A Graphical Environment to Support the Development of Affordable Digital Manufacturing Solutions

Zhengyang Ling, Lavindra de Silva, Greg Hawkridge, Duncan McFarlane, Giovanna Martínez-Arellano, Benjamin Schönfuß, Alan Thorne

Abstract Digital solutions have the potential to drastically transform manufacturing operations, but smaller manufacturing businesses (SMEs) have been reluctant to adopt digital solutions due to perceived investment and upskilling costs. The Digital Manufacturing on a Shoestring project was thus established to facilitate the process of digital solution adoption in manufacturing SMEs. To this end, a solution development approach was proposed including a graphical environment to support the design of affordable digital solutions. This paper discusses the concepts and methods underlying this graphical design environment, including its early implementation. A preliminary evaluation is also presented involving industrial user studies with SMEs.

1 Introduction

The use of digital technologies and solutions in manufacturing businesses has the potential to drastically transform manufacturing operations, with the key benefits including cost reductions and faster delivery of goods and services. However, a number of recent findings and our own investigations have revealed that the smaller manufacturing businesses (SMEs), which have limited technical know-how compared to larger organisations, are reluctant to adopt digital solutions due to perceived investment and upskilling costs [17, 19, 11]. The Digital Manufacturing on a Shoestring project [14] was thus established and aimed toward facilitating the process of adoption of digital solutions in manufacturing SME operations.

To achieve this, two key strategies were proposed in the project. The first was to focus only on affordable, readily available, off-the-shelf technologies, including open source software libraries and affordable computers such as Raspberry Pis, and the second was to create a simple 'digital solution development approach' that ordinary manufacturing SMEs can use to put together digital technologies to form coherent and useful digital solutions. The resulting development approach [8] advocates top-down development, starting from the business needs of a company (identified through in-company 'requirements workshops' [18]) and an identification and specification of the next top-priority digital solution for those needs, followed by a hierarchical and template-based development of the solution down to the level of choosing individual technologies, with the option to choose existing, ready-made elements in place of templates at any point in the process (e.g. an off-the-shelf, complete digital solution that matches the solution specification).

A core part of the solution development approach is a graphical environment to support the *design* of a digital manufacturing solution, either by loading and configuring an existing solution provided by a community of users, or at the other extreme, putting together digital technology 'building blocks' from scratch, with the option to also then automatically *generate* the relevant connections between the chosen software building blocks, as well as the necessary guidelines for solution deployment (e.g. hardware connections). This paper discusses the concepts and methods underlying the solution design aspect of the graphical support environment, and its early implementation as a web-based online '*Solution Configurator*' to assist SMEs with the adoption of digital manufacturing solutions.

Z. Ling, L. de Silva, G. Hawkridge, D. McFarlane, B. Schönfuß, A. Thorne
Institute for Manufacturing, University of Cambridge, UK, e-mail: zl461@cam.ac.uk

G. Martínez-Arellano
Institute for Advanced Manufacturing, University of Nottingham, UK

This work was supported by the EPSRC [grant number EP/R032777/1].

This paper is organised as follows. Sec. 2 identifies the requirements for a graphical support environment for manufacturing SMEs, and Sec. 3 addresses the design oriented requirements through a conceptual design environment. Sec. 4 discusses the implementation of the concept, as well as some preliminary results from qualitative experiments. Finally, Sec. 5 compares closely related work, and the conclusions and future directions are discussed in Sec 6.

2 Requirements for a Graphical Support Environment

A number of particular requirements emerged for a graphical development environment that suited manufacturing SMEs, which were first identified through discussions within the research team and then revised through a series of meetings with project partners, e.g. UK manufacturing SMEs and service providers for such companies. The main requirements originally set for this environment are listed below, split up into *high-level* requirements (H1-H5) involving the entire development environment, and *detailed* requirements (D1-D4) involving more specific aspects.

H1 The environment must support the development of digital manufacturing solutions from a set of *modular* 'building blocks' (BBs), abstractly representing *both* hardware and software, e.g. sensors and databases.

H2 BBs and their connections must look *simple and intuitive*, e.g. like Lego or jigsaw pieces, which contrasts with design elements used in traditional component-based development environments such as Netbeans [5] and Archimate [12].

