Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 00 (2019) 000–000

www.elsevier.com/locate/procedia

CIRP Manufacturing Systems Conference 2019

A Participatory Programming Model for Democratizing Cobot Technology in Public and Industrial Fablabs

Tudor B. Ionescu[a]*, Sebastian Schlund[a]

[a]Human-Machine Interaction Group, Institute of Management Science, Technische Universität Wien; Theresianumgasse 27, 1040-Vienna, Austria.

* Corresponding author. Tel.: +43 (1) 58801-33057; fax: +43 (1) 58801-33098. *E-mail address:* tudor.ionescu@tuwien.ac.at

Abstract

Collaborative robots (cobots) can improve productivity by flexibly assisting production workers. Nevertheless, due to missing use cases, cobots have yet to be widely adopted in manufacturing. Democratizing robot programming by introducing cobots in public fablabs (or makerspaces) could lead to new applications transferrable to industry thanks to the creativity of interested laypersons. Fablabs could also mitigate the risk of unemployment due to automation by training cobot programmers for manufacturing jobs. In support of the proposed approach this paper introduces a participatory three-layer programming model designed to enable people with little or no programming experience to become effective cobot operators. The model builds on task-oriented programming environments extending them by more versatile yet intuitive programming environments. The model has been successfully evaluated on the basis of a real human-robot assembly application.

Keywords: Intuitive robot programming, participatory programming model, cobots, human-robot interaction, assembly system

1. Introduction

Industrial collaborative robot (cobot) use still lags behind initial expectations of a boost to industrial applications once new players will enter the market and prices will drop to 10-20,000$ per system. Although some vendors now offer cobots within this price range, the estimated optimal cost of robot operation in production of merely 5-10% that of a human worker has not been reached, especially for human-robot collaboration (HRC) applications. This has several reasons, including the following:

(1) The current legal requirement to certify cobot applications individually prevents the transfer of certified applications from one use case to another [1] and certification services are provided by a limited number of organizations. However, experts are optimistic concerning the possibility of certifying entire application classes within the next years [2].

(2) The overhead of implementing safe and cost-effective cobot applications is now cited as a major impediment to market penetration. For example, [3] show that the implementation of such applications require formally trained robot programmers and around 4 person months of development time.

(3) Knowledge and experience in planning, designing, and implementing cobot applications in manufacturing are limited to a small number of system integrators and application-oriented research institutes. This may lead to path dependencies and vendor lock-in on the part of manufacturing companies as well as unfair access to implementation know-how on the part of small and medium-sized companies that cannot afford to hire robot programmers.

(4) There is a lack of publicly available cobot experience reports, best practices, use cases, and application libraries. Available information is rather sporadic and lacks implementation details, such as parameterizations and invested resources. Sharing and reusing knowledge and experience on a wider scale is not encouraged because of the perceived need to protect that kind of knowledge in an increasingly competitive manufacturing market.

To address the latter three challenges, we argue that cobot technology should be democratized [4,10] in the sense of making it more accessible for non-experts. This—we believe—will help to leverage the "value-free" creativity and problem-solving abilities of current factory employees and laypersons to the end of generating know-how in the form of replicable cobot use cases and applications. By "value-free" it is meant that assembly workers, production planners, and interested laypersons are likely to come up with simple yet effective solutions for human-robot interaction, which can enrich the functionality of off-the-shelf robot skills developed by roboticists and software engineers. Currently, many robot programing models build upon the same languages and tools used to create robot APIs (application programmable interfaces) and require advanced programming skills. Other vendors provide proprietary programming languages, which cannot be easily extended. In this context, we aim to compensate for missing programming and analytical skills on the part of cobot operators by leveraging the various talents and abilities of technically-apt non-experts. To this end, we propose an approach to democratizing cobot technology by making it available in public fablabs (or makerspaces). Fablabs offer access to well-equipped laboratory-like spaces for so-called makers and hackers interested in implementing their own creative ideas or creating unique—or, in manufacturing parlance—*lot size 1* products. Makers are usually technically-apt laypersons but they may also be students or self-employed persons looking to implement applications for leisure or useful purposes. While fablabs commonly provide advanced manufacturing equipment, such as 3D printers, laser cutters, CNC-mills, etc., and associated training workshops, from reasons of safety as well as (presumably) lack of knowledge and use cases they currently do no yet offer cobots. And yet, the open, collaborative, creative, and fostering culture of the maker movement renders fablabs ideal for crowdsourcing cobot use cases and applications in a *lot size 1* scenario.

