The Interoperability Challenge: Building a Model-Driven Digital Thread Platform for CPS

Tiziana Margaria^{1,2,3} , Hafiz Ahmad Awais Chaudhary^{1,2} ,
Ivan Guevara^{1,2} , Stephen Ryan^{1,2} , and Alexander Schieweck^{1,2,3}

¹ CSIS, University of Limerick, Limerick, Ireland

tiziana.margaria@ul.ie

² Confirm - Centre for Smart Manufacturing, Limerick, Ireland

³ Lero - The Science Foundation Ireland Research Centre for Software,
Limerick, Ireland

Abstract. With the heterogeneity of the industry 4.0 world, and more generally of the Cyberphysical Systems realm, the quest towards a platform approach to solve the interoperability problem is front and centre to any system and system-of-systems project. Traditional approaches cover individual aspects, like data exchange formats and published interfaces. They may adhere to some standard, however they hardly cover the production of the integration layer, which is implemented as bespoke glue code that is hard to produce and even harder to maintain. Therefore, the traditional integration approach often leads to poor code quality, further increasing the time and cost and reducing the agility, and a high reliance on the individual development skills. We are instead tackling the interoperability challenge by building a model driven/low-code Digital Thread platform that 1) systematizes the integration methodology, 2) provides methods and techniques for the individual integrations based on a layered Domain Specific Languages (DSL) approach, 3) through the DSLs it covers the integration space domain by domain, technology by technology, and is thus highly generalizable and reusable, 4) showcases a first collection of examples from the domains of robotics, IoT, data analytics, AI/ML and web applications, 5) brings cohesiveness to the aforementioned heterogeneous platform, and 6) is easier to understand and maintain, even by not specialized programmers. We showcase the power, versatility and the potential of the Digital Thread platform on four interoperability case studies: the generic extension to REST services, to robotics through the UR family of robots, to the integration of various external databases (for data integration) and to the provision of data analytics capabilities in R.

Keywords: Interoperability · Digital Thread (DT) · Model Driven Development (MDD) · Low Code Development (LCD) · Software platforms.

1 Introduction

The manufacturing industry in the context of Industry 4.0 demands automated and optimized production lines and is moving towards connected and smarter supply chains processes [25, 60]. Cyber Physical Systems (CPSs) are core building blocks of future factories [41] and researchers believe that, with the emergence of systematic industrial integrations of ICTs and external information systems, CPSs will contribute towards “smart anything everywhere” in particular also smart cities and smart factories [18]. This medium-to-large scale industrial integration implies interoperation of interconnected, heterogeneous virtual and physical entities and devices towards a shared goal [5]. Interoperation includes real time data from machines, production lines, IoT devices, networks, programmable logic controllers and external systems into a smarter, connected manufacturing systems [9].

In this context, the integration and interoperability among all these entities is a key challenge for the success of Industry 4.0. Due to architectural convergence, the holistic integration challenge can be organized in three levels [51]:

1. **Physical Integration**, handling the connectivity and communication among devices.
2. **Application Integration**, dealing with the coordination and cooperation among different software applications and data stores.
3. **Business Integration**, covering the collaboration between different functions, processes and stakeholders.

In this context, considering the “reprogrammable factory” vision brought forward within the CPS Hub of the Confirm research centre [32] and the high-level depiction in Fig. 1, we find a broad correspondence between the three layers above and the three layers implicit in the picture. The Digital Twin is there a “sosa” of any individual component, software or process, and the Digital Thread is a fitting analogy for the role played by any integration and interoperability layer delivering that ability to communicate and cooperate. Ideally, the digital thread should not be provided through a myriad of scripting quick fixes, nor through a vast patchwork of bespoke technologies, that may be adequate serve individual point-to-point interfacing needs, but become a nightmare to understand, test, validate, manage, and evolve.

In Fig. 1, the Digital Thread is the collection of blue lines (solid or dotted), that manage the communication and interoperation between the **Business layer** (at the top), the integration and communication middleware and their platforms (like e.g., EdgeX foundry) as well as the Digital Twins, both at the **Application layer** (in the middle), and the myriad devices, machines, sensors, dashboards, and more at the **Physical layer**, which may also include software for SDNs, SCADA, analytics, AI and ML, and more.

If properly provided and managed, these many heterogeneous vertical and horizontal integrations can enable CPSs to leverage the many advances in industrial systems, big data, AI/ML and cloud computing systems. This way, the

Fig. 1. Confirm HUB CPS – the Reprogrammable factory vision (source: Confirm HUB2)

seamless integration needs advocated by leading technology providers, vendors, and end-users [17] can be fulfilled.

The IEEE defines interoperability as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged” [14]. The author highlighted this need and introduced best practices to develop smarter applications rather than fragmented applications [61]. Literature shows that five categories of interoperability can have quite different arrangements [49]:

1. Device interoperability
2. Network interoperability
3. Syntactical interoperability
4. Semantic interoperability
5. Platform interoperability

The effort of building interoperable systems is an outstanding challenge in the adoption of new technology. The integration layer is too frequently neglected and left for developers to solve as a side issue. This means that a number of experts are required over and over again to reprogram these complex systems in accordance with evolving needs and standards. This is a time-consuming and expensive task [45], and such systems are hard to produce and difficult to maintain. The author [23] also concluded that manual integrations between APIs reduces the agility, and that inaccuracies in the integration may also lead to financial losses and unexpected delays in production. CPSs are typically embedded into a more complex system via interfaces, so modularity (plug and play) and autonomy are important enablers to adapt the upgrades and reconfigurations effectively, in accordance with rapidly changing customer needs [24]. Trustworthy interoperability both at the vertical and horizontal level is critical for setting up Industry 4.0 operations [17].