H3 The environment must be *configuration (or 'composition') oriented*, rather than a coding environment; e.g., BBs and their connections should represent interactions between subsystems, rather than between snippets of code (the latter approach is used, e.g., in Scratch [16] and MakeCode [3]).

H4 The environment must *guide the user* during the design process, e.g., helping identify optimal BBs (technologies) for a solution, and flagging potential issues when BBs are put together in undesirable or potentially problematic ways (e.g. a Raspberry Pi is connected to an incompatible camera).

H5 The environment should ideally *generate* guidelines for how to manually integrate the hardware and software depicted in the design, as well as code for BB communication and orchestration; this contrasts with development environments that generate function skeletons to be filled in by a programmer (e.g. Eclipse [20]) and code for machining a designed part (e.g. AutoCAD).

D1 BBs must have *levels of abstraction*, enabling the design to start from a top-level solution need [18] and end at a level that suits the end-user. For example, the user may want to only design at the level of 'service modules' [8] (e.g. 'data collection' and 'data storage'), without designing or even viewing the BBs they encapsulate (e.g. a sensor, software library, and database).

D2 Related to the above need is the ability to *hide* (or show) the constituents of any BB, allowing the user to focus on only the relevant detail.

D3 It should be possible to *dynamically adjust* the number of connectors in a BB, e.g. to enable any number of widget BBs to be part of a data visualisation BB.

D4 BBs should allow suitable *connection types*, e.g. 'side-by-side' and 'nested' (in contrast to, e.g., Node-RED [1]), to represent relationships between software and hardware, analogously to relationships identified in Software Engineering, e.g. association and composition.

These requirements have provided an ongoing reference point for the developments described in the following sections.

3 Conceptual Development of the Graphical Environment

Requirements H1-5 and D1-4 outlined in Sec. 2 were addressed first in the form of a conceptual design environment, driven by other aspects of the overall Shoestring solution development approach [14, 8], which advocates top-down solution development, starting from the business needs of a company and the identification and specification of the next top-priority digital solution for those needs, followed by a hierarchical, template-based development of the solution. This can involve firstly choosing (guided by the specification) a suitable 'solution template', which comprises a collection of 'service module' types, and then incrementally filling in the

template through the selection of a service module template for each type, until either the desired level of detail is reached or a full solution instance has been designed in terms of individual BB instances (e.g. technologies).

Following [14, 8], the conceptual design environment therefore distinguishes between four main types of elements: a Solution, Service Module (SM), Basic BB (or simply BB), and BB Component, with each of them being a higher level of abstraction than the next. There are 6 types of SM [8], e.g. 'Data Collection', 'State and Data Storage', and 'User Interface', and many types of BB, with the mandatory ones being 'Computing Device' (representing a computer that hosts one or more SMs) and 'Service Wrapper' (representing the API used by an SM to communicate with other SMs), and other common BB types including 'Database', 'Sensor', 'Camera', 'Microphone', 'Interface Adaptor', and 'Visualisation Platform'. BBs themselves can also have a few levels of abstraction, e.g. two 'Visualisation Dashboard' BBs within a 'Visualisation Platform' BB. Examples of BB Components (types) include a (dashboard) 'Panel' and (database) 'Column'.