In support of the proposed approach this paper introduces a participatory programming model designed to enable people with little or no programming experience to become effective cobot operators. The model builds on task-oriented programming environments, extending them by more versatile yet intuitive programming environments. We begin with a review of related work before turning to a more detailed description of the proposed programming model. Then we discuss an evaluation experiment and provide an outlook on a publicly funded research project aimed at introducing cobots to a makerspace.

Related Work—Existing cobot programming environments, such as RoboDK [7] or ABB RobotStudio [8], provide rich graphical interfaces for creating procedural programs composed of chains of robot movements and other program structures like functions, variables, positions in the robot's coordinate systems, and different flow control structures. These simplified programming environments still require a certain degree of computational thinking [9] on the part of robot operators. Other environments, like the web-based Franka Emika Desk environment [5], implement simple task-oriented programming models assisted by haptic guidance of the robotic arm and control of the end effector (e.g., gripper). Assembly workflows can be created by dragging and dropping automated handling functions (called apps) in a sequence. Yet, Franka Emika's business model is based on selling custom robot skills bundled as apps. New apps can, however, only be created by software engineers knowledgeable of the robot's complex joint model and having access to the Franka Emika code base. This model may thus prevent people without advanced programming skills from extending the off-the-shelf app collections.

Other approaches to programming cobots recognize the need for operators to also (re)program robots on the shop-floor to some extent [5,11-13, 22] in a way reminiscent of the *DevOps* practices from software engineering. As a result, an increasing number of these environments are web-based and more intuitive. Nevertheless, web-based environments are still mostly driven by the vision and expertise of roboticists and software engineers although they are designed to be used in the manufacturing domain. This situation is perhaps best reflected in the skill-oriented programming model for (semi-)autonomous manufacturing robots implemented in RAZER [11] and other frameworks (e.g., [14]). Within these models, robot skills are expected to embody complex automated handling functions inspired by norms such as VDI 2680 [15]. To implement the skill model, RAZER builds upon two separate roles and perspectives of *experts* and *users*. The *expert* role is assumed by software engineers or roboticists who implement skills in Java, C++, Python, and other programming languages and provide them as plugins in a way reminiscent of the APP paradigm for smart phones. Once plugged-in, robot skills can be used for application development by *users* with limited or no programming skills. This "off-the-shelf" robot skill model has several inherent limitations when it comes to practical manufacturing applications. First, given the heterogeneity of factories (layout, people, tooling, materials, etc.), software skills cannot encompass all the relevant parameters of the physical environment for all potential applications at the moment of their creation. This lack of versatility of most robot skills may limit their applicability to a predefined set of use cases envisioned by their creators. Second, the tacit long term knowledge [16] of skilled workers cannot be universally represented in software. The ineffability of tacit knowledge [17] implies that, even in optimistic scenarios, robot skills will only achieve the performance of clumsy workers in untested situations because software cannot represent all the details of "skilled knowing" [17]. Machine learning is often regarded as a future solution to the problem of representing tacit knowledge in software [18] although such techniques are still in their infancy. And third, the complexity and secrecy of the internal workings of off-the-shelf robot skills prevents workers and other experts to adapt the software quickly so as to facilitate their application in situations unforeseen by their creators.

