Agile Architecting of Distributed Systems for Flexible Industry 4.0

Christensen, Henrik Barbak; Jepsen, Sune Chung; Worm, Torben

Published in:
2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)

DOI:
10.15439/2021F17

Publication date:
2021

Document version:
Final published version

Document license:
CC BY

Citation for pulished version (APA):
Christensen, H. B., Jepsen, S. C., & Worm, T. (2021). Agile Architecting of Distributed Systems for Flexible Industry 4.0. In M. Ganzha, L. Maciaszek, M. Paprzycki, & D. Slezak (Eds.), *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)* (pp. 533-536). Polish Information Processing Society. Annals of Computer Science and Information Systems Vol. 25 https://doi.org/10.15439/2021F17

DOI: 10.15439/2021F17
ISSN 2300-5963 ACSIS, Vol. 25

Agile Architecting of Distributed Systems for Flexible Industry 4.0

Henrik Bærbak Christensen
Computer Science
University of Aarhus
Aarhus, Denmark
Email: hbc@cs.au.dk

Sune Chung Jepsen, Torben Worm
Software Engineering
University of Southern Denmark
Odense, Denmark
Email: {sune, tow}@mmmi.sdu.dk

***Abstract*—Small and medium sized businesses within mechanical manufacturing cannot benefit from Industry 4.0 automation as small production batches are unable to pay for up-front robotic configuration and programming costs. In this paper, we report on early results from a project aiming at developing a software architecture supporting fast, easy, and flexible reconfiguration of a robotic manufacturing process, using an agile and prototyping approach.**

I. Introduction

ROBOTIC manufacturing is well adopted in for instance the automotive industry. Such manufacturing is characterized by production of large volumes of nearly identical products which can justify high cost of setting up the production line and programming robots in the production-line. However, small and medium-sized businesses (SMB) in mechanical production often have low production volumes, often just a single or less than 10 products. Thus, adopting robotic manufacturing, Industry 4.0—*the intelligent networking of machines and processes for industry with the help of information and communication technology* [7], is challenging for SMBs. In our project, we are exploring flexible production, customization, and handling changing requirements in collaboration with a number of Danish SMBs.

Our main contribution is early results from applying *architectural prototyping* to formulate distributed architectures for Industry 4.0 with an emphasis on flexibility as a central quality attribute. A second contribution is early architectural insights from this work, which adopts an agile and run-time focus in contrast to prevalent work that achieves flexibility through elaborate ontologies [8], [9] which in turn require intensive up-front engineering efforts [3].

II. Background

The project is a collaboration between three SMB within the mechanical production area (machine shops) as well as two Danish universities with competences within robotics and software architecture.

The main research challenge is:

> *Design a distributed systems software architecture that allows flexible production specification, adaptable to a small machine shop, providing high usability by workers trained in mechanical production.*

Ideally, a skilled worker (but with little prior computer science training) should be able to set up individual machine functions (metal cutting, shaping, drilling, assembling, packaging, etc.) as well as workflow (process order, movement of product between machines, etc.) fast and easy.

A schematic example is show in Figure 1 in which a worker (1) defines a workflow plan to be handled by the platform, which instructs transport robots (3 + 9), to move proper materials from a raw warehouse (2) to specified production cells/programmable robots (5 + 7) that do assembly, drilling, cutting, etc. Production cells may also receive/deliver materials using a magnetic track (4 + 6), before the final product is delivered to a finished goods warehouse (10).

III. Agile Architecture Design

Architecturally, a robotic machine shop is a *distributed system*, having independent nodes with specialized capabilities, adaptable by programming. The key challenge is designing a software architecture that supports flexible (re)configuration, ease of defining processes and workflow, as well as efficient/cost-effective production.

As a research project, another goal is to experiment with the architectural design space in a efficient manner in order to allow stakeholders early and agile feedback. *Architectural prototyping* [2], [4] matches these requirements, as it emphasizes architectural learning and exploring, using lightweight development of executable demonstration systems.

Architectural prototyping is an incremental and iterative process in which architectural design is postulated as a hypothesis, programmed in an architectural prototype (AP), and next validated for feasibility. Based upon the outcome, the AP is either rejected or refined, similar to a scientific process. This way the architectural design space is explored and refined, more than rigorously designed and evaluated up-front.

We therefore hypothesized a software architecture for a robotic machine shop, as exemplified in Figure 1, to be a *distributed system of programmable nodes (production cell, transports, warehouses, etc.) orchestrated by the software architecture pattern "Blackboard"* [1].