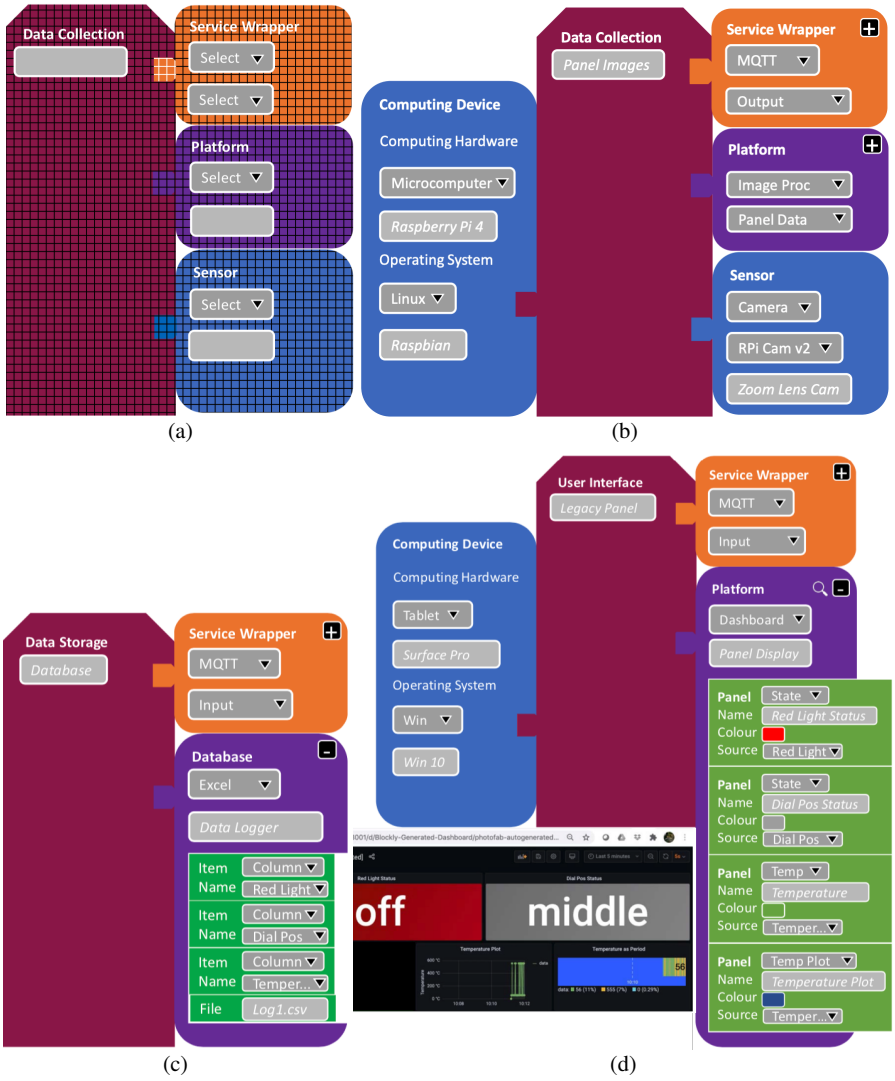


Fig. 1: A Data Collection SM template (a); its instantiation (b); a Data Storage SM (c); and a User Interface SM (d), together with a possible data visualisation for the dashboard (bottom center).

Solution ::= **Category**, [**Name**], {**SM**}+	Job Tracking; // **Solution**
SM ::= **SMType**, [**Name**], **SW**, **CompuDev**, {**BB**}+	• Data Storage; // **SM**
SW ::= **Connectivity**, ("Input" | "Output"), {**DataStream**}	• Name:Database; • • MQTT; // **SW**
BB ::= **BBType**, {**BBSubtype**}, [**Name**], **Detail**	• • Input;
Detail ::= {**BB**} | {**Comp**}	• • Red Light;
CompuDev ::= {**BBSubtype**}, [**Name**], **OS**, [**Name**]	• • Dial Pos;
Comp ::= **CompType**, {**CompSubType**}, [**Name**], {**Attribute**, **Value**}	• • Temp;
Category ::= "Fault Monitoring" | "Job Tracking" | · · ·	• • Microcomputer; // **CompuDev**
SMType ::= "Data Storage" | "User Interface" | · · ·	• • RPi3;
BBType ::= "Sensor" | "Visualisation Platform" | · · ·	• • Linux;
BBSubtype ::= "Camera" | "Dashboard" | · · ·	• • Raspbian;
CompType ::= "Item" | "Widget" | · · ·	
CompSubType ::= "State" | "Plot" | · · ·	• • Database; // **BB**
Connectivity ::= "MQTT" | "OPC-UA" | · · ·	• • Excel;
OS ::= "Windows" | "Linux" | · · ·	• • Data Logger;
Name ::= *unique string*	• • • Item; Col; Red Light; // **Comp**
Attribute ::= *unique string*	• • • Item; Col; Dial Pos;
Value ::= *unique string*	• • • Item; Col; Temp;
DataStream ::= *unique string*	• • • File; Log1.csv

Fig. 2: EBNF for a Solution in terms of SMs, BBs, and BB Components (left), and an SM instance generated by the EBNF (right). Indentation (•), spacing, and comments (//) are for readability, and a semicolon indicates concatenation. Abbreviations used are: SW=Service Wrapper, CompuDev=Computing Device, Comp=BB Component, OS=Operating System, and Col=Column.