2. A Participatory Cobot Programming Model

Currently, cobot programming is generally reserved for experts, while the end users of the technology—assembly workers and production planners—usually do not possess the required programming skills. Also, the long term knowledge of workers and planners cannot easily be represented in software [23, 24]. In response to this situation, we propose a cobot programming model, which leverages the long-term knowledge of workers as well as the newest technological advancements.

Table 1. Participatory three-layer cobot programming model.

Programming Layer	**Roles and Expertise**	**Required Training**	**Supporting techniques**
Layer 1: *Basic assembly workflows* (robot movements and tool actuation)	*Assembly workers* and *laypersons* with some assembly experience	Some technical training (e.g. professional school)	Mounting and assembly devices, multimodal teach-in
Layer 2: *Block-based programs* (task blocks, variables, and control structures)	*Industrial engineers* with computational thinking abilities and technical intuition	Formal technical training and a programming course	CAD-modelling, 3D-printing, laser cutting
Layer 3: *Advanced functionality* (databases, connectivity, etc.)	*Software engineers* with advanced programming skills	Formal software engineering training	Internet / intranet, databases, cloud, MES

The proposed model is similar to [14] in purpose and, partly, in the technologies being used in that it aims to enable unexperienced users to become effective robot programmers and/or operators using block-based visual programming. However, it differs fundamentally from [14] in that it does not build on complex off-the-shelf robot skills, which may induce dependencies on vendors and robotics experts. Instead, the three-layer model leverages the skills, experience, and knowledge of existing factory employees (assembly workers and industrial engineers) to accomplish what in [14] is expected of off-the-shelf robot skills. Starting from the observation that robot skills bundle and orchestrate a series of robot movements in a meaningful way, we argue that much of this meaning resides in the expertise of shop-floor planners, workers, and other manufacturing professionals. In this sense, the identification of reusable assembly operations and automated handling function workflows becomes the result of a situated, context-dependent analysis of robot programs developed by persons who best understand the meaning and role of the functions and assembly workflows being automated. Table 1 illustrates the different layers of the proposed programing model with their associated program structures, required expertise, roles, and supporting techniques. The model fosters the development of reusable assembly workflows, which are easy to understand and reproduce even by people with limited or no programming skills (i.e., laypersons). Additional programming structures, such as loops and control structures, requiring a certain level of computational thinking, can be implemented on top of sequential assembly workflows developed by people with little or no programming skills.

Layer 1—Creating **basic assembly workflows (BAW)** using an intuitive multimodal programming environment can be mastered by assembly workers and laypersons with some assembly experience, technical talent, and intuition. These basic workflows represent the core of any automated handing task because they require precise positioning and many repetitive adjustments and can be implemented by factory workers. The key to success is to keep the workflows simple, which requires additional support in the form of mounting and assembly devices and jigs designed for the task at hand. Such devices can be created on-site using, for example, 3D printers. In addition, multimodal robot interaction features are required to facilitate the intuitive teach-in of robots by laypersons.

Layer 2—Employees with a formal technical training (e.g. industrial engineers) and a basic programming course can develop more complex **block-based programs** on top of BAWs. In these higher level environment, BAWs can be wrapped by specialized blocks or functions. If, for example, Franke Emika Desk [5] is used for creating BAWs and Blockly [6] for block-based programs, then a specialized block having the name of a taught-in robot task can be used to invoke it in the robot's native programming environment. This can be achieved, for example, through code injection into web-based programming environments, as described in the next section. This way, several BAWs can be orchestrated using conditional control structures, loops, and variables, which enables the development of more complex assembly workflows. This programming layer is conceived for industrial engineers, who can use it to extend the BAWs created by workers. Engineers and planners can also support the CAD-based design and physical realization of 3D-printed and/or laser-cut assembly devices and jigs. Workers, planners and industrial engineers can thus work in teams to create new human-robot assembly applications without necessarily requiring the support of robotics experts.

