The Digital Thread in Industry 4.0

Tiziana Margaria(✉) and Alexander Schieweck

Chair of Software Systems, University of Limerick, and Confirm, Limerick, Ireland
{tiziana.margaria,alexander.schieweck}@ul.ie

Abstract. Industry 4.0, the new wave of Smart Manufacturing in Europe and globally, relies on a Digital Thread to connect the data and processes for smarter products, smarter production, and smarter integrated ecosystems. But what is the Digital Thread?

We discuss a few key questions about modelling, the nature of models and the use of models that arose from the experience in the first two years of Confirm, the Irish Centre for Smart Manufacturing. We also provide an example of how the new model-powered and integrated thinking can disrupt the status quo, empower a better understanding, and deliver a more automatic management of the many cross-dimensional issues that future connected software and systems will depend upon.

1 From the Digital Twin to the Digital Thread

The Industry 4.0 movement towards smart advanced manufacturing presupposes a thorough revolution in the perception and practice of the "Art of Manufacturing" that shifts the innovation focus increasingly from the traditional crafts and engineering, like mechanical engineering, materials science and plants construction, to immaterial assets, like knowledge, flexibility and ultimately models for their exploration 'in silico'. The role of models is well understood for what concerns blueprints of facilities, CAD of machines and parts, P&IDs of various tooling processes in production [68], and various behavioural models like stress curves of materials, yield of various processes, and isolated simulations. The new trend of modelling concerns the *Digital Twin* of processes, machines and parts. That the essence of a Digital Twin is not yet stabilized is witnessed by the fact that everyone seems to have a different definition. For example, as of today Wikipedia [67] provides an own description (note: not a definition) as "*A digital twin is a digital replica of a living or non-living physical entity.*[1] *By bridging the physical and the virtual world, data is transmitted seamlessly allowing the virtual entity to exist simultaneously with the physical entity. Digital twin refers to a digital replica of potential and actual physical assets (physical twin), processes, people, places, systems and devices that can be used for various purposes.*" and "*Definitions of digital twin technology used in prior research emphasize two important characteristics. Firstly, each definition emphasizes the connection between the physical model and the corresponding virtual model or virtual counterpart.* [8] *Secondly, this connection is established by generating real time data using sensors.* [2]". Wikipedia then lists 10 different definitions, from

W. Ahrendt and S. L. Tapia Tarifa (Eds.): IFM 2019, LNCS 11918, pp. 3–24, 2019.
https://doi.org/10.1007/978-3-030-34968-4_1

2012 to 2019, that include references to as diverse concepts as multilevel, multiphysics, multiscale, real time, cloud platform, lifecycle, health condition, and many more. A recent, quite realistic and encompassing definition is by Ashtari et al. [65]: "*The Digital Twin is a virtual representation of a physical asset in a Cyber-Physical Production System (CPPS), capable of mirroring its static and dynamic characteristics. It contains and maps various models of a physical asset, of which some are executable, called simulation models. But not all models are executable, therefore the Digital Twin is more than just a simulation of a physical asset. Within this context, an asset can be an entity that already exists in the real world or can be a representation of a future entity that will be constructed.*"

In this context - an adaptation of Cyberphysical Systems to the production environment - most engineers accept the need of continuous models or discretized numerical simulation models like in finite element analysis. They are however not familiar with models for software, nor are they aware that software, and thus software correctness and thus software quality, are essential to the information propagation, aggregation, and analysis that combines collections of such models to deliver or at least enable the desired insight from those models. In other words, the **Digital Thread** that connects real things and their twin models, but also the communication networks, the decision algorithms, the visualisations needed to work in design, construction, and operation within a mature Industry 4.0 environment are still not in the sphere of awareness of the responsibles. Thus software in general and software models in particular are hopelessly underestimated in their relevance, complexity, challenges and cost.

The modern version of the Digital Thread can be seen as the information-relay framework which enables the holistic view and traceability of an asset along its entire lifecycle. This framework includes any data, behaviours, models, protocols, security, and their standards related to the asset as well as to the context where it is expected to operate. As such, it goes well beyond the customary understanding of the thread as mostly a collection of data, cast in terms of the Digital Twin as a new way of managing variants in a Product Line Engineering context and with the Digital Thread being the new counterpart of the established Product Lifecycle Management [13].

Being a Principal Investigator in Confirm responsible for the co-direction of the Cyberphysical Systems research hub, involved in the Virtual and Physical Testbeds hub, and working with Digital Twins to deliver the Confirm platform, the current unawareness in industry and academia alike about the high demands and enormous potential of the Digital Thread are a threat and an opportunity at the same time. We found out that the first step of communication needed to reach professionals who do not know much about software, less about models, and even less about (discrete) behavioural models for software and systems, is to build a micro-size demonstrator in exactly their domain.