3.1 Conceptual solution design examples

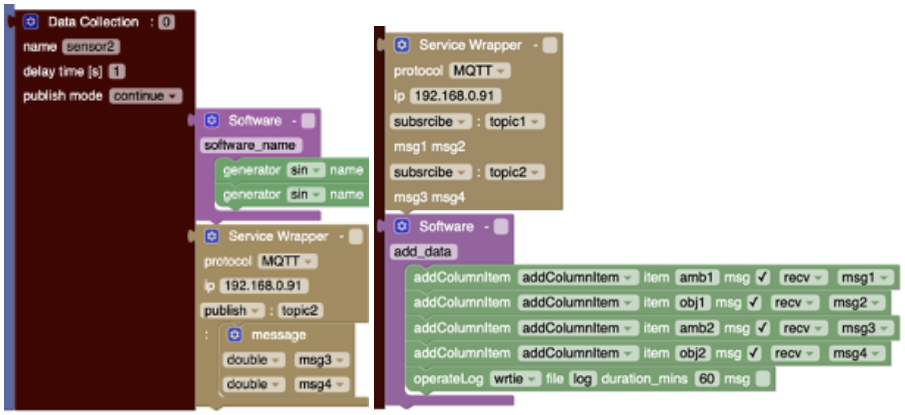
An example of a 'Data Collection' SM template is shown in Fig. 1(a), with its instantiation in Fig. 1(b). (Some computing devices are not shown in the figure due to space.) We use a jigsaw-like design for elements, with colour coding to distinguish between them. Drop-down menus enable selecting subtypes or instances, e.g. the 'Microcomputer' subtype for 'Computing Device' and the 'RPi Cam v2' technology (instance) for 'Sensor'. Text fields allow users to optionally give their own identifiers (shown in italics) to BBs. The Service Wrapper indicates that MQTT will be used as the connectivity technology [9] for communication, and that the SM will be publishing (rather than subscribing to) data.[1] By default, all the data that is sensed by the camera and processed by the image processing platform will be published, with the option to customise this by clicking on the '+' symbol in the Service Wrapper BB to view the hidden detail, and then removing data elements. Communication between SMs can be depicted with arrows between SMs (not shown in the figure).

There are also other ways to design the above. For example, the SM in Fig. 1(b) could be arranged so that the Computing Device is on the right side of the red BB, with the other BBs connected to the Computing Device. That would be more of a 'service-centric' view as opposed to the 'device-centric' view shown in the figure. Secondly, the operating system could appear as a nested BB within the Computing Device, rather than as a drop-down menu option.

An example of the use of BB Components is shown in the 'Data Storage' SM in Fig. 1(c), which are stacked within the Database BB rather than connected to its side. The Service Wrapper by default subscribes to all the data streams published by other SMs already designed, and the Database BB, with 'Excel' as the instance, stores each stream as an 'Item' BB Component, which corresponds to an Excel column. To facilitate this design, the environment automatically selects and instantiates a default template comprising a BB Component per data stream in the Service Wrapper.

Another example of the use of BB Components is shown in the 'User Interface' SM in Fig. 1(d). The Service Wrapper by default subscribes to all data streams as before, and the 'Visualisation Platform' BB, with 'Dashboard' as the subtype, displays each stream as a 'Panel' BB Component. (This is a simplification of the more flexible design approach of having separate Dashboard BBs connected to the Platform BB.) As before, the environment automatically selects and instantiates a default template comprising the 4 panels shown in the figure, which can be manually customised if necessary, e.g. to change a panel colour or type, or to add or remove a panel. Alternatively, a different dashboard or template could be chosen by clicking on the 'search' icon in the top right corner of the Platform BB in Fig. 1(d). The resulting data visualisation might look as shown in Fig. 1(d), produced using the Grafana visualisation platform as discussed in [13].

[1] Provision for more sophisticated communication mechanisms is left to future work, e.g. where the request-response protocol is also allowed.



(a) (b)

Fig. 3: Bits of the Data Collection and Data Storage SMs from the configurator.