Layer 3—While the first two programming layers would likely suffice for most (human-robot) assembly applications, software engineers and robotics experts may create more **advanced functionality** by extending the programming environments at the first and second layers. Custom blocks may thus be created, for example, to facilitate gesture-based human-robot interaction or to enable connectivity with databases, MES systems, or the Cloud. While, for example, in the RAZER programming model, software engineers were required to provide new robot skills for drilling, pick & place and similar handling functions, which could then be used by robot operators, in the proposed 3-layer model software engineers are needed for very specific high-level extensions to an otherwise fully operational 2-layer robot programming model. By separating the development of advanced functionality by software engineers from that of developing human-robot assembly applications, manufacturing companies can reduce costs while still benefitting from the most advanced robot technologies.

In order to facilitate a peer learning process between layer 1,2 and 3 programmers as well as to ensure safety during development, the "pair programming" principle from agile software development [19] can be used in combination with the proposed programming model. In pair programming, two programmers work in teams of two to produce application code and/or logic, whereby one member (called *driver*) writes the code and the other one "continuously and actively observes the driver's work, watching for defects, thinking of alternatives, looking up resources, and considering strategic implications" [19]. These roles are seen as equal and active participants in the process and switched periodically. Pair programming can thus help to solve HRC problems faster while improving safety and facilitating a learning process for all participants by leveraging heterogeneous skills, talent, and competencies shared in tight collaboration loops between team members. The development model and process should ideally be conducted in a fablab environment, in which collaborations are open and non-hierarchical and supporting technologies and expertise, such as 3D-printing or laser cutting, are available in close proximity. This also enables the agile, rapid prototyping of assembly devices needed to leverage the full potential of workers and cobots.

3. Implementation and Evaluation

This section provides technical details about a prototypical implementation of the proposed 3-layer programming model using Desk (Franka Emika Panda robot's programming environment [5]) and Blockly [6]. The web-based Desk environment allows non-experts to create cobot applications without previous knowledge of robotics and programing, thus providing a suitable "layer 1" development environment for laypersons and assembly workers. Google Blockly has been successfully used in many educational projects as a tool for teaching and learning the basics of programming. Some robot programming environments build upon it to generate C++ or Python code for low-level APIs or ROS [14, 22]. The main advantage of Blockly for non-expert programmers is that, for simple applications, it only requires computational thinking abilities and a basic understanding of general programming principles and structures on the part of users, who are not required to master the syntax of any programming language. For this reason we regard Blockly as a suitable "layer 2" programming environment, which can be extended by "layer 3" programmers as needed to provide advanced functionality.

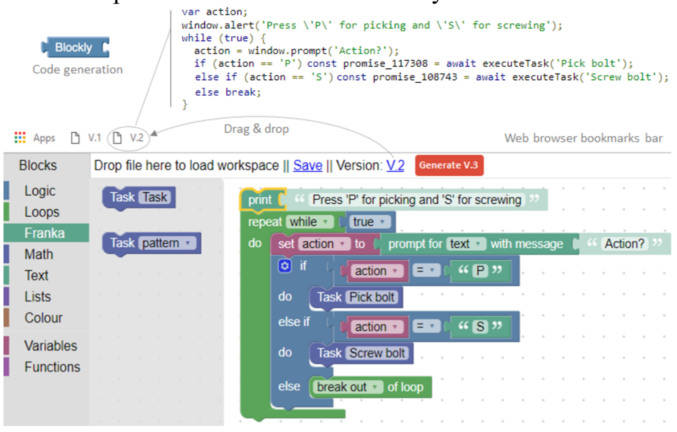
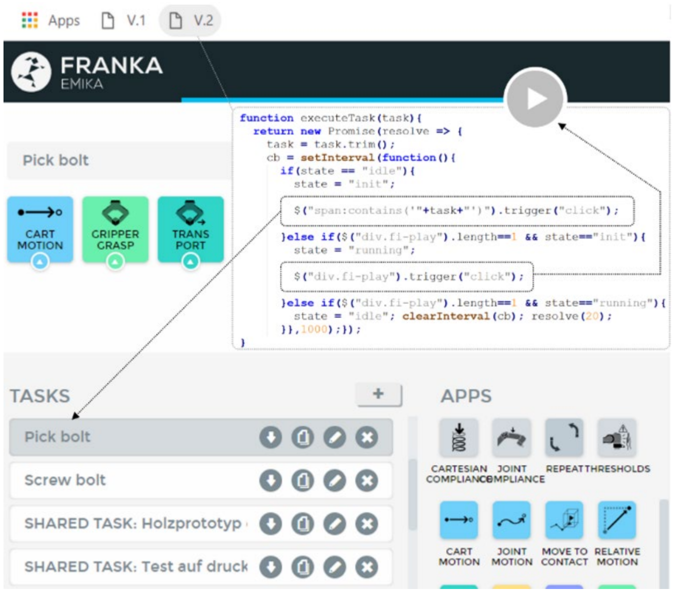

```
var action;
window.alert('Press \'P\' for picking and \'S\' for screwing');
while (true) {
  action = window.prompt('Action?');
  if (action == 'P') const promise_117308 = await executeTask('Pick bolt');
  else if (action == 'S') const promise_108743 = await executeTask('Screw bolt');
  else break;
}
```