In the rest of the paper we provide a description of the micro-demo we built this Summer, essentially a web based UR3 robot controller (Sect. 2), we give a little background on the Model-driven Integrated Development Environment

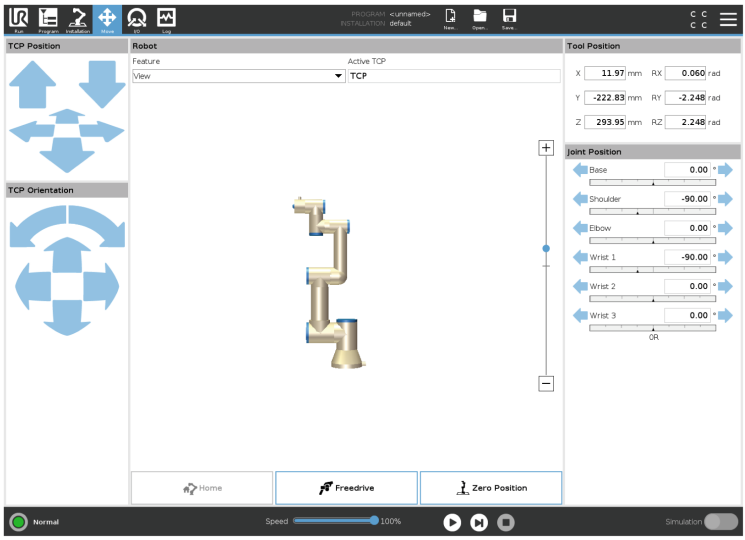

Fig. 1. The simulator provided by UR.

DIME [5] that we used to build it, and the Robotics DSL (domain specific language) we built to integrate robotics capabilities in DIME (Sect. 3). Then we provide some final considerations about the prospective of using this approach and framework as the technical platform to build the Confirm Digital Thread and Digital Twin shared assets (Sect. 4).

2 Bringing the UR3 Controller in the Web

The mini-project we identified as the first nugget of model driven design and development of a demonstrator in the robotic field for Confirm literally *unleashes* the normally tethered UR3 robot control. Universal Robots, an originally Danish company acquired in 2015 by Teradyne, is a leader in collaborative robots (cobots), i.e. robots that can work sharing the same space with humans, instead of having to be isolated from humans in shielded work cells. UR was the first company to deliver commercially viable collaborative robots. This ability to act next to or with humans enables advanced robotics to help with mixed product assembly, and is transforming companies and entire industries.

The UR3 is their smallest cobot model. It is a compact table-top robot for light assembly tasks and automated workbench scenarios. It weighs only 11 kg, but has a payload of 3 kg, 360-degree rotation on all wrist joints, and infinite rotation on the end joint. Confirm has bought one for demonstration and outreach purposes, and a few small applications have been built with it using the programming techniques offered by UR3.

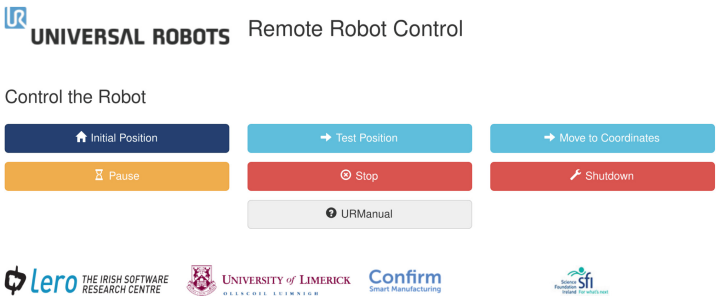

Fig. 2. UR3 web controller: the main page.

2.1 Programming the UR3

Polyscope GUI
A tablet with a graphical GUI for steering and commanding the robot is tethered to the robot. This Polyscope GUI interface is programmed using the touch screen of the tablet with URP files. Figure 1 shows the Polyscope GUI in the UR3 simulator environment: one can control the robot movements using the arrows on the left, or program waypoints by entering the coordinates on the right. It is also possible to upload scripts in the UR script language to the tablet.
UR Script Language
We used instead the UR script language. UR Script has variables, types, flow of control statements, function etc. In addition, UR Script has several built-in variables and functions which control the I/O and the movements of the robot. UR script commands can be sent from a host computer or PC via an Ethernet TCP socket connection directly to the UR robot, for motion and action control without using the tablet pendant. We use the UR simulator to execute .urp progams. At the bottom of Fig. 1 we see that the simulator screen provides buttons to go to a Home position, to perform Freedrive using the arrows, and to a Zero Position.
The RoboDK Simulator
RoboDK is a simulator and offline programming software for industrial robots. It can generate script and URP files which can be executed by a UR robot. Additionally, it is possible to execute programs on the robot from the RoboDK if the robot is connected to the computer. RoboDK can also import script files to the simulator. This allows to simulate existing script programs, modify them and re-export them.