In the Blackboard pattern, individual nodes, *knowledge sources*, contribute data and events to the *blackboard* (i.e. a repository) while a *control plane* actively monitors the

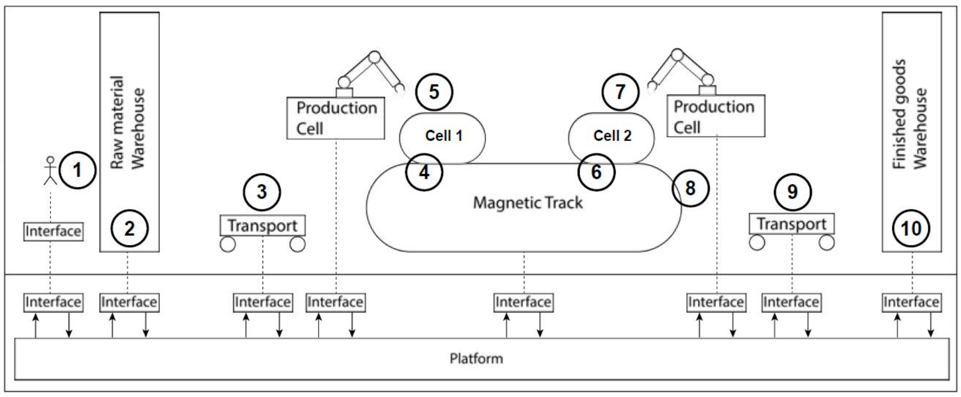

Fig. 1. Schematic Machine Shop [6]

TABLE I
DEVELOPED ARCHITECTURAL PROTOTYPES. SOURCE OF DATA IS A) DEVELOPER'S WORKING HOUR REGISTRATION, B) GIT LOG MESSAGE TIMESTAMPS, AND C) GIT BLAME LOGS OF DEVELOPMENT DIARIES FOR EACH AP.

AP No.	Goal	Outcome	Hour count
1	Establish Modelling and Blackboard Pattern	Accept	9h
2	Demonstrator for stakeholders	Accept	3.5h
3	Workflow. Insight: Carrier concept missing	Accept	7h
4	Knowledge Engine (JESS) Learning	Accept	3h
5	Carrier Introduced. JESS Integration	Accept/Failure	10.5h
6	EasyFlow Learning	Accept	1.5h
7	Workflow using EasyFlow. Demonstrator	Cond Accept	10h

blackboard, picking up state changes and issues new actions to be performed by nodes. To exemplify in a robotic machine shop context, a production cell may notify the blackboard that drilling a hole in a plate is finished, which trigger that the control plane tells the transport robot to move the finished plate to the assembly robot, etc.

Our first AP, see Table I, developed the core concepts adhering to the blackboard architecture, with a strong emphasis on *modeling physical objects and processes with computational equivalents* that support fast experimentation. Central examples of "computational equivalents", developed in our APs are:

- *Physical material*: Modelled by strings. Example: A bolt is just the string "B", a metal plate of 100x20x10 mm is just "P/100-20-10", etc.
- *Production cells*: Modelled by threads/processes, that receive materials from an in-queue, perform a "Production cell function" on the materials, before emitting it to its out-queue.
- *Production cell function*: Modelled by Strategy pattern, an algorithm to process material from one form to another. Example: Drilling a 3mm hole in the above plate "P/100-20-10" at position (10mm, 20mm) will return material "drill/3-10-20(P/100-20-10)". Note how the string just is a recursive specification of functions applied. To simulate time taken for a given function, delays are part of a cell function's execution.
- *Transport of Materials*: Also modelled by threads, but their "function" is set to move material from the out-queue of one production cell to the in-queue of the another. Essentially, just copying strings from an out-queue to an in-queue, with a delay.
- *Warehouse*: Again, just a thread, whose "function" is either to provide (raw warehouse) or store (finished goods) material. That is, it is just a collection of strings.
- *Queues (at cells)*: To simplify we used a blocking queue with just room for one material/string. Later changed to contain *Carrier* objects, see below.
- *Control plane*: Initially, we hard-coded the simplest possible production scenario, we could think of—one cell drills a hole in a plate, which is then moved to a second cell that screws a nut+bolt through the hole in the plate. That is, going from raw materials (P, B, N) to "screw(B, N, drill(P))" (dimensions omitted for clarity).

Executing an AP simply produces log messages from each thread outlining the actions taken by each transport ("MoveBot" in output below) or production cell ("Station" below), as exemplified in Figure 2.