Fig. 1. A basic example for generating bookmarklets in Blockly.

Building upon the Desk programming environment and Blockly we implemented the 3-layer model described above using the JavaScript (JS) "bookmarklet" technique. A bookmarklet is a bookmark containing JS commands stored in the browser's "favorites" bar, which adds "one-click" functionalities to the currently displayed web page. This technique has been effectively used to embed or extend programming environments in existing webpages (e.g., [20]). The current approach uses the same mechanism to inject JS code into the Desk programming environment, which can dynamically orchestrate the loading, execution, and termination of existing robot tasks (i.e., "layer 1" BOWs) created in Desk. These bookmarklets are dynamically generated in Blockly, as shown in figure 1. To enable this we extended the Blockly code editor by Franka Desk-specific tasks and facilitated the generation of bookmarklets (see "Generate V.3" button in figure 1) appearing as hyperlinks ("V.2") next to the code generation button. To use the blocks from the "Franka" tab, the user needs to drag and drop them to the workspace and name them according to an existing task created in the Desk environment. Upon dragging and dropping the generated hyperlink onto the web browser's bookmarks bar, the bookmarklet containing the program code is created.

With generated programs now residing on the browser's bookmarks bar, the user can switch to the Desk browser tab and inject them into the Desk environment as shown in figure 2. When clicking on a bookmarked program (i.e. V.2 in figure 2), the JS code generated in Blockly is injected into the Desk environment and executed. Figure 2 also shows a trimmed implementation of the *executeTask* method implementing a simple state machine, which orchestrates the loading of the Panda task specified by the *task* parameter and its execution in JavaScript. This is achieved using the "macro" programming style by triggering a click event on the list element containing the specified task name (e.g. "Pick bolt"), waiting for it to load, and triggering a click event on the "play" button in the Desk environment. The state machine then waits for the task to finish—an event signaled by the return of the "play" button, which is replaced by a "stop" button (not shown in figure 2) during execution—and returns. This technique builds upon so-called asynchronous JavaScript functions marked with the *await* keyword in figure 1. Using this asynchronous calling system is mandatory because a synchronous implementation of the *executeTask* method would block the entire Desk environment, considering that JavaScript is single-threaded. The execution of the program can also be traced and controlled using the browser's JavaScript debugger (e.g. the Chrome developer tools). While this way of orchestrating task execution within the Desk environment may seem somewhat unorthodox, it should be noted that macro-style manipulation of web-based interfaces is a reliably common practice especially for automated web application testing. The technique also enables browser extensions (e.g., Chrome extensions) used to implement add blockers, web content scrapers, and tools for improving the look and functionality of existing websites. A productive version of the Blockly-Desk programming model and environment could thus also be implemented as a browser extension.