Model-driven development (MDD) is an approach to develop complex systems using models and model refinement from the conceptual modelling phase to the automated model-to-code transformation of these models to executable code [38]. The main aim [52] of MDD is to produce flexible, cost effective and rapid applications, that are adaptive to enhancements and less complex in terms of maintenance. Achieving this on the basis of direct source code editing is costly, and it systematically excludes the application domain experts, who are the main holders of domain knowledge and carriers of responsibility. At the same time, the cost of quality documentation and training of new human resources for code-based development are other urgent concerns today in companies and organizations that depend on code.

For an adequate, scalable and possibly general and evolvable solution to the interoperability challenge, we propose instead to use adequate, modern software platforms based on model driven development concepts, paying attention to choose those that best support a) high assurance software and systems, b) a fast turnaround time through agility and a DevOps approach, and c) an inclusive understanding of the stakeholders, where few are professional coders. Therefore we adopt a low-code application development paradigm, combined with code generation and service orientation.

The paper is organized as follows: Sect. 2 introduces the digital thread concept and its relation with interoperability. Section 3 discusses the low-code environment platform we use to develop the digital thread platform itself. Section 4 describes the current status of the platform: latest integrations, ideas and enhancements that benefits the bootstrapping of components in the smart manufacturing domain. Section 5 addresses the specific questions posed by the Special track organizers. Finally, Sect. 6 concludes and sketches the planned future work.

2 Digital Thread in the Middle – Interoperability

Digital Twin and Digital Thread are two transformational technological elements in digitalization of the Industry 4.0 [47]. The Digital Twin covers individual aspects of physical assets, i.e., their virtual representation, their environments and the data integrations required for their seamless operations. Digital Twins and AI models are the two kinds of models that the manufacturing Industry has meanwhile accepted as useful. However, they are not the only ones. A Digital Thread connects the data and processes for smarter products, smarter production, and smarter integrated ecosystems. In the modern era, the Digital Thread provides a robust reference architecture to drive innovation, efficiency and traceability of any data, process and communication along the entire system (or system of systems') lifecycle. This is a new, much more structured and organized way to look at integration and interoperability. It is unfamiliar to the manufacturing world, and it is also still unfamiliar to many in some software engineering communities.

For this new paradigm to enter mainstream, systems and their models need to be connected through **an integrated platform** for automatic data and process transformation, analysis, generation and deployment that should be able to take systematic advantage of the formalized knowledge about the many immaterial and material entities involved. Referring to Fig. 1 again, data and operations from and to any of the heterogeneous elements (component, subsystem) in the picture, should be mediated (i.e., adapted, connected, transformed) through the Digital Thread platform, which becomes both the nervous and circulatory system of the overall system:

- The nerves, as whatever is sensed needs to be sent to the decision systems and the commands then relayed to the actuators.
- The circulatory system, as plenty of data is moved in order to “nourish” the information-hungry services that store, aggregate, understand, visualize what happens in the system, increasingly in real time or near-real time.

The choice of which concrete IT system to adopt for this central role is not an easy one, and it is not a choice that can be amended or reversed easily later on. The properties of the Digital Thread will depend very intimately on the characteristics and features of the IT platform on which it bases: whatever the IT platform does not support will be difficult to overlay a posteriori, and whatever is easy in that platform will likely be adopted and become mainstream for the community of users.

Bearing in mind all the desired characteristics, we chose DIME [8] as the IT platform of choice underlying the Digital Thread solution.

3 The Underlying Low-Code Development Environment

DIME is an Eclipse based graphical modeling environment developed with the Cinco SCCE Meta Tooling Suite [42]. It is a low-code application development environment that follows the philosophy of OTA (One Thing Approach) [34] and the eXtreme Model Driven Development paradigm [36, 37] to support the design, development, and deployment of (originally web) applications in an agile way. DIME empowers application domain experts that are not proficient coders/programmers to fully participate in the entire design, development and evolution process because it supports easy modelling, done by means of drag and drop of pre-existing components. For separation of concerns, DIME supports several model types that express distinct perspectives on the same comprehensive model. This “write once” rule is the essence of the coherence by construction principle central to the One Thing Approach. The DIME model types encompass:

- A **Data model**, which covers the persistence layer (both types and relations) of the application in a form similar to a UML class diagram.
- A collection of **Process models**, that define the business logic on the basis of internal and external libraries of basic functionalities provided by means of the

Native DSL mechanism. Each DSL exposes a collection of SIBs (for Service Independent Building blocks), that are reusable, instantiable and executable modeling components with either an associated code implementation or an associated hierarchical process model.

- A collection of **GUI models**, defining the elements (look and feel, actions and navigation) of the pages of the web application, and
- **Security and Access control models**, mainly handling the security and access permission aspects of the application.

This is different, for example, from the standard UML models [50]: UML and related tools support a variety of different model types (static, like UML class diagrams and DIME’s Data model, and dynamic like UML’s activity diagrams and DIME’s process models) serving different purposes, but those models/model types are not connected among each other. Therefore, it is very easy in UML to breach consistency of the overall model collection, because changes do not propagate from one model to the other.