Note the efficiency of the approach in exploring the control plane architectural aspects (Table 1): Only 9 hours was spent to establish an architectural sketch, defined core concepts, and their computational equivalents. A further 3½ hours was spent to polish the AP into a form that allowed demonstration to the project's SMB stakeholders.

```
[INFO] MoveBot :: MoveBot 1 - PICK UP material 'P/100-20-10'
[INFO] MoveBot :: MoveBot 1 - Start moving to Station 'Drill Station'
...
[INFO] MoveBot :: MoveBot 1 - DELIVER TO Drill Station
...
[INFO] Station :: Drilling 66%...
[INFO] Station :: Drilling 100% - producing 'drill/3-10-20(P/100-20-10)'
[INFO] Station ::  -- Adding to OUT QUEUE
[INFO] MoveBot :: MoveBot 1 - retrieved job:
       MoveJob{source='Drill Station', destination='Assembly Station'}
```

Fig. 2. AP-2 demonstrates workflow and actions performed through log messages (...indicates portions omitted for brevity)

The conclusion of AP-1 and AP-2 was that stakeholders judged that core modeling concepts were feasible, but "programming the control plane was tedious". The APs only supported a single workflow scenario, and involved lots of tedious and hand-coded handling of threads and queues. Another outcome of the AP-2 demonstration session was the need to introduce strong support for "state machines to model workflows".

APs 3–7, in Table 1, represent steps and sidesteps in exploring implementing a flexible, usable, control plane based upon state machines. AP-3's focus was on introducing simple statemachines, but quickly lead to the conclusion that a *Carrier* concept was missing:

- *Carrier*: A physical tray, organizing a set of materials in predefined positions for easy idenfication by a robot, ala "Pick the nut in position 3", see Figure 3. Our computational equivalents was an array, indexing a set of strings (representing materials), as well as the product's associated state machine.
- *Control plane*: Rewritten from the fixed workflow of AP-1 into a listener on any state change (production cells or transports finishing their tasks) which in turn leads to deciding on next state transition.

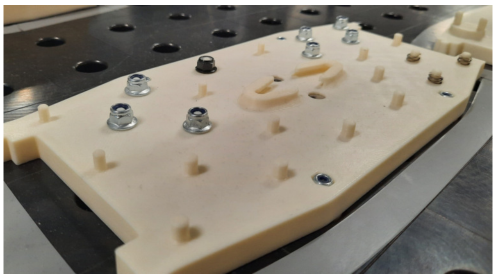
Fig. 3. A *carrier* from the Robotic Lab, holding nuts and springs.

The carrier concepts was introduced and validated in AP-5. Two APs were dedicated to exploring suitable libraries for programing state-machines (AP 4+6), before settling on EasyFlow for the stakeholder demonstrator. The (so far) final AP-7 was demonstrated at a second workshop to stakeholders—showing three workflows, producing two complex and one simple product.

One key outcome was that the carrier besides holding materials, also embody the state of the materials from its journey from a set of raw materials to the final product. Alas, the workflow statemachine is directly tied to the carrier. A second outcome was how the Blackboard architecture automatically optimizes the flow of material and processes in the production line: As soon as a production cell has finished, its carrier is available at the out-queue and the blackboard/control plane is notified. The control plane makes the state transition of the carrier's state machine, typically finding an idling transport robot to pick up the carrier; or selecting an available production cell that can serve the manufactoring task at hand: drilling, cutting, assembly, etc.

However, while our AP-7 validated the architectural design, the definition of workflows via state machines was still requiring low level programming, far from the required usability requirements.

Never-the-less, only a total of 44.5 hours was spent to establish a sound architectural basis validating an architectural approach based upon the blackboard pattern, carriers associated with product's state machines, and central concept implementations—as well as rejected several ideas, such as using knowledge engines, with little wasted effort. Future work focuses on bringing the state-machine programming into a user friendly format, likely exploring visual tools for generating state machines, like Visual Paradigm, SinelaboreRT, and of course proof of concept of the architecture by letting it control real manufacturing production cells and transports at the Robotic Lab at University of Southern Denmark.

A. *Preliminary Architecture*

In Figure 4, our preliminary architecture is sketched, using UML class diagram notation: Associations are annotated with essential behavior. Robot Units are general and represent independent processes such as the production cells, the warehouses, and transports like magnetic track or transport robots. The Blackboard controls them by upload programs, "functions", to them. Carriers contain the product as a set of (partially processed) materials, as well as the state machine representing the state of the partially processed product. Any state transitions (like a cell finishing and moving the carrier to its out queue) notify the Blackboard which determine next actions which is always "moving" carrier from one unit's out queue to the next unit's in queue. The actual sequence of units to visit and functions to apply is determined by the particular state machine of the product.