Evaluation—The Blockly-Desk system was evaluated on the basis of a real HRC assembly application developed by students without practical programming experience. Two of them had taken a programming course two years earlier and acted as "layer 2" programmers. The other two did not have any programming training and experience at all and were thus considered "layer 1" programmers. One of the authors played the role of a "layer 3" programmer and extended Blockly as needed (e.g., to support keyboard event handling). All participants received a safety training before starting to work with the robot. The "pair programming" principle was followed in teams of two, during which one person would watch the activities of the other one in order to prevent any safety-related issues as well as to observe and, eventually, propose alternative solutions and technical improvements. The developer-observer roles were often switched so that both team members could experience the development process in the same way. This contributed to a better understanding of the problem at hand and helped participants to learn from each other.



```
function executeTask(task){
  return new Promise(resolve => {
    task = task.trim();
    cb = setInterval(function(){
      if(state == "idle"){
        state = "init";
      $("span:contains('"+task+"')").trigger("click");
      }else if($("div.fi-play").length==1 && state=="init"){
        state = "running";
      $("div.fi-play").trigger("click");
      }else if($("div.fi-play").length==1 && state=="running"){
        state = "idle"; clearInterval(cb); resolve(20);
      }},1000);});
}
```

Pick bolt	
Screw bolt	
SHARED TASK: Holzprototyp	
SHARED TASK: Test auf druck	

Fig. 2. Injecting and running custom programs in the Desk environment.

Figure 3 illustrates the application used to mount four semiconductors on an aluminum heat dissipator. While the Panda robot picks and places the dissipator into a specially designed assembly device, the human inserts the semiconductors and the screws into a stencil to facilitate faster screwing. Then the Panda robot retracts the finished part from the assembly device and stores it. In the process, the worker can choose to do the screwing herself or let the cobot carry out this task while arranging the dissipators and semiconductors. To let the robot do the screwing, the human pushes a button. An additional button is used to signal the robot to store the finished part. The combination of Blockly and Desk proved very useful during development because it enabled the programmers to partition the tasks more effectively so as to optimize assembly time, ensure safety, and to implement the "shared tasks" principle [21] by providing the worker with more options in a simple yet effective way. Because the robot tasks could be partitioned freely by the layer 2 programmer in Blockly, testing and precise position adjustments could be carried out more effectively since the participants could work on short chains of robot movements in Desk and orchestrate them using Blockly (e.g. conditional flows). The pair programming principle also facilitated a learning process, during which participants became skilled in layer 2 programming or improved their existing programming skills. In addition, all participants learned how to use the cobot's multimodal teach-in features more effectively.

During this evaluation experiment, some limitations of the Desk environment also became evident. For example, Desk does not allow conditional program flows (e.g. if-else) and loops (e.g. while) needed to implementing the shared tasks principle). Another issue with Desk is that it does not display the current coordinate of the tool center point (TCP). This makes it difficult for users to reposition the end effector in the exact same location or to respect the layout of predefined assembly stations during teach-in. To overcome this, it was necessary to implement a service based on top of the robot's C++ API, which may be regarded as advanced functionality within the 3-layer programming model.

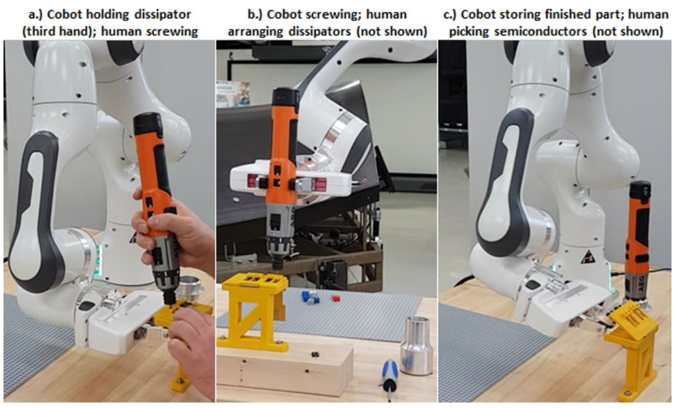

a.) Cobot holding dissipator (third hand); human screwing
b.) Cobot screwing; human arranging dissipators (not shown)
c.) Cobot storing finished part; human picking semiconductors (not shown)

Fig. 3. Example assembly applications with shared screwing task.