2.2 Case Study: A Web UR3 Controller

We used the Script language and the TCP socket to program the UR3, creating a web based application that controls the robot in a very simple fashion. After having entered the IP of the robot in a first screen, Fig. 2 shows the main

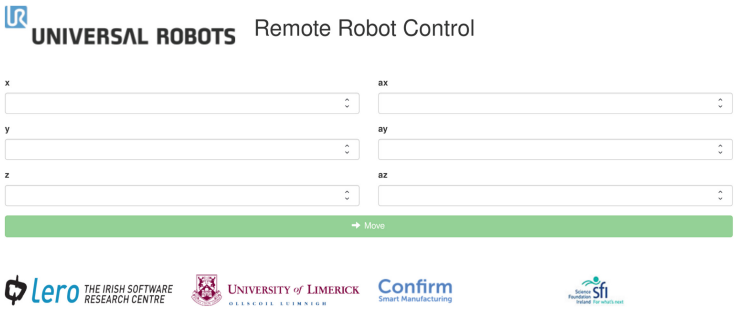

Fig. 3. UR3 web controller: the Move to Coordinates page

page of the controller: One can bring the robot to a predefined Initial Position (equivalent to the simulator's Home button), bring it to a predefined Test position (equivalent to the simulator's Zero Position), ask it to Move to Coordinates, Pause the robot, Stop it, and perform its Shutdown. All these buttons drive the simulator, or, alternatively, the real robot. The UR Manual button links to the online manual provided by UR. The only action that requires further inputs is Move to Coordinates: Fig. 3 shows the corresponding page.

As mentioned before, the Web application is connected per TCP to the Robot and the execution happens on the simulator or on the real UR3 cobot.

But how did we program the Web application?
This was started as this Summer as a small Proof of Concept demo by two interns from INSA Rouen in France, mathematics students after the 2nd year, not really familiar with programming, in particular not with Web programming, Java, nor deployment.

3 DIME as a Model Driven IDE for Robotics

The "program" that steers the UR3 controller application as well as the GUI layer for the Web are actually not hand-programmed in code at all. We see in Fig. 4 that models cover the complete web application design thanks to the use of DIME [5], the DyWA Integrated Modelling Environment [45], which is the currently most advanced and comprehensive Cinco-product [43]. DIME is a allows the integrated modelling of all the aspects needed for the design of a complete web application in terms of Graphical Domain-Specific Languages (GDSLs). Figure 4 shows that models capture the control flow as in the Service Logic Graphs of METAFrame [53,57] and jABC [36,44,61] before DIME. In DIME we have additionally also data models and UI models in the same Integrated Development Environment (IDE). In fact we see in this model:

– the **control flow**, showing that from the Start we proceed to the GetAddress page, with the symbol of a UI model, and then from there to the PublicHome page, another UI model, from which the control branches (solid arrows) lead