3.2 Guiding the user through the solution design process

Since the templates available for a Solution, SM, or BB may be limited, the user may choose for example to select and adapt a predefined template, or build a solution from scratch by putting together BBs and BB Components. There is therefore the need for a mechanism that can guide the user (with minimal delay) during the solution design process (requirement H4 in Sec. 2). We propose the use of an approach where the structure of Solutions (including SMs, BBs, and Components) is defined using a grammar which is then used to check the solution as it is being designed. Fig. 2 shows a high-level grammar for a Solution in the ISO/IEC EBNF [2] format, as well as a valid string generated by the grammar, which is a textual version of the graphical design shown in Fig. 1(c). The grammar requires further restrictions to preclude more nuanced connections that are invalid, e.g. a (database) 'Item' BB component being mistakenly added into a 'Dashboard' BB (which should only include 'Panel' BB Components), but we leave the extended grammar to future work.[2] Nonetheless, even the grammar shown, once it has been normalised (i.e., converted into Chomsky Normal Form [6]) can be used to provide the user with useful (automated) guidance while a solution is being designed. For example, an attempt to replace a BB Component in the Dashboard BB in Fig. 1(d) with a (Basic) BB can be detected and flagged as being invalid, as indicated by the non-terminal symbol 'Detail' in Fig. 2 (left).

Detecting such inconsistencies requires two steps: (i) converting the graphical solution into a string as shown in Fig. 2 (right), and (ii) checking whether the string can be generated by the (normalised) grammar. The first step only requires one traversal of each element (SM, BB, and BB Component) in the current 'snapshot' of the solution being designed, which can be done in polynomial time relative to the number of elements in the solution; the second step can also be done in polynomial time relative to the length of the resulting string [21] (and thus the number of elements in the current solution). The overall polynomial-time complexity of a user guidance algorithm is vital to minimise the 'lag' that the user may experience during design (i.e., waiting for the algorithm to check whether a change done to the design is valid).

4 Initial Implementation of the Graphical Support Environment

The implementation of the Solution Configurator was based on the conceptual design environment discussed in Sec. 3, and used the Blockly [15] library, which facilitates creating block-based visual programming languages using client-side Javascript. Fragments of SMs put together using the configurator are shown in Fig. 3(a)-(b), which correspond to the conceptual examples in Fig. 1(b)-(c), respectively. The implementation is still a prototype, with the following main differences compared to the concept: (i) a Basic BB cannot yet be connected to another *Basic* BB (e.g. a Dashboard (Basic) BB connected to the Platform (Basic) BB); (ii) some terms that describe BB types are yet to be made consistent (e.g. use of the term 'Platform' in the concept versus 'Software' in the implementation); and (iii) compared to the concept, there is more detail in some SMs/BBs/Components in

[2] Techniques from the area of Behaviour Composition, e.g. [7], might prove useful to this end.

the implementation, allowing more customisation (e.g. the ability to subscribe to specific 'topics') but perhaps at the expense of user friendliness – the right amount of detail to show is still being investigated. There were also limitations in the Blockly library itself, which dictated certain modeling choices: e.g. it was not possible to (i) model a BB that has both a nested BB(s) within it as well as a BB(s) connected to its side, and (ii) draw arrows to connect SMs, in order to illustrate how they communicate. These limitations posed difficulties in conforming to requirement D4 in Sec. 2.

In the rest of this section we present some preliminary results from evaluations conducted using both the conceptual design environment discussed in Sec. 3 and its current implementation (the Solution Configurator).

4.1 Feedback on the conceptual development environment

The first evaluation was carried out through an (online) meeting with 8 participants in total who had varied roles within their organisations; from the group, 6 participants were from UK manufacturing SMEs and 2 were from solution/technology providers for such companies. The participants were first given an overview of the conceptual design environment as described in Sec. 3, explaining the top-down, template-based approach, and the option to both design and customise digital solutions at the level of SMs, BBs and to the more detailed level of (BB) Components. The participants were then asked the following question:

Q1 '*Do you have an "IT savvy" person in your organisation who might be comfortable with using such a solution configurator (once it is finalised) in order to design and then deploy digital solutions?*'. The possible answers were: (**A**) '*Yes*'; (**B**) '*Maybe - we'll need to check*'; (**C**) '*No - but we might be able to, after looking at documentation/tutorials*'; (**D**) '*No - but we might be able to, if someone guides us*'; and (**E**) '*No - but we might hire someone to deploy solutions via the configurator*'.