4. Discussion and Outlook

Democratizing cobot technology starts with democratizing robot programming. In this sense, we proposed a participatory 3-layer programming model aimed at leveraging the skills and knowledge of non-experts while requiring a minimum of additional training. One of the innovative aspects of the proposed 3-layer model is that, unlike other robot programming models (e.g. [11,14]), it allows non-expert programmers to develop what currently robotics experts refer to as "robot skills." In this context we argued that, while a limited number of robot skills like screwing or drilling may be offered off-the-shelf for a limited number of use cases (e.g., 10-20 types of screws and bits), automated handling functions truly gain meaning in the context of very specific manufacturing applications and thus cannot capture all aspects of those applications *a priori*. Our preliminary evaluation of the model showed that it enables people without programming skills to develop cobot applications in an effective way. The proposed approach to the democratization of cobot technology will be evaluated in depth as part of a research project called "Cobot Meets Makerspace" [25] co-funded by the Austrian Research Promotion Agency (FFG).

By emphasizing the active roles of workers and industrial engineers in the development of HRC applications, the proposed 3-layer model is not meant to trade autonomy for simplicity but to achieve *autonomy through simplicity and cooperation* between all actors involved in and knowledgeable of the workings of the shop-floor. On the long term, a clear definition of the interfaces for cooperation between workers, industrial engineers, robotics experts, and cobots themselves will enable the creation of autonomous robotic software, in which the knowledge of both robotics experts and regular factory employees can be effectively represented as flexibly combinable, reusable automated handling functions and basic assembly workflows. This will enable the software to accomplish a wide range of handing tasks, currently reserved for humans, without needing to be reprogrammed. The role of experts, workers, and industrial engineers will be to create and extend a library of reusable handling tasks and assembly workflows aimed at implementing an increasing number of assembly operations.

The proposed 3-layer model also has implications for current norms and practices concerning the safety of and responsibility for HRC applications in different contexts. In European factories, each HRC application must be certified individually by a certifying body (e.g. TÜV Austria). This means that each application will be thoroughly scrutinized in consideration of all the relevant standards before workers will interact with it. The responsibility for interactions with cobots during the development of applications must be shared between the workers and engineers involved. Only selected workers, who will be required to take a full safety training, will be allowed to develop cobot applications. In makerspaces, the responsibility is shared between customers (i.e., persons interested in working with cobots) and the specially trained personnel of the makerspace, who are mentoring the "makers" and supervising their activities. The customers are required to take a safety training and assume responsibility for applications by signing an agreement with the makerspace. In addition, active safety systems are in place to monitor the safety zones of the robot (cooperation, interaction, co-location, etc.) so as to ensure that only one person—the one who also assumes responsibility—can work with the robot undisturbed by others.

In sum, the goal of the proposed approach is to advance intuitive robot programming while preserving the safety of human-robot collaboration in manufacturing. Its aim is to bring workers, industrial engineers closer together when developing applications in order for them to learn from each other. Our experience shows that off-the-shelf robotic skills (like screwing or drilling) are not universally applicable and need to be adapted for the application at hand on the shop-floor. This usually takes a long time to achieve production quality. It is here that a closer collaboration between workers, industrial engineers, and robotics experts is required to reduce application development time by also representing the long-term knowledge of shop-floor employees in cobot software.

Acknowledgements

This research was co-funded by the Austrian Research Promotion Agency (FFG) as part of the *HCCPPAS* grant project.