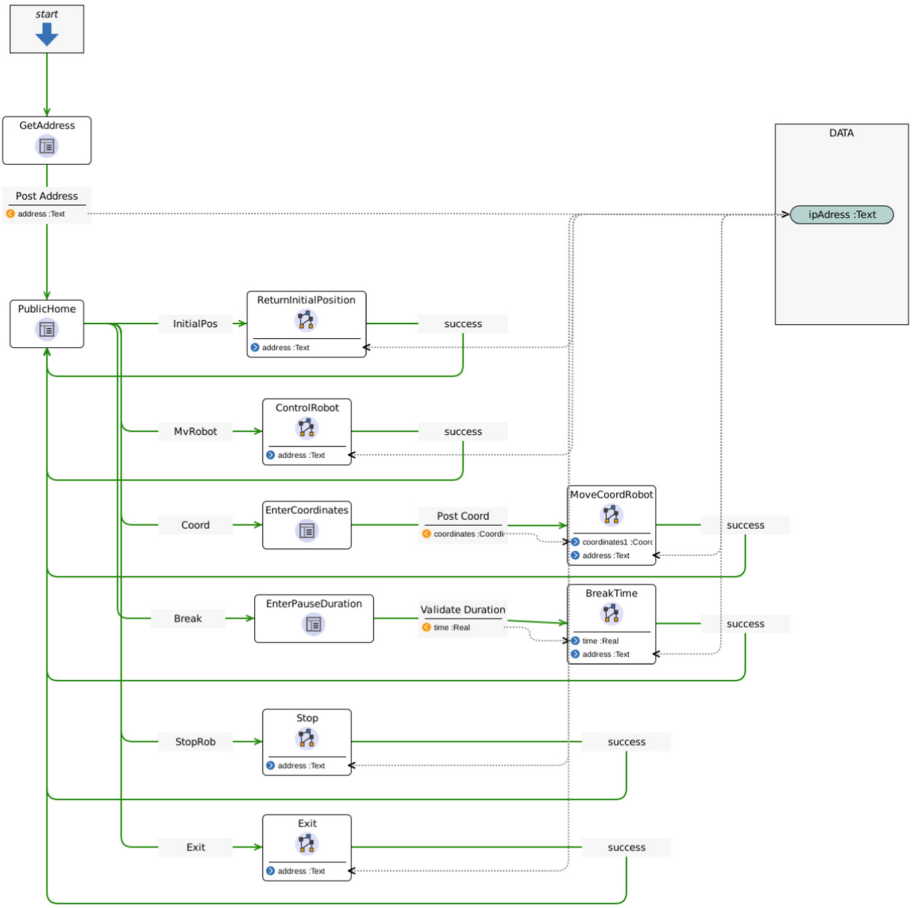

Fig. 4. The main workflow of the control application.

us to one of the successive actions: ReturnInitialPosition, ControlRobot, EnterCoordinates, EnterPauseDuration, Stop, Exit.

- the **data flow**, showing that there is a Data Context containing the data collected or used in the application. The data flow arrows are dotted, they connect the data from the output branches (like the IPaddress and their type, here text, the coordinates, and the pause time) to the data context, which models the store (i.e. the memory) and the data context to the inputs of the successive processes. For example the MoveCoordRobot process receives both the Coordinates and the IPaddress.
- the **subprocesses**, like ReturnInitialPosition, ControlRobot, which have own models,
- the **Web pages**, with GUI models, which are the only part the user experiences, as the processes work in the background.

– We also see that this is a **reactive application**: there is no End action, all the paths through the web application workflows lead back to the PublicHome main page of Fig. 2.

All these models are interdependently connected, shaping the 'one thing' [39, 62] global model in a manner which is formal, yet easy to understand and to use. The continuous and model driven deployment cycle is simplified to the extent that its code can be one-click-generated and deployed as a complete and ready to run web application. This happens along the methodology and process of [5].

3.1 Behavioural Models and Feature DSL

The power of DIME as a modelling and development tool for code-less [10] and even no-code development is connected with behavioral model-driven design as auspicated in [31] and [32]: the original DyWA was for the user a web-based definition facility for the type schema of any application domain of choice. Coupled with the defined types is the automatic generation of corresponding Create, Read, Update, Delete (CRUD) operations, so that application experts are able to model domain specific business processes which are directly executable in our modelling environment. Upon change, the prototype can be augmented or modified stepwise by acting on one or more types in the type schema, the corresponding data-objects, and the executable process models, while maintaining executability at all times. As every step is automated via a corresponding code generator, no manual coding is required.

In this case, we see that the main workflow includes various other process models, indicated by the little graph symbol. For example, the MoveCoordRobot subprocess that sends the coordinates to the robot is shown in Fig. 5. Upon start it receives the coordinates collected in the respective webpage (see Fig. 3), it prepares the UR script program that initialises the robot, instructs it to move to those coordinates, then sends the commands and shuts down the robot.

The collection of such processes is a **domain specific language (DSL)** at the **feature** [1,9,16,28] level: these process accomplish one user level intent each, which is the definition of feature. Additionally, these features are **behavioural**: the processes express operations of the UR3 (i.e., what it does) instead of static affordances (i.e., what are its components). We see in Fig. 6 the collection of subprocesses created for this application. Behavioural features are usually domain specific but application independent: we could easily reuse them to create a different demonstration with the UR3. As in DIME, we distinguish

– Basic processes, like CreateUser needed to log in and access the WebApplication. Basic processes are DIME-level processes, that deal with the essential elements of a web application, like creating users. This process has been in fact reused from a previous application, and not developed specifically for this demo.
– Interaction processes are executed client-side within the user's web browser. They define the immediate interaction between user and application and can be regarded as a site map.