We value DIME’s characteristics of open source, flexibility, ease of extension, support of high-assurance software quality, agility, service-oriented approach, and containerization. For the specific low-code support, its model-driven approach is based on DSLs at two levels:

1. **Language DSLs**, as a mechanism to design and implement the application design environment itself, i.e., the Integrated Modeling Environment (IME), and
2. **Application domain DSLs**, at application design time. We want to use **Native DSLs** as the means to integrate and expose collections of capabilities offered by end devices and other sources of functionalities to the application designers, and **Process DSLs** as the means to foster reuse of medium and large grained business logic across applications.

As different models cover different aspects of the target application, to ensure the intended behavior each model of the application is validated at compile time both at DSL and platform level for syntactic and semantic errors. After validation, these models act as input for subsequent model-to-code transformation phases. The key design principles of DIME are simplicity [39], agility [35] and quality assurance [59], hence, DIME is a promising “game changer” low code development environment (LCDE) for the realization of sophisticated web applications in tremendously shorter development cycles.

4 Digital Thread Platform: The Current Status

We target the application domain of advanced manufacturing including manufacturing analytics. Accordingly, we intend to support the conception, design and implementation of a set of applications, like for example robotics navigation and control, proactive maintenance, MES monitoring, but also analytics dashboards that analyse or summarise in real time or near-real time data provenient

Fig. 2. Architecture Overview of DIME and Custom DSLs

from various systems and subsystems of a complex, possibly distributed production plant. In this context, data, processing and communications are expected to concern a large variety of devices, data sources, data storage technologies, communication protocols, analytics or AI technologies and tools, visualization tools, and more. This is where the integration of external native DSLs plays a key role. The current architecture of the Digital Thread platform is depicted in Fig. 2.

We see that DIME’s **Language DSL**, used to design the applications, encompasses for the moment in the advanced manufacturing setting primarily the Data, Process and GUI models.

We also see that already a significant variety of external platforms (like EdgeX for IoT), technologies (like REST services, or R for analytics) and tools (like the UR family of robots) have been integrated. All these are part of the **Application DSL** layer mentioned in Sect. 3, including quite a number of **Native DSLs** external to DIME.

The central property of simplicity here is that, once integrated, the Native DSLs all look “alike” within DIME: the collection of individual functionalities has an own, but uniform representation, and their use within DIME is uniform as well. This means that once a DIME user has learned how to work with the three model types and with the basic functionalities, this user can produce high quality applications that span a variety of technologies and application domains without need to be able to master any of their underlying technologies, programming languages, or communication protocols, as these are part of the encapsulation of this heterogeneity within the DSLs, and its virtualization by means of the uniform representation and handling. Note that this approach is not completely unusual: with more or less success, generations of platforms have pursued this goal. Some platforms are domain specific and special purpose, like for example EdgeX [1] for the provision of an extensible, uniform service-oriented middleware for (any of the) supported IoT devices and their management. EdgeX defines itself as “the preferred Edge IoT plug and play ecosystem-enabled open software platform” [1], a “highly flexible and scalable open-source framework that facilitates interoperability between devices at the IoT edge”. Its data model is YAML profiles, its exposed services are implemented as REST microservices, it supports the C and Go programming languages for users to write their own orchestrations (instead of DIME’s process models). It does not support GUI models as this user interfaces are not an aspect in their focus. Other platforms have a broader scope. For example, GAIA-X [3] aspires to become “a federated data infrastructure for Europe”. Among the platforms that have meanwhile over a decade of history, FI-WARE [2] describes itself as “the Open-Source Platform for Our Smart Digital Future” and offers a wide collection of services and service components that can be reused by application developers.

They all require programming ability, none of them offers a low-code approach, they all provide collections of reusable components, and do not envisage support for the orchestration on top. Their view is the bottom-up approach of component provision, that an expert will then somehow orchestrate.

In this respect, our value proposition sits clearly at the upper, application development layer, where we see the interoperability challenge truly reside.

We also see ourselves as systematic users of such pre-existing platforms, who are for us indeed welcome providers of Native DSLs. In this context, a number of integrations in DIME relevant to the advanced manufacturing domain have already been addressed.

Seen from an **Application domain** point of view, for example, the following have already been integrated:

- A IoT through (some parts of) EdgeX [19,20]
- Robotics through the UR command language [32]
- Persistency layer through various data storage alternatives, from CSV files, to relational (PostgreSQL) and no-SQL (MongoDB) databases (own work)
- Cloud services [10]
- Data analytics with R libraries (own work)

and own work is ongoing on

Table 1. SIBs information for REST services integration.

SIB name	Input(s)	Output(s)	Explanation
Rest_read_str_str	Url Input_var Input_val Output_var	Output_val (single var)	This SIB accepts inputs required to initiate the communication with an external REST service and receives a single variable as result
Rest_read_str_list	Url Input_var Input_val Output_var	Output_val (array/list)	This SIB accepts inputs required to initiate the communication with an external REST service, receives as result an array or a list and feeds it to the subsequent SIB for iterative processing of the elements

- some forms of AI and Machine Learning (classifiers, Grammatical Evolution [15], and more)
- Robotics through ROS, additional to [12, 21, 32]
- Distributed Ledger Technologies through blockchain
- Visualization tools with, e.g., Quickchart

Seen from a **Technology portfolio** point of view,

- REST services [10]
- R, seen as a programming language (own work)
 - are already supported, and the next months will see own work on
- Matlab and Julia, as programming languages/tools for simulations
- MQTT and other native IoT protocols, as in some cases it is impractical to have to use EdgeX.

In the following, we will provide some details on a few selected examples of these integrations.