More feedback was then requested after each of the following three demonstrations: (i) showing a digital solution being put together only by selecting *predefined* SM instances, i.e., the ones shown in Fig. 1(b)-(c), but with 'CouchDB' instead of 'Excel', and without the use of BB Components; (ii) showing how those two SM instances could be designed by starting from SM *templates* (e.g. the one shown in Fig. 1(a)) and manually filling in the information; and (iii) showing an example of the customisation of *BB Components*, which involved the following steps: taking the 'Data Storage' SM instance built as part of the previous demonstration, changing 'CouchDB' to 'Excel', changing the filename, and finally, introducing the 'User Interface' SM instance shown in Fig. 1 and changing some of the panel colours. After the first demonstration, feedback was requested as follows.

Q2 Participants were shown the following options: (**A**) '*The design process seemed easy - I could do it*'; (**B**) '*The design process seemed easy - someone in my company could do it*'; (**C**) '*The design process didn't seem that easy*', followed by some specific reasons to choose from; and (**D**) '*Other feedback*'.

The same feedback options were shown after the second and third demonstrations but with the specific reasons within option C adapted to suit the demonstration. The feedback received is summarised in Table 1 below. The key insights were that 75% (of those who answered) said they had an IT savvy person in their organisation who might be comfortable with using the configurator, and 25% said they would need to check. All those who answered felt that the high-level design approach – building a solution by simply putting together entire SM instances – was easy or that someone in their company could do it. Finally, 90% felt the approach that involved instantiating or customising BBs, or even Components, was easy or that someone in their company could do it.

Q	**A**	**B**	**C**	**D**	**E**
1	6	2	0	0	0
2	3	5	0	-	-
3	2	3	0	-	-
4	2	2	1	-	-

Table 1: The first column shows the question numbers, with Q3 and Q4 being the same as Q2 except for the specific reasons within answer C. Columns A-E correspond to the answers, and the numbers indicate how many participants gave those answers.

While these preliminary results look promising, the group of participants was relatively small, which we intend to improve on in the future. In the next section, we discuss some preliminary user studies for the option of designing solutions from scratch – without using predefined SMs, through company trials using the Solution Configurator (as opposed to the conceptual design environment).

4.2 Lessons from industrial trials

The preliminary evaluation of the Solution Configurator involved onsite trials at 3 UK manufacturing SMEs; one of them was carried out (at SME1) by a member of the research team, and the other two were carried out (at SME2 and SME3) by one 'IT savvy' member of staff from each SME. For the latter trials, three key pieces of information were given to the users on how to design SMs for their desired solutions:[3] (i) following the approach in [8], a high-level diagram of the SMs and BBs needed for the solution and how they should be connected; (ii) the basic functionalities of the configurator, e.g. where to find BBs, how to connect them, and how to adjust the number of connectors in a BB; and (iii) screenshots showing how each completed SM should look – the development of each SM was treated as a separate 'exercise'. In future work we intend to explore to what extent users can develop SMs without screenshots showing the *exact* SM needed.

The 3 digital solutions fell under 2 solution categories [18]: SME1 needed a 'Process Monitoring' solution, which used a Raspberry Pi, a low-cost camera, and machine vision to read legacy display panels in a factory (e.g. dials, lights, and LCD displays), and also stored and visualised the resulting data; SME2 needed a 'Fault Monitoring' solution, which used a similar approach to monitor a material braiding machine and alert operators when the braiding angle was outside of a given tolerance; and SME3 also needed a 'Process Monitoring' solution, which used low-cost temperature sensors together with a Raspberry Pi to capture, store and visualise temperature data as hot wax was gradually cooled along a conveyor belt. Fig. 3 shows screenshots (cropped due to lack of space) of SMs required in some of the exercises.

Initial results from the trials were promising, with the following main insights: (i) an 'IT savvy' user in the context of SME2 and SME3 meant someone with an Engineering degree; (ii) these users took under 40 minutes per exercise on average, including the time taken to read the guidelines; and (iii) after setting up the Raspberry Pi and camera at SME1, the researcher took under an hour on average (from three visits) to design as well as deploy a bespoke solution (comprising a Data Collection, Data Storage, and User Interface SM and their constituent BBs and Components) from scratch. Despite the researcher being an expert user of the configurator, we anticipate that it would have taken him much longer to program the 3 SMs and their communication directly in Python.