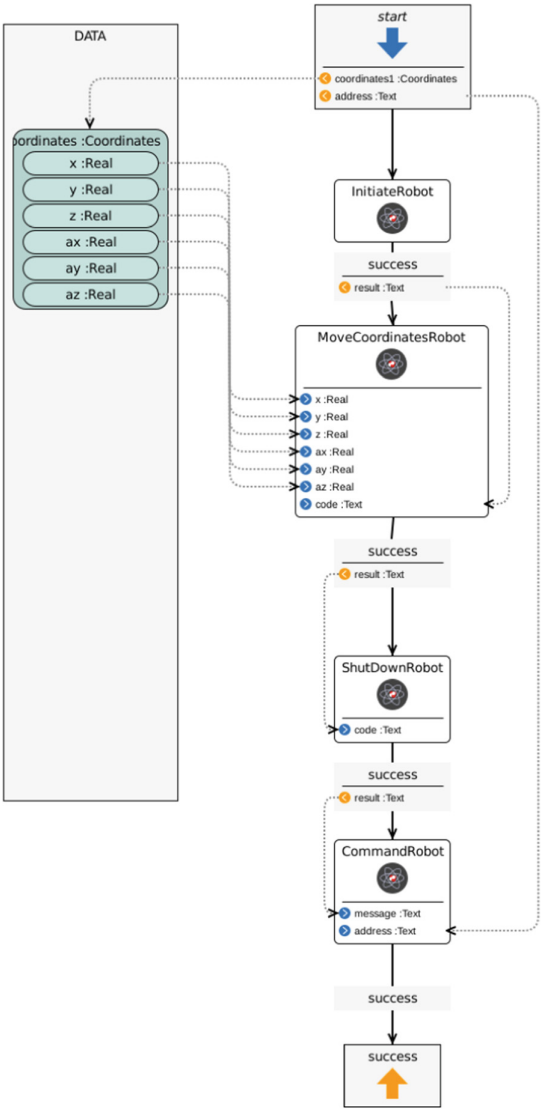

Fig. 5. Subprocess MoveCoordRobot: sends the Move command to the robot.

– Interactable processes are slightly restricted basic processes that are provided as REST services and can thus be included in interaction processes.

Not used here are additionally

– Long running processes, that describe the entire lifecycle of entities. They integrate interactions with one or multiple users as well as business logic in the form of interactable and basic processes.

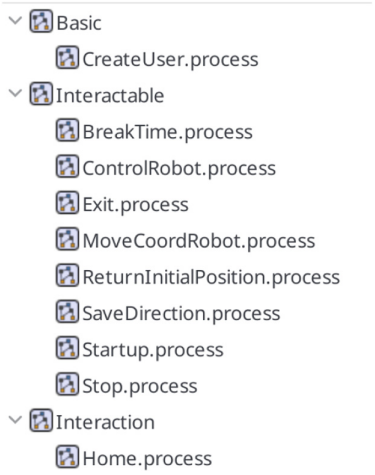

Fig. 6. Overview of the created processes: the behavioural Features DSL

– Security processes, that realise access control based on the specific user currently logged in and the association, e.g. in form of access rights, to other entities in the system.

It is also very likely that we could easily reuse the behavioural features in a demo for a different UR robot, like the larger UR5, even larger UR10, and the brand new UR16. In this particular case of the UR cobot family, they are built in a physically similar way, because they all have the same arm component types, only scaled dimensionally or made internally more robust, but they have the same functionalities and thus share the same list of behavioural capabilities. What makes a difference is the reach, own weight, maximum load, which impact the parameter (i.e., the concrete data) but not the collection of capabilities. As the workflows are parameterised in the data, they could be reused very likely as they are (Fig. 7).

3.2 The UR3 Native DSL

As we see in Fig. 4, the process models can contain other process models, hierarchically and also recursively, collected in this case in the Feature DSL of Fig. 6, but at some point processes also include atomic actions, like the InitiateRobot, MoveCoordinatesRobot, ShutdownRobot, and CommandRobot in Fig. 5. Figure 7 shows the collection of UR3-specific atomic actions (called SIBs, for service independent building blocks) used in this application. Notice the symbol of an atom, indicating their basic character, i.e. their referring to either native code, or an external API or service. This distinguishes them from the processes. While processes are models (Service Logic Graphs in our terminology) and are "implemented" by models and symbolised by a little graph icon, atomic SIBs are

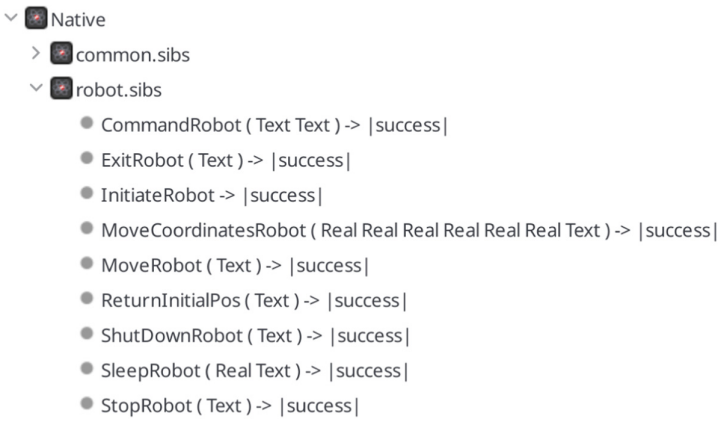

Fig. 7. Overview of the created SIBs: the native UR3 DSL

implemented by code. Therefore they are the basic units (thus "atoms") from a model point of view, and they embody and provide the link to the traditional programming level.

The native SIB palette robot.sibs encapsulates at the model level a subset of the native commands in the UR script programming language. There are more commands in the script language, and in order to use them in this or other applications they would need to be encapsulated similarly, to be included in the palette and made available to the application modellers.