4.1 REST Services

A case study [10] details a generic extension mechanism, where two LCDE platforms based on formal models were extended following the analogy of microservices. This extended the capabilities of DIME by integrating cloud and web services thorough REST. RESTful APIs are a standardized way how applications can communicate, firstly described by Roy in his PhD Thesis [13], have become one of the most used APIs schemas. DIME uses REST to share information between the front and the back end. While the commands are encoded via the widely supported HTTP standard, data can be exchanged in many formats. The most common data format is the Java Script Object Notation (JSON), but also Extended Markdown Language (XML) and others can be used.

In this context, this new DIME DSL allows to act as client for those APIs, i.e., to send request to external applications and to decode JSON responses into the data domain of DIME. Table 1 shows a list of sample SIBs with relevant IOs and explanation.

Fig. 3. DIME Process for the UR robot position control

4.2 Robotics with the UR Language

UR3 is a well-known lightweight collaborative robotic arm designed to work on assembly lines and production environments in the smart manufacturing context. The robotic arm is not only easy to install but has a simple command language to program all the tasks required, with a tethered tablet. The paper [32] showed how to build a remote controller through a DIME Web application that manages the remote communication with UR cobots and the commands through a UR-Family native DSL. Figure 3 shows the hierarchical process model in DIME for the outer working of the controller: the robot is initialized (started and ready to respond), it is sent to an initial position to test the correct functioning of the command channel, then the program with the real task is uploaded (this is itself a DIME a process) and the communication is then closed upon execution completion. Table 2 shows a list of sample SIBs with relevant IOs and explanation.

4.3 Data Management via Files and External Databases

DIME supports basic files handling operations, sufficient for text and Comma Separated Value (CSV) files. However, handling large datasets requires coordination with dedicated structured or non-structured databases. Recent work integrates MongoDB Atlas and Elephant SQL, two fully managed NOSQL cloud databases, and the PostgreSQL database service. The integrations use the MDD approach to provide functionalities to import and export data from/to these

Table 2. SIBs information for robotic arm integration.

SIB name	Input(s)	Output(s)	Explanation
InitiateRobot	No input	result (boolean)	The SIB does not need any input. It reads the robotic configurations from a pre-defined location, initiates a connection with the robotic arm and returns a boolean status code according to the execution outcome
MoveRobotToTestPosition	No input	result (boolean)	The SIB reads predefined coordinates from the application code, sends them to the robotic arm over a network connection. After the mechanical movement, it returns the according boolean status code
MoveCoordinatesRobot	x y z ax ay az	result (boolean)	The SIB accepts a set of coordinates (3D position, 3 angles) as input, sends them to the robotic arm requesting it to move to this position, then returns a boolean status code
StopRobot	No input	result (boolean)	The SIB sends to the robotic arm the stop command, with instructions to shut down and stop the communication, and returns a boolean status code

storage alternatives - an essential capability for the data interoperability and data migration in the Digital Thread platform. Table 3 shows a list of sample SIBs from the MongoDB integration with the relevant IOs and explanations.

4.4 Analytics with R

DIME is built upon J2EE and supports all its functionalities and capabilities. However, specialized languages and platforms like MATLAB for simulations and R for data analytics are optimized for those tasks and need to be supported in a proper Digital Thread platform. We recently extended DIME with the R environment by encapsulation through a Native DSL shown in Table 4. Figure 4 shows the runtime architecture: the application and the R environment are deployed in two different docker containers. The Rserve library is the entry point of the R environment, it handles all the external communication using TCP/IP. DIME uses this mechanism to provide the R data analytics capabilities.

The impact of having a platform mindset is that the functionality needs to be implemented only once and is reusable across multiple domains by very different domain experts, as illustrated in Fig. 5 and Fig. 6. The same `plot_R_histogram` SIB is used in fact in Fig. 5 (left) with a manufacturing domain dataset to draw

Table 3. SIBs information for external databases (MongoDB) integration

SIB name	Input(s)	Output(s)	Explanation
ConnectMongoDB	ConnectionString	result (boolean)	The SIB accepts as input a ConnectionString containing the address and credentials of the server. The boolean result tells whether or not the connection is successful
ReadfromDB	projectName databaseName attributeID	result	The SIB accepts the relevant project, database and attribute names and returns the result of the query as a single or multiple tuples

Fig. 4. Runtime infrastructure of DIME and R - environment

the histogram of manufacturing fitting failures per installation year (left), and in Fig. 6 on the Irish census data of 1901: in this history/humanities domain the same SIB is used to visualize the breakdown of the 1901 population by age.

5 Programming: What's Next?

Considering the questions posed to the authors in this Special Track, we answer them briefly from the point of view of the technologies described in this paper, considering also our experience in projects and education.

1. *What are the trends in classical programming language development?, both wrt. applications programming and systems/embedded programming?*

While the state of the art in these domains is still dominated by traditional,

Table 4. SIBs information for the R – Environment integration.

SIB name	Input(s)	Output(s)	Explanation
execute.R_instructions	instruction	result	The SIB accepts a single R – language instruction as input and returns the execution result (if there is any) from the R – environment
execute.R_file	file_name	result	The SIB accepts a filename as input, executes the batch of R instructions contained in that file in the R environment and returns the execution result
plot.R_histogram	file_name col_name title x_label y_label color	image	The SIB accepts the dataset, parameters and formatting details as input, generates a histogram in the R environment and returns the resulting image
plot.R_wordcloud	file_name col_name min_frequency max_words	image	The SIB accepts the dataset, parameters and specification details as input, generates a word cloud in the R environment and returns the resulting image

hand-coded software, the low-code development wave is reaching industry adoption and a certain degree of maturity. So far it is more prominent in the general application programming and not yet in the CPS/embedded systems domain, but that is in our opinion a matter of diffusing across communities. We are surely working to reach the embedded systems, CPS and Industry 4.0 industrial adopters for our methods.