5 Related Work

There are a number of graphical development environments that are similar to the Solution Configurator; we discuss the most closely related approaches in this section.

In software and systems engineering, the Object Management Group (OMG) provides various languages for modeling software and hardware entities and their relationships, such as block diagrams, object diagrams, activity diagrams, deployment diagrams, component diagrams, and sequence diagrams [4]. Of particular relevance is SysML, which allows modelling systems of systems by extending existing OMG languages. Similarly, Archimate [12], hosted by the Open Group, offers comparable modeling capabilities with a focus on the high-level modeling of entire enterprises, both within and across business domains. While some of these languages are expressive enough to address many of the requirements in Sec. 2 (particularly requirement D4, which could not be entirely addressed in our implementation), these languages are still meant for skilled users (e.g. engineers or computer scientists) and tend not to be sufficiently simple and intuitive for staff at manufacturing SMEs.

There are various block-based graphical development environments that are closely related to our work, in particular MIT's Scratch [16] and Microsoft's MakeCode [3]. The main difference with our work is that these are high-level *programming* environments, rather than environments for modeling the *composition* of a cyber-physical system. Furthermore, there are some modeling details that are harder to realise with the mentioned environments, e.g. it is not possible to connect blocks side-by-side (requirement D4 in Sec. 2) as shown in Fig. 1. Another closely related graphical development environment is Node-RED [1], a high-level flow-based programming language. Node-RED makes it easy to 'wire' together various IoT devices, by using predefined 'nodes' provided by a community of users and/or by developing new nodes, where a node can represent various entities including a Javascript function, a programming language construct such as a switch statement, a message

[3] Two of the desired solutions were determined through in-company requirements workshops [18], and the third through discussions with a company director.

containing data, or a connectivity technology such as MQTT. While this makes Node-RED similar in principle to the environment presented in this paper, there are two key differences. The first is that Node-RED is still programming-oriented, as suggested by the the core nodes which include capabilities for writing functions and switch statements, for error handling, for splitting and combining messages passed between nodes, etc. The second key difference is that nodes can only be 'wired' together (through the same type of connector) but not grouped in more intuitive ways, e.g. nested as shown in Fig. 1. Nonetheless, Node-RED could complement our tool, as it has a capability for modeling the flow of data between SMs.

Integrated Development Environments such as Eclipse [20], Netbeans [5], and Mendix [10] are also related to the Solution Configurator, which facilitate building individual software applications. Like the Solution Configurator, such tools offer an interface for dragging and dropping components – albeit GUI components such as buttons, progress bars, and charts – onto a workspace, and then generating the resulting solution or solution skeleton. However, Netbeans and Eclipse still require a software developer to fill in the corresponding function skeletons with code, and Mendix also seems to require a certain level of skill when combining multiple applications together, e.g. pulling data from an external database to display the data in a visualisation application being developed, which would require an understanding of data models and flows represented similarly to OMG's class diagrams and Node-RED's flow diagrams. However, Mendix could, for example, correspond to a Platform BB in our framework, for visualising data as shown in Fig. 1(d).

6 Conclusions and Future Work

This paper has discussed a graphical support environment that facilitates the development of affordable digital manufacturing solutions for SMEs, with a focus on the solution design aspect of the environment. After describing the environment's underlying concepts and the resulting implementation, a preliminary evaluation was presented, both for the conceptual environment and its implementation. Two useful features for such an environment include the ability to generate code (based on the design) that can be deployed on the identified computing devices, and guiding the user through the process of designing a digital solution. We leave a discussion of the former to a longer paper, and we have given some insights into an approach for feasibly verifying whether a solution being designed makes sense.

Possible avenues for future work include using these insights to develop a more thorough account of user guidance and thereby address requirement H4 in Sec. 2, finding a solution to the limitations in Blockly in order to fully address requirement D4 (support for suitable connection types), and finally, coming up with an approach for addressing requirement H5 (guideline/code generation), even if to a limited extent – e.g. generating guidelines only at a high level. We believe that the remaining requirements have been suitably addressed by our approach.