Like the Feature DSL, also this DSL is reusable. While the Feature DSL is a collection of models, and thus it can change and evolve largely independently of programming artefacts in native code, the evolution of the Native DSLs depends on changes in the languages, APIs, and (service) interfaces they encapsulate. In this sense, the use of these layered DSLs provides also a very clean structure for organising and managing the domain and application evolution, which are usually in distinct areas of competence and responsibility:

- On the basis of the available Native DSLs, application designers can change, enrich and evolve their application, like our UR3 Controller demo, at pleasure without the need of programming. Modifications happen in the GUI or in the business logic, on the basis of existing native libraries and thus with the same code basis. The new workflows and GUI models are compiled to new application code and deployed without the need of any programming skills nor activity. In this sense, our XMDD approach [38,42] is a zero-code environment for this application design and evolution.
- If the UR script language changes, or an API or another interface, library, or code that are encapsulated in respective Native DSLs change, this happens in the decision and management sphere of the organisation or entity that is responsible for that platform or code. In such case it may be necessary to revisit the corresponding Native SIB DSL, and potentially take action.

In some cases, changes to implementations internal to a platform do not impact the interface. In that case the Native DSL is still valid as is. In case there are changes, they can lead to the addition, modification or deletion of operations, which may require the Native DSL manager to modify the palette accordingly. Potentially this can lead to versioning of atomic SIBs, because applications relying on the older palette may have to be retained running on the previous version. Modifications and deletions in a Native DSL may also have consequences for the applications that use them. This is however not different from the usual practice with any application relying on external code or libraries.

3.3 GUI Models and Their DSL

GUI models allow the definition of the user interface of the web application. As we see in Fig. 8 (right), they reflect the structure of the individual web pages. They can be included within the sitemap processes as an interaction point for the user, like in this case, or within other GUI models to reuse already modelled parts, as in the case of the Header GUI model at the top of the model, containing the UR logo and the top line (Remote Robot Control) that is the title of the demo, and the Footer GUI model, containing the logos of the university, Confirm, SFI etc.

On the right we see the GUI Native DSL of DIME. It is important to note that this GUI comes with DIME and is shared by all the applications, win whatever domain, built with DIME. Not only it ensures a common technological basis for all the DIME applications, it also enables application designers with no experience of GUI and web programming, like our interns, to create really nice applications without the need to learn the various server side and client side technologies, nor to have to deal with compilation and deployment stacks. In our application, the students concentrated their effort exclusively on the creation of the Native UR DSL, the conception of the application, and its design and debugging at the level of the (executable) models. If the common GUI palette is not enough there is a way to write Native Frontend Components, similar to the Java Native SIBs.

In the design, the connection between the different model types is evident in the combination of the GUI model of Fig. 8 (right) and the workflow model in Fig. 4. When composing the GUI model of the homepage, the designer drags and drops the various buttons and elements from the GUI palette onto the canvas, places them and defines their surface properties, like e.g. what's written on a button and its colour. Due to the knowledge about these models and model elements embedded in the DIME environment, for example adding a button to the page makes its GUI model add one outgoing branch with that name. As a consequence, once the designer has defined the Home GUI model in Fig. 8 (right), so that it now appears in the list of defined GUIs, and then drags it onto the Workflow canvas when composing the workflow model of Fig. 4, the symbol of the Home GUI automatically displays also the 6 outgoing branches, each labelled with the name of a button in the corresponding GUI model. This way