2. *What are the trends in more experimental programming language development, where focus is on research rather than adoption? This includes topics such as e.g. program verification, meta-programming and program synthesis.* In this context, we see the evolution of meta-programming from the classic and traditional UML-driven community and mentality, that we see still prevail in recent surveys [46], towards the more radical approach promoted by Steffen et al. via Language Driven Engineering [54] and purpose driven collaboration using purpose specific languages (PSLs) [64]. This is a powerful, yet still niche, area of research and adoption. In this line of thought, also [6] advocates intent-based approaches and platforms as a way of channelling complexity by focusing on what matters. As adopters of the LDE and DSLs paradigms through the use of the Cinco-products DIME and Pyro/Pyrus [62,63], we see the advantages and the power of these new paradigms and tools. The need to understand the platforms, the various levels of “meta” and their interplay, which needs to be respected and embraced, require more understanding of the interna of these paradigms, their implementations, and also the limitations imposed by

Fig. 5. Histogram plotting in R: SIB instance in the Manufacturing domain (manufacturing fitting failures per year)

Fig. 6. Histogram plotting in R: SIB instance in the humanities domain (1901 census population breakdown by age)

the languages and platforms they at the end based upon (like Eclipse, E-core, and more). This is also underlined by Lethbridge [26], who provides also recommendations for the next generation of Low-code platforms. Core advantages of model-driven and low-code taken together are in the rapidity of evolution, and the precision of the generated artefacts. Taking out the human factor from a number of steps in the software implementation process may eliminate some genial solutions, but it also eliminates a wealth of errors, misunderstandings, and subjective local decisions that may be incoherent with other local decisions elsewhere. This enforced “uniformity by generation” has the advantage of enforcing a standard across the generated code base, and a generation standard is less unpredictable and easier to maintain and evolve.

In terms of program synthesis, we have a long experience in synthesis of workflows [33], of mashups and web services [29,31], of applications in robotics and bioinformatics [22,30] and of benchmark programs with well defined semantic profiles [55]. The potential for application to low-code and in particular

no-code development environments that support a formal methods-underpinned semantics is certainly enticing. The fact is, so far the popular platforms of that kind do not have a formal semantics, and in this sense the Cinco-DIME-Pyro family of platforms is indeed quite unique.

3. *What role will domain-specific languages play and what is the right balance between textual and graphical languages?*

Concerning DSLs, we are keen adopters of them both at the language design level (as in DIME) and at the application domain level, with the External native DSLs. In our experience, they are useful to address the knowledge, the terminology and the concerns of both programmers and non-programmer stakeholders in a collaborative application development team. They are a key element of the bridge building [28] so necessary to get the right things right. Currently, most domain specific languages are at the coding level and do not leverage a model driven approach at the platform level. On the DSL side, the internal DSLs built in Scala of [16] address specific aspects in the design of embedded systems. They are an attractive step towards the preparation of abstractions that can connect well with the modelling level. The construction of meta-models behind these DSLs is challenging, since they must capture all the domain knowledge, i.e. provide both semantic and syntactic rules. For example, Ktrain [27] is a popular coding level DSL: a python wrapper that encapsulates Tensor Flow functionalities and facilitates developers to augment machine learning tasks with fewer lines of python code.

We see the graphical presentation of, specifically, coordination languages as an advantage for those tasks that privilege evidence and intuition. In this sense, “seeing” a workflow and a dataflow in a native representation as in DIME and Pyrus exposes some errors in a more self-evident way than if this representation had to be first derived from the linear syntax of customary code. Extracting again the Control Flow Graph and the Dataflow Graph, e.g., is common practice to then analyze dependencies or do the meanwhile well established program analysis and verification. We see an advantage to use them as the explicit, mathematically correct, representation facing the designers rather than to extract them from the traditional program code where they are only implicitly present.

4. *What is the connection between modeling and programming?*

In the light of the above, the connection is tight between, e.g., the program models used in DIME and the code they represent. We are here concentrating on the software that enables the operation, in particular the interoperation and control, of applications and systems, and therefore we do not delve into the kind of cyberphysical systems modelling that concerns the physics, mechanics, and general simulation models.

In terms of our own experience, being able to cover a variety of models in a single IME is a great advantage. The METAFrame [56] and jABC platforms [57] supported only process models, and even in DyWA [43] the integration between data model and process models happened through import/export

across two tools. In comparison, the current integration of language DSLs in DIME provides a level of comfort, ease of development and built-in checks that makes DIME a success in our teaching of agile development to undergraduates and postgraduates.

5. *Will system development continue to be dominated by programming or will there be radical changes in certain application areas/generally? E.g. driven by breakthroughs in AI and machine learning techniques and applications.*

Next to the traditional hand-coded programming and the full AI/ML based approach, we see a significant and growing role for the XMDD style of modelling [36,37], that we see as an intermediate paradigm, more controllable, analyzable and explainable than those based on AI/ML. In our opinion it covers the sweet spot between these two schools of thought and practice.

Several other approaches seem to inhabit this middle too: CaaSSET [40] is a Context-as-a-Service based framework to ease the development of context services. The transformation into executable services is semi-automatic. Agent-based modelling paradigm [53] is another popular approach to increase the development productivity in simulation environments.