References

[1] Menges, B., Sarrey, M., & Henaff, P. (2018, August). Integration of a collaborative robot in a hard steel industrial environment. In 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE) (pp. 634-637). IEEE.

[2] Unger, H., Markert, T., & Müller, E. (2018). Evaluation of use cases of autonomous mobile robots in factory environments. Procedia Manufacturing, 17, 254-261.

[3] Ranz, F., Komenda, T., Reisinger, G., Hold, P., Hummel, V., & Sihn, W. (2018). A Morphology of Human Robot Collaboration Systems for Industrial Assembly. Procedia CIRP, 72(1), 99-104.

[4] Tanenbaum, J. G., Williams, A. M., Desjardins, A., & Tanenbaum, K. (2013). Democratizing technology: pleasure, utility and expressiveness in DIY and maker practice. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 2603-2612). ACM.

[5] Franka Emika Desk: https://www.franka.de/apps/

[6] Fraser, N. (2015). Ten things we've learned from Blockly. In Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE (pp. 49-50). IEEE.

[7] RoboDK: Simulator for industrial robots and offline programming: https://robodk.com

[8] ABB RobotStudio: https://new.abb.com/products/robotics/robotstudio

[9] Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. Educational Researcher, 42(1), 38-43.

[10] Lindtner, S., Hertz, G. D., & Dourish, P. (2014, April). Emerging sites of HCI innovation: hackerspaces, hardware startups & incubators. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 439-448). ACM.

[11] Steinmetz, F.,Wollschläger, A., Weitschat, R. (2018). RAZER-A HRI for Visual Task-Level Programming and Intuitive Skill Parameterization. IEEE Robotics and Automation Letters, 3(3), 1362-1369.

[12] Drag & Bot: https://www.dragandbot.com/

[13] Rethink Robotics Intera: https://www.rethinkrobotics.com/intera/

[14] Hangl, S., Mennel, A., & Piater, J. (2017). A novel skill-based programming paradigm based on autonomous playing and skill-centric testing. arXiv preprint arXiv:1709.06049.

[15] VDI-Richtlinie 2860 (2010): Montage-und Handhabungstechnik; Handhabungsfunktionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbole, Berlin Düsseldorf.

[16] Bainbridge, L. (1983). Ironies of automation. In Analysis, Design and Evaluation of Man–Machine Systems 1982 (pp. 129-135).

[17] Tsoukas, H. 2003. Do we really understand tacit knowledge? M. Easterby-Smith, M. Lyles, eds. The Blackwell Handbook of Organizational Learning and Knowledge Management. Blackwell, Oxford, UK, 410–427.

[18] Ionescu, T. B., Merz, M. (2018). Cyber-physical Production: Models and Enactment of the Smart Factory. Arbeits- und Industriesoziologische Studien, 11(2), 247-261.

[19] Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. IEEE software, 17(4), 19-25.

[20] Zhang, X., & Guo, P. J. (2017). DS. js: Turn Any Webpage into an Example-Centric Live Programming Environment for Learning Data Science. In Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (pp. 691-702). ACM.

[21] Ansari, F., Erol, S., & Sihn, W. (2018). Rethinking Human-Machine Learning in Industry 4.0: How Does the Paradigm Shift Treat the Role of Human Learning?. Procedia Manufacturing, 23, 117-122.

[22] Huang, J., & Cakmak, M. (2017). Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts. In Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction (pp. 453-462). ACM.

[23] Ionescu, T.B. (2019). When Software Development Meets the Shop-floor: The Case of Industrial Fablabs. In Proceedings of the 40th ACM/IEEE International Conference on Software Engineering. IEEE.

[24] Ionescu, T.B. (2019). Developing Software for the Shop-floor on the Shop-floor: An Ethnographic Study of Software Engineering Practices in and for the Smart Factory. In Proceeding of IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE). IEEE.

[25] "Cobot Meets Makerspace" project website: http://www.comemak.at