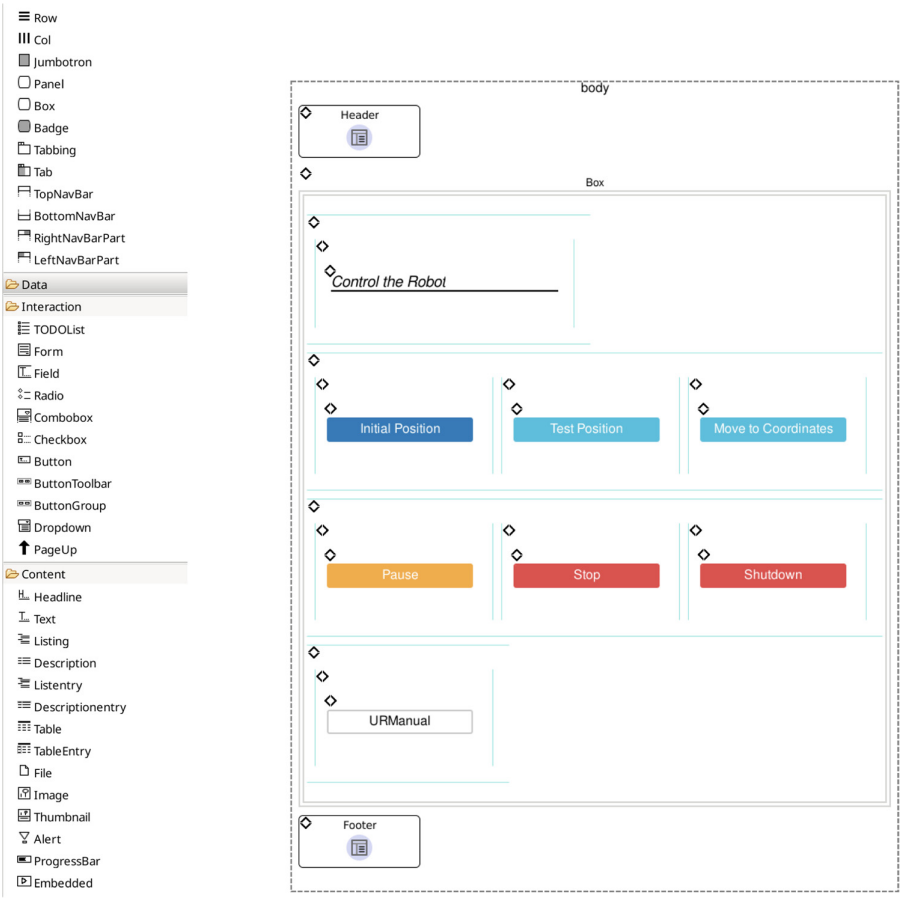

Fig. 8. The DIME GUI palette (left) and the GUI editor for the control page.

the consistency and correctness of the connections between different model types and different aspects and element of the application are enforced or supported by DIME in its nature as knowledge-infused IDE for models.

4 XMDD for the Digital Thread

The rising dependence on intersectoral knowledge and the convergence of economic sectors are being increasingly experienced in Industry 4.0. This happens internally due to the smart manufacturing model, which connects hardware, software, networks and communications, materials, business models, laws and regulations. It happens also externally, due to the manufacturing of all kinds of goods, in any sector, and the need to be more flexible in moving from one product or kind of products to another due to increased market demand fluctuations and opportunities.

As described last year in [32], referring to the Irish and global situation, a number of "needs" are mentioned over and over again:

- the need to integrate across specialisation domains, spanning across various disciplines of research and professions;
- the need to become increasingly agnostic about the specific technologies: the programming language, the operating system, the data management/information system, the communication networks, the runtime platforms, and more;
- the need to be future-ready: projects, collaborations, consortia and alliances change. No IT product can afford being locked into technological walled gardens, the need is voiced over and over again to be as technology- and as platform-independent as possible;
- the need to be able to try fast and improve fast: time to market is important, but time to test/time to retest are equally important. What is called "continuous development" or "continuous integration" needs to be supported as the new mainstream paradigm of system design and evolution.

The answer rests in the up-skilling of knowledge, at the same time trying to go to the essential bits, without lengthy and costly education paths. It seems the case that at the centre of the Smart Manufacturing movement lie the *increased use of models*, first and foremost the creation and adoption of the Digital Twin, and the *increased integration of heterogeneous models* in a Digital Thread, and this has to happen within the next few years, this will have to happen with the current labour force in the design and decision workshops. This population of selected experts is sectorally trained, sectorally competent, and unlikely to be able or willing to undergo significant retraining specifically in software.

The creation of a software or system is itself *innovation*, and the change or evolution of an existing system is innovation, too. Consistently with the widely successful school of *lean* approaches to innovation, one needs to *fail fast* and *eliminate waste*. The key is to recognise as early as possible that something is not as wished, possibly before investing time and resources into producing what will need to be amended, and make changes right away on the artefacts that are available at that stage. The systematic use of portable and reusable models, in a single paradigm, combined with generative approaches to code, has the potential to provide a Next Wave of IT that meets by and large the ambitious goals described above.

Revisiting some of the aspects mentioned in [32] after over one year of further interaction with various industries and more direct experience on what works and what is too complicated or looks too unfamiliar to be successful, we think that the approach described in the UR3 case study has a number of essential traits that make it attractive to the new public of Industry 4.0.

4.1 Innovation Directions

An efficient way to deal with this paradigm addresses a variety of aspects and innovation directions, that we briefly summarise.

The **cultural aspect** is made accessible by resorting to the description of artefacts within domain models, rather than code: the immediacy of understanding of these simple models has a chance to concretise "**what happens when**" (addressing comprehension, structure, documentation, training) and "**what happens if**" (covering simulation and prognosis) to a nearly haptic level. In discussions with engineers, they called the models and elements "blocks" of what happens, that one can assemble and construct in a virtual way but feel concrete and real. The concept of DSLs and domain specific Palettes of features and native SIBs in DIME covers this aspect in an intuitive way.