In terms of AI support, for example, Xatkit [11], still in early stages of development, increases the reusability of chat bots by evolving NLP/NLU engine for text analytics. At the language level they support several versions of bots, but the generation of chatbots from existing data sources at the framework level is in future plans.

In terms of trends that have an influence on the programming and modelling philosophy, service orientation and more recently microservices play a significant role. This architectural style that tries to focus on building single-function modules with well-defined interfaces and operations can be seen in part as an evolution of web services [7], in a trend towards the production of fine-grained systems [44] that seems to conceptually align with the growing attention to limiting scope in order to tame complexity. There are graphical approaches [48], but mostly they use standard programming languages. Dedicated programming languages like Jolie [4] offer native abstractions for the creation and composition of services, but add to the layers of infrastructure needed to develop and then execute microservices. Here, we see our abstraction as one level higher, so that we integrate microservices as simply just one additional flavour of decentralized execution [10], building on previous experience with Webservices and WSDL.

6. *Is teaching classical programming as third discipline sensible/required?*

We would advocate that an XMDD approach based on DSLs as we have presented is easier to understand, largely (programming) language and application domain independent. In our approach, the largest part of these technical, infrastructural and knowledge layers are dealt with by IT and programming professionals who integrate the domains and this way encapsulate them. What users do see, in terms of Native DSLs and the coordination layer, has a domain specific meaning but a language and domain independent general syntax and

semantics. Accordingly, we would consider it a better choice of abstraction level to bring to the masses of professionals as third discipline than the traditional programming in one paradigm/language, which is necessarily a very specialized choice.

There are also other frameworks in the making: for example, Aurera [58] is a low-code platform for automating business processes in manufacturing. It is standalone desktop system that addresses the challenges of frequent changes to IT solutions. It is however still in early stages of development and does not support communication with external systems.

7. *Can we imagine something like programming for everybody?*

Yes, we can! And the XMDD paradigm for Low-code and no-code application development is in our experience a strong candidate toward that aim.

6 Conclusion and Outlook

We addressed the principles, the architecture and the individual aspects of growing Digital Thread platform we are building, which conforms to the best practices of coordination languages. Through the adoption of the Low-Code Development Environment DIME it supports a level of reuse, refactoring and analysis at the coordination layer that goes beyond what is achieved today with the current practice of glue code. We illustrated the current status, and described various extension through generic REST services, to robotics through the UR family of robots, to the integration of various external databases (for data integration) and to the provision of data analytics capabilities in R.

We are currently working in various collaborative contexts to enrich the set of supported DSLs, as shown in Fig. 2. The choice of what to address next depends on the needs arising in various contexts, and it is limited by the time and staff available. The snowball effect of the impact has however already kicked in: in more than one case, a new application, sometimes in a completely different domain and collaboration, has already been able to avail of existing native DSLs, or even processes, developed in a totally different context.

Over time, we expect reuse to be increasingly the case, reducing the new integration and new development effort to a progressively smaller portion of the models and code needed for at least the most standard applications. We also expect this kind of paradigm to attract the attention of those sectors and industries that require a tighter cooperation between stakeholders with different expertise and knowledge, where there is a lack of skilled developers, and where the need for a faster turn around time can make code generation attractive as a form of automation.

Acknowledgement. This work was supported by the Science Foundation Ireland grants 16/RC/3918 (Confirm, the Smart Manufacturing Research Centre) and 13/RC/2094.2 (Lero, the Science Foundation Ireland Research Centre for Software).

References

1. Edgex foundry: The edgex foundry platform. <https://www.edgexfoundry.org/>. Accessed July 2021
2. Fi-ware: The open-source platform for our smart digital future. <https://www.fiware.org/>. Accessed July 2021
3. Gaia-x: A federated data infrastructure for Europe. <https://www.data-infrastructure.eu/>. Accessed July 2021
4. Jolie: The service-oriented programming language. <https://www.jolie-lang.org/index.html>. Accessed Sept 2021
5. Allian, A.P., Schnicke, F., Antonino, P.O., Rombach, D., Nakagawa, E.Y.: Architecture drivers for trustworthy interoperability in industry 4.0. *IEEE Syst. J.* (2021)
6. Balasubramanian, D., Coglio, A., Dubey, A., Karsai, G.: Towards model-based intent-driven adaptive software. In: *Proceedings of ISO/ISA 2021. LNCS*, vol. 13036 (2021)
7. Baresi, L., Garriga, M.: Microservices: the evolution and extinction of web services? *Microservices*, pp. 3–28 (2020)
8. Boßelmann, S., et al.: DIME: a programming-less modeling environment for web applications. In: Margaria, T., Steffen, B. (eds.) *ISO/ISA 2016. LNCS*, vol. 9953, pp. 809–832. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_60
9. Burns, T., Cosgrove, J., Doyle, F.: A review of interoperability standards for industry 4.0. *Procedia Manuf.* **38**, 646–653 (2019)
10. Chaudhary, H.A.A., Margaria, T.: Integration of micro-services as components in modeling environments for low code development. In: *Proceedings of the Institute for System Programming of the RAS*, vol. 33, no. 4 (2021, September (in print))
11. Daniel, G., Cabot, J., Deruelle, L., Derras, M.: Xatkit: a multimodal low-code chatbot development framework. *IEEE Access* **8**, 15332–15346 (2020). <https://doi.org/10.1109/ACCESS.2020.2966919>
12. Farulla, G.A., Indaco, M., Legay, A., Margaria, T.: Model driven design of secure properties for vision-based applications: a case study. In: *Proceedings of the International Conference on Security and Management (SAM)*, p. 159. The Steering Committee of The World Congress in Computer Science, Computer (2016)
13. Fielding, R.T.: *Architectural Styles and the Design of Network-Based Software Architectures*. University of California, Irvine (2000)
14. Geraci, A.: *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*. IEEE Press (1991)
15. Guevara, I., Margaria, T.: Mazegen: an evolutionary generator for bootstrapping robotic navigation scenarios. In: *37th International Manufacturing Conference, Irish Manufacturing Council, September 2021, Athlone, Ireland* (2021)
16. Havelund, K., Bocchino, R.: Integrated modeling and development of component-based embedded software in scala. In: *Proceedings ISO/ISA 2021. LNCS*, vol. 13036 (2021)
17. Jepsen, S.C., Mørk, T.I., Hviid, J., Worm, T.: 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)*, pp. 571–575. IEEE (2020)
18. Jirkovský, V., Obitko, M., Mařík, V.: Understanding data heterogeneity in the context of cyber-physical systems integration. *IEEE Trans. Ind. Inform.* **13**(2), 660–667 (2016)