The proposed support environment forms part of the larger digital solution development approach proposed by the Digital Manufacturing on a Shoestring project [8], which accounts for further aspects such as requirements capture, solution maintenance and operator training. Thus, another main aim going forward is to make the environment more closely aligned with the larger development approach, e.g. by including a design element that represents a solution, and starting the design process by identifying the highest priority solution element, followed by a relevant template for it (comprising SM types).

References

1. Node-RED guide. http://noderedguide.com.
2. Information technology - syntactic metalanguage - Extended BNF. http://standards.iso.org/ittf/PubliclyAvailableStandards/, 1996 (accessed March 10, 2017).
3. T. Ball, A. Chatra, P. de Halleux, S. Hodges, M. Moskal, and J. Russell. Microsoft makecode: embedded programming for education, in blocks and typescript. In *Proceedings of the 2019 ACM SIGPLAN Symposium on SPLASH-E*, pages 7–12, 2019.
4. G. Booch. *The unified modeling language user guide*. Pearson Education India, 2005.
5. T. Boudreau, J. Glick, S. Greene, V. Spurlin, and J. J. Woehr. *NetBeans: the definitive guide: developing, debugging, and deploying Java code*. O'Reilly Media, Inc., 2002.
6. N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.
7. G. De Giacomo, F. Patrizi, and S. Sardina. Automatic behavior composition synthesis. *Artificial Intelligence*, 196:106–142, 2013.

8. G. Hawkridge, D. McFarlane, B. Schönfuß, J. Kaiser, L. de Silva, and G. Terrazas. Designing shoestring solutions: An approach for designing low-cost solutions for manufacturing. In *SOHOMA*, page to appear, 2021.
9. G. Hawkridge, M. Perez Hernandez, L. de Silva, G. Terrazas, Y. Tlegenov, D. McFarlane, and A. Thorne. Tying together solutions for digital manufacturing: Assessment of connectivity technologies & approaches. In *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1383–1387, 2019.
10. M. Henkel and J. Stirna. Pondering on the key functionality of model driven development tools: The case of mendix. In *International Conference on Business Informatics Research*, pages 146–160. Springer, 2010.
11. D. Horváth and R. Z. Szabó. Driving forces and barriers of industry 4.0: Do multinational and small and medium-sized companies have equal opportunities? *Technological forecasting and social change*, 146:119–132, 2019.
12. A. Josey, M. Lankhorst, I. Band, H. Jonkers, and D. Quartel. An introduction to the archimate® 3.0 specification. *White Paper from The Open Group*, 2016.
13. G. Martínez-Arellano, M. McNally, J. C. Chaplin, Z. Ling, D. McFarlane, and S. Ratchev. Visualisation on a shoestring: a low cost approach for building visualisation components of industrial digital solutions. In *SOHOMA*, page to appear, 2021.
14. D. McFarlane, S. Ratchev, A. Thorne, A. K. Parlikad, L. De Silva, B. Schönfuß, G. Hawkridge, G. Terrazas, and Y. Tlegenov. Digital manufacturing on a shoestring: Low cost digital solutions for SMEs. In *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*, pages 40–51, 2019.
15. E. Pasternak, R. Fenichel, and A. N. Marshall. Tips for creating a block language with blockly. In *2017 IEEE Blocks and Beyond Workshop (B&B)*, pages 21–24, 2017.
16. M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.
17. B. Schönfuß, D. McFarlane, N. Athanassopoulou, L. Salter, L. de Silva, and S. Ratchev. Prioritising low cost digital solutions required by manufacturing SMEs: A shoestring approach. In *SOHOMA*, pages 290–300, 2019.
18. B. Schönfuß, D. McFarlane, G. Hawkridge, L. Salter, N. Athanassopoulou, and L. de Silva. A catalogue of digital solution areas for prioritising the needs of manufacturing SMEs. *Computers in Industry*, 133(103532), 2021.
19. A. Sevinc, Ş. Gür, and T. Eren. Analysis of the difficulties of smes in industry 4.0 applications by analytical hierarchy process and analytical network process. *Processes*, 6(12):264, 2018.
20. D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.
21. D. H. Younger. Recognition and parsing of context-free languages in time n3. *Information and Control*, 10(2):189–208, 1967.