The **code aspect** is addressed by separating the (application specific or reusable) "logic" from the implementation of the operations and the system. What's native is system-side, and it encapsulates and integrates what's already there, reusing it and making it better understandable. Models, simulation environments, tools and machines, but also AI tools, visualisation tools and communication networks, IoT devices etc come with APIs amenable to the DSL transformation and integration as we just illustrated for the UR3. Here, the service oriented computing embraced by DIME and its underlying integration paradigm leverages component based design together with a high level description of the interfaces and properties of the components.

The **testing aspect** is streamlined and anticipated in the design lifecycle by choosing modelling languages that facilitate an early stage "checking" of the model structure, of the architectural and behavioural compatibilities. This is happening at the Digital Twin level with the choice of adequate simulation languages and environments. There is an interest spike for SysML due to the influence of the traditional UML community on the software engineering mainstream, but not without critics due to the heterogeneity and heavy weight approach to modelling it imposes. This is perceived as costly and bulky, so there is interest for lighter alternatives. On the software side, Architecture Analysis and Description Languages (AADLs) cover the architectural and static aspects, while the use of formal models like the KTS used in jABC [36], DIME [5] and in general graph-based models supported by Cinco [43] allow also a behavioural analysis. Early analysis capabilities, e.g. by model checking on the kind of models DIME provides, and in some cases a correct-by-construction synthesis of property-conform models seem to be useful to deliver the "speed" of early detection and even avoidance of errors that makes the successive testing on the (mostly generated) code much faster.

The **dissemination and adoption aspect** is going to be taken care of by sharing such models, for example within the Confirm community of practice. Such models are understandable to the domain experts, and to a good extent (at the level of the processes and features) independent of the specific technology of a vendor. In several cases it would be possible to share also the implementations of the building blocks, at the level of Native DSLs, using for instance Open

Source Software facilities and structures [64]. Libraries of services have been in use in the telecommunication domain since the '80s [59], they are increasingly in use in bioinformatics, geo-information systems, and are slowly taking a center stage attention also in the advanced manufacturing community, albeit mostly still in form of reference architectures and shared component models for which standards need to be developed.

The **speed to change** is accelerated by using generative approaches that transform the models into lower level descriptions, possibly into code. The speed and quality advantage of this demo in comparison with a handcrafted programmed version was immediately evident to all the non-software engineers who saw it. The core engine for this is the generative approach from models to code, and - also an aspect that arose by itself - to re-targetable code. In jABC, Cinco and DIME we use both model-to-model and model-to-code transformations, starting from the Genesys approach of [19] and [17], up to the generalized approach that Cinco adopts also at the tool metalevel [43].

The **rich description** of the single components, data, and applications is achieved by means of both domain-independent and domain-specific knowledge about the functionalities, the data and business objects, and the application's requirements and quality profile, as in language-oriented programming [11,66] or language-driven engineering [51]. The variety of vendor-specific robot programming languages, the individual APIs coming with sensors, actuators and IoT devices, in spite of standardisation efforts, are a well known obstacle to interoperability. The ease of mapping from abstract DSLs to native DSLs, via code or also via smaller adapter processes that manage the data and protocol adaptation, is possible and already experienced in DIME. It could bring here a significant relief.

The **scalability and speed of education** are supported by teaching domain specialists to deal with these domain specific models, their analysis and composition, and the validation using tools that exploit the domain-specific and contextual knowledge to detect the suitability or not of the current version of the application's models for solving a certain problem in a certain (regulatory, technological, economic) context. We have had successes in the context of school pupils, postgraduate students with a background in biology and geography [21]. We are now creating an online course that will provide a gentle introduction to these models and a number of e-tivities so that professionals and students alike will be able to self-pace and gain experience in the use of these models.

The **quick evolution** is delivered by means of integrated design environments that support the collaboration of all the professional profiles and stakeholders on the same set of models and descriptions, as in the XMDD [38,42] and One Thing Approach [39,62], applied to models of systems and of test cases [47].

The **structuring approaches** are based e.g. on hierarchy [54,55,58], on several notions of features like in [7,16,20,56], or contracts for abstraction and compositionality as in [14]. These structures allow a nice and incremental factoring of well characterised system components.This factoring aides the hierarchical

and collaborative organisation of complex or large systems, while supporting the intuition of the domain experts. It also supports finding, maybe in approximations, the most opportune units of reuse (for development, evolution and testing), and units of localised responsibility, e.g. for maintenance, evolution and support.

4.2 Experiential Evidence so Far

We used DIME, Cinco or its predecessors in a large number of industry projects, research projects, and educational settings. Various aspects of the overall picture needed in a suitable and mature integrated modelling environment for the digital thread have been already exercised in those context. They focussed mostly on a single or a small number of aspects, due to the focus of the respective project, while a Confirm digital Thread platform would be