19. John, J., Ghosal, A., Margaria, T., Pesch, D.: Dsls and middleware platforms in a model driven development approach for secure predictive maintenance systems in smart factories. In: Proceedings ISoLA 2021. LNCS, vol. 13036. Springer (2021)
20. John, J., Ghosal, A., Margaria, T., Pesch, D.: Dsls for model driven development of secure interoperable automation systems. In: Proceedings Forum for Specification and Design Languages (FDL). IEEE Press (September 2021 (in print))
21. Jorge, S., Kubczak, C., Pageau, F., Margaria, T.: Model driven design of reliable robot control programs using the jABC. In: Proceedings EASE 2007, vol. 07, pp. 137–148 (2007). <https://doi.org/10.1109/EASE.2007.17>
22. Jörges, S., Kubczak, C., Pageau, F., Margaria, T.: Model driven design of reliable robot control programs using the jABC. In: Proceedings of 4th IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASE 2007), pp. 137–148 (2007)
23. Kovatsch, M., Hassan, Y.N., Mayer, S.: Practical semantics for the internet of things: physical states, device mashups, and open questions. In: 2015 5th International Conference on the Internet of Things (IOT), pp. 54–61. IEEE (2015)
24. Lee, H., Ryu, K., Cho, Y.: A framework of a smart injection molding system based on real-time data. *Procedia Manuf.* **11**, 1004–1011 (2017)
25. Lelli, F.: Interoperability of the time of industry 4.0 and the internet of things. *Fut. Internet* **11**(2), 36 (2019)
26. Lethbridge, T.C.: Low-code is often high-code, so we must design low-code platforms to enable proper software engineering. In: Proceedings ISoLA 2021. LNCS, vol. 13036 (2021)
27. Maiya, A.S.: ktrain: a low-code library for augmented machine learning. arXiv preprint [arXiv:2004.10703](https://arxiv.org/abs/2004.10703) (2020)
28. Margaria, T., Steffen, B.: Service engineering: linking business and IT. *Computer* **39**(10), 45–55 (2006)
29. Margaria, T., Bakera, M., Kubczak, C., Naujokat, S., Steffen, B.: Automatic generation of the SWS-challenge mediator with jABC/ABC. In: Petrie, C., Margaria, T., Zaremba, M., Lausen, H. (eds.) *Semantic Web Services Challenge. Results from the First Year*, pp. 119–138. Springer Verlag (2008). <https://doi.org/10.1007/978-0-387-72496-6>
30. Margaria, T., Kubczak, C., Njoku, M., Steffen, B.: Model-based design of distributed collaborative bioinformatics processes in the jABC. In: Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2006), Los Alamitos, CA, USA, pp. 169–176. IEEE Computer Society, August 2006
31. Margaria, T., Meyer, D., Kubczak, C., Isberner, M., Steffen, B.: Synthesizing semantic web service compositions with jMosel and Golog. In: Bernstein, A., et al. (eds.) *ISWC 2009*. LNCS, vol. 5823, pp. 392–407. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04930-9_25
32. Margaria, T., Schieweck, A.: The digital thread in Industry 4.0. In: Ahrendt, W., Tapia Tarifa, S.L. (eds.) *IFM 2019*. LNCS, vol. 11918, pp. 3–24. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34968-4_1
33. Margaria, T., Steffen, B.: Backtracking-free design planning by automatic synthesis in metaframe. In: Astesiano, E. (ed.) *FASE 1998*. LNCS, vol. 1382, pp. 188–204. Springer, Heidelberg (1998). <https://doi.org/10.1007/BFb0053591>
34. Margaria, T., Steffen, B.: Business process modeling in the jABC: the one-thing approach. In: *Handbook of Research on Business Process Modeling*, pp. 1–26. IGI Global (2009)