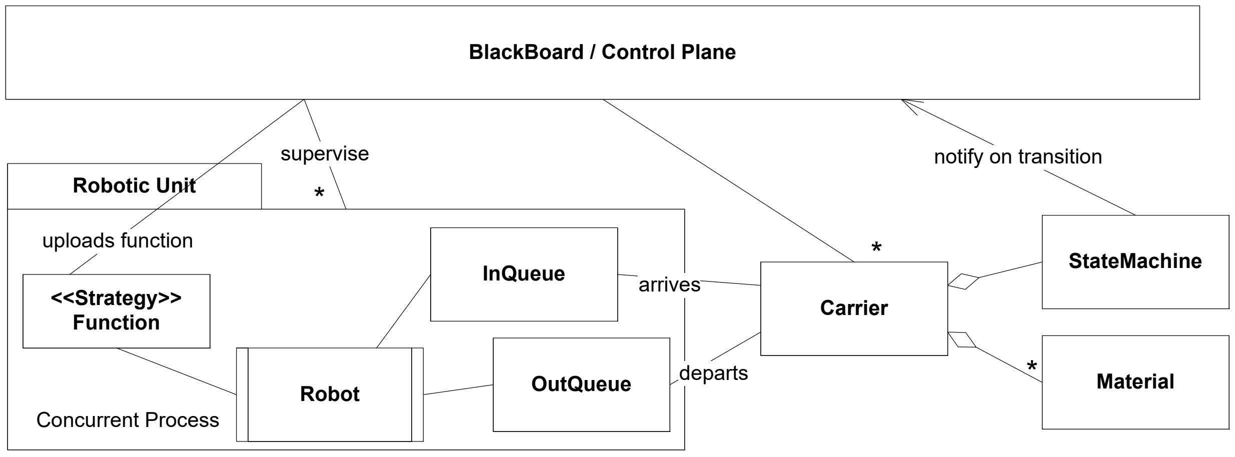

Fig. 4. UML Class diagram of preliminary architecture

We argue that even this preliminary architecture achieves a degree of flexibility, as the input for any given product is just the specification (set of functions, state sequence): that is, define the set of processings (drilling, assembly, cutting, etc.) as well as the ordering—drilling must be performed before assembling, which again must be performed before storing in the final warehouse, etc.

IV. Conclusion

In this paper, we have presented initial results on agile architecture development for distributed systems for SMB machine shops, with an emphasis on flexible and easy reconfiguration of manufacturing. We have presented results from early architectural design work based upon architectural prototyping, and shown how this technique provides fast feedback in the architectural work, allowing us to establish a solid architectural basis for further work in less than 45 hours of staff time. Furthermore, we have presented our suggestions on how the complex mechanical and distributed nature of a machine shop can be translated into computational equivalents that allows a fast and experimental development cycle in collaboration with workers in mechanical production.

Acknowledgements

The authors wants to acknowledge the financial support of InfinIT [5].

References

[1] Paris Avgeriou and Uwe Zdun. Architectural Patterns Revisited—A Pattern Language. In *Proceedings of 10th European Conference on Patterns Languages of Programming*, 2005.
[2] J. E. Bardram, H. B. Christensen, and K. M. Hansen. Architectural Prototyping: An Approach for Grounding Architectural Design and Learning. In *Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, pages 15–24, 2004.
[3] Haibo Cheng, Peng Zeng, Lingling Xue, Zhao Shi, Peng Wang, and Haibin Yu. Manufacturing Ontology Development based on Industry 4.0 Demonstration Production Line. In *2016 Third International Conference on Trustworthy Systems and their Applications (TSA)*, pages 42–47. IEEE, 2016.
[4] Henrik Bærbak Christensen and Klaus Marius Hansen. An Empirical Investigation of Architectural Prototyping. *Journal of Systems and Software*, 83(1):133–142, 2010.
[5] Infinit website. https://infinit.dk/om-infinit/, 2020.
[6] S. C. Jepsen, T. I. Mørk, J. Hviid, and T. Worm. A Pilot Study of Industry 4.0 Asset Interoperability Challenges in an Industry 4.0 Laboratory. In *2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 571–575, 2020.
[7] Plattform Industrie 4.0. Plattform Industrie 4.0 - What is Industrie 4.0? https://www.plattform-i40.de/PI40/Navigation/EN/Industrie40/WhatIsIndustrie40/what-is-industrie40.htm. Accessed: 2020-11-24.
[8] Emanuel Trunzer, Ambra Calà, Paulo Leitão, Michael Gepp, Jakob Kinghorst, Arndt Lüder, Hubertus Schauerte, Markus Reifferscheid, and Birgit Vogel-Heuser. System architectures for Industrie 4.0 applications. *Production Engineering*, 13(3-4):247–257, 2019.
[9] Jiafu Wan, Shenglong Tang, Di Li, Muhammad Imran, Chunhua Zhang, Chengliang Liu, and Zhibo Pang. Reconfigurable Smart Factory for Drug Packing in Healthcare Industry 4.0. *IEEE Transactions on Industrial Informatics*, 15(1):507–516, 2018.