info=?ScanCoords}},
10)   {next_state,scanning,[Coords]}.
11) %STATE: scanning
12) scanning(Msg=#service{message_type=done,result=true},
    Coords) ->
13)   robot_pi ! {robot_exec,#service{message_type=start, in-
    fo=Coords#coords.place_coords},
14)   {next_state,placing,Coords}.
15) %STATE: placing
16) placing(Msg=#service{message_type=done,result=true}, _) -> …
```

The inserted code shows the definition of the new *scanning* state and, in lines 10 and 14, updates the transitions from and to the *picking* and *placing* states. The fixed coor-

dinates of the scanner are defined in the module as the macro `?ScanCoords`. The code shown above is added to the *Execution* FSM module and can, through hot code-loading, replace the old FSM code while the holon is operating.

6 Conclusion

RMSs commonly employ holonic control architectures to enable the rapid reconfiguration of hardware and control resources to adjust production capacity and functionality. This paper shows that Erlang/OTP is an attractive solution for implementing holonic control and presents an implementation of a resource holon as example.

The implementation example uses a pick-'n-place robot as resource holon. The robot picks up circuit breakers from a fixture, places them in testers and ultimately removes them again. The paper describes the implementation of the functional holon components as Erlang processes, with specific use of the OTP generic finite state machine library. The reconfigurability of the holon is demonstrated through an experiment where an additional operation is added to the pick-'n-place process. The experiment shows that reconfiguration is easy, as the FSM code offers good encapsulation of functionality and state transitions are clearly defined and easily changed. The reconfiguration could also have been done during holon operation.

Future work will entail the expansion of the Erlang/OTP implementation to the execution control system for an entire manufacturing cell, in which all of the PROSA holons will be incorporated.

References

1. Bi, Z.M., Lang, S.Y.T., Shen, W. and Wang, L., 2008. Reconfigurable Manufacturing Systems: The State of the Art. *International Journal of Production Research*. Vol. 46, No. 4: 967 - 992
2. Martinsen, K., Haga, E., Dransfeld, S. and Watterwald, L.E., 2007. Robust, Flexible and Fast Reconfigurable Assembly System for Automotive Air-brake Couplings. *Intelligent Computation in Manufacturing Engineering*. Vol. 6
3. Vyatkin, V., 2007. IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design. *North Carolina: Instrumentation, Systems and Automation Society, ISA*
4. Mehrabi, M.G., Ulsoy, A.G., Koren, Y., 2000. Reconfigurable Manufacturing Systems: Key to Future Manufacturing. *Journal of Intelligent Manufacturing*. Vol. 13: 135 - 146
5. ElMaraghy, H., 2006. Flexible and Reconfigurable Manufacturing System Paradigms. *International Journal of Flexible Manufacturing System*. Vol. 17:61-276
6. Bi, Z.M., Wang, L. and Lang, S.Y.T., 2007. Current Status of Reconfigurable Assembly Systems. *International Journal of Manufacturing Research*, Inderscience. Vol. 2, No. 3: 303 - 328
7. Paolucci, M. and Sacile, R., 2005. *Agent-Based Manufacturing and Control Systems*. London: CRC Press
8. Leitao, P. and Restivo, F.J., 2006. ADACOR: A Holonic Architecture for Agile and Adaptive Manufacturing Control. *Computers in Industry*. Vol. 57, No. 2: 121-130

9. Armstrong, J., 2003. *Making Reliable Distributed Systems in the Presence of Software Errors*. Doctor's Dissertation. Royal Institute of Technology, Stockholm, Sweden
10. *Get Started with OTP*. [S.a.]. [Online]. Available: http://www.erlang.org (18 July 2013)
11. Chirn, J.L. and McFarlane, D., 2000. A Holonic Component-based Approach to Reconfigurable Manufacturing Control Architecture. *Proceedings of the International Workshop on Industrial Applications of Holonic and Multi-Agent Systems.* pp. 219–223.
12. Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L. and Peeters, P., 1998. Reference Architecture For Holonic Manufacturing Systems: PROSA. *Computers in Industry*. Vol. 37: 255 – 274
13. Kotak, D., Wu, S., Fleetwood, M. and Tamoto, H., 2003. Agent-Based Holonic Design and Operations Environment for Distributed Manufacturing. *Computers in Industry*. Vol. 52: 95–108
14. Leitao, P. and Restivo, F.J., 2002. A Holonic Control Approach for Distributed Manufacturing. *Knowledge and Technology Integration in Production and Services: Balancing Knowledge and Technology in Product and Service Life Cycle*. pp. 263–270. Kluwer Academic Publishers.
15. *XMErL Reference manual*. [S.a.]. [Online]. Available: http://www.erlang.org/doc/apps/xmerl (28 March 2014)
16. *Erlang Kernel Reference Manual*. [S.a.]. [Online]. Available: http://www.erlang.org/doc/apps/kernel (28 March 2014)