35. Margaria, T., Steffen, B.: Simplicity as a driver for agile innovation. *Computer* **43**(6), 90–92 (2010)
36. Margaria, T., Steffen, B.: Service-orientation: conquering complexity with XMDD. In: Hinchey, M., Coyle, L. (eds.) *Conquering Complexity*, pp. 217–236. Springer, London (2012). https://doi.org/10.1007/978-1-4471-2297-5_10
37. Margaria, T., Steffen, B.: extreme model-driven development (XMDD) technologies as a hands-on approach to software development without coding. In: *Encyclopedia of Education and Information Technologies*, pp. 732–750 (2020)
38. Mellor, S.J., Clark, T., Futagami, T.: Model-driven development: guest editors' introduction. *IEEE Softw.* **20**(5), 14–18 (2003). ISSN 0740–7459
39. Merten, M., Steffen, B.: Simplicity driven application development. *J. Integr. Des. Process Sci.* **17**(3), 9–23 (2013)
40. Moradi, H., Zamani, B., Zamanifar, K.: CaaSSET: a framework for model-driven development of context as a service. *Future Gener. Comput. Syst.* **105**, 61–95 (2020)
41. Napoleone, A., Macchi, M., Pozzetti, A.: A review on the characteristics of cyber-physical systems for the future smart factories. *J. Manuf. Syst.* **54**, 305–335 (2020)
42. Naujokat, S., Lybecait, M., Kopetzki, D., Steffen, B.: CINCO: a simplicity-driven approach to full generation of domain-specific graphical modeling tools. *Int. J. Softw. Tools Technol. Transf.* **20**, 1–28 (2018). <https://doi.org/10.1007/s10009-017-0453-6>
43. Neubauer, J., Frohme, M., Steffen, B., Margaria, T.: Prototype-driven development of web applications with DyWA. In: Margaria, T., Steffen, B. (eds.) *ISoLA 2014*. LNCS, vol. 8802, pp. 56–72. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45234-9_5
44. Newman, S.: *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Inc. (2015)
45. Nilsson, J., Sandin, F.: Semantic interoperability in Industry 4.0: survey of recent developments and outlook. In: *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, pp. 127–132. IEEE (2018)
46. Ordoñez, K., Hilera, J., Cueva, S.: Model-driven development of accessible software: a systematic literature review. In: *Universal Access in the Information Society*, pp. 1–30 (2020)
47. Pang, T.Y., Pelaez Restrepo, J.D., Cheng, C.T., Yasin, A., Lim, H., Miletic, M.: Developing a digital twin and digital thread framework for an 'Industry 4.0' shipyard. *Appl. Sci.* **11**(3), 1097 (2021)
48. Rademacher, F., Sorgalla, J., Wizenty, P., Sachweh, S., Zündorf, A.: Graphical and textual model-driven microservice development. In: *Microservices*, pp. 147–179. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-31646-4_7
49. da Rocha, H., Espirito-Santo, A., Abrishambaf, R.: Semantic interoperability in the Industry 4.0 using the IEEE 1451 standard. In: *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, pp. 5243–5248. IEEE (2020)
50. Rumpe, B.: *Modeling with UML*. Springer, Cham (2016). <https://doi.org/10.1007/978-3-319-33933-7>
51. Sanchez, M., Exposito, E., Aguilar, J.: Industry 4.0: survey from a system integration perspective. *Int. J. Comput. Integr. Manuf.* **33**(10–11), 1017–1041 (2020)
52. Sanchis, R., García-Perales, O., Fraile, F., Poler, R.: Low-code as enabler of digital transformation in manufacturing industry. *Appl. Sci.* **10**(1), 12 (2020)
53. Santos, F., Nunes, I., Bazzan, A.L.: Quantitatively assessing the benefits of model-driven development in agent-based modeling and simulation. *Simul. Model. Pract. Theory* **104**, 102126 (2020)

54. Steffen, B., Gossen, F., Naujokat, S., Margaria, T.: Language-driven engineering: from general-purpose to purpose-specific languages. In: Steffen, B., Woeginger, G. (eds.) *Computing and Software Science*. LNCS, vol. 10000, pp. 311–344. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91908-9_17
55. Steffen, B., Howar, F., Isberner, M., Naujokat, S., Margaria, T.: Tailored generation of concurrent benchmarks. *Softw. Tools Technol. Transf.* **16**(5), 543–558 (2014). <https://doi.org/10.1007/s10009-014-0339-9>
56. Steffen, B., Margaria, T.: METAFramE in practice: design of intelligent network services. In: Olderog, E.-R., Steffen, B. (eds.) *Correct System Design*. LNCS, vol. 1710, pp. 390–415. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48092-7_17
57. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-driven development with the jABC. In: Bin, E., Ziv, A., Ur, S. (eds.) *HVC 2006*. LNCS, vol. 4383, pp. 92–108. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70889-6_7
58. Waszkowski, R.: Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine* **52**(10), 376–381 (2019)
59. Windmüller, S., Neubauer, J., Steffen, B., Howar, F., Bauer, O.: Active continuous quality control. In: *Proceedings of the 16th International ACM Sigsoft Symposium on Component-Based Software Engineering*, pp. 111–120 (2013)
60. Xu, L.D.: The contribution of systems science to Industry 4.0. *Syst. Res. Behav. Sci.* **37**(4), 618–631 (2020)
61. You, Z., Feng, L.: Integration of Industry 4.0 related technologies in construction industry: a framework of cyber-physical system. *IEEE Access* **8**, 122908–122922 (2020)
62. Zweihoff, P., Naujokat, S., Steffen, B.: Pyro: generating domain-specific collaborative online modeling environments. In: Hähnle, R., van der Aalst, W. (eds.) *FASE 2019*. LNCS, vol. 11424, pp. 101–115. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-16722-6_6
63. Zweihoff, P., Steffen, B.: Pyrus: an online modeling environment for no-code data-analytics service composition. In: *Proceedings ISoLA 2021*. LNCS, vol. 13036 (2021)
64. Zweihoff, P., Tegeler, T., Schürmann, J., Bainsczyk, A., Steffen, B.: Aligned, purpose-driven cooperation: The future way of system development. In: *Proceedings ISoLA 2021*. LNCS, vol. 13036 (2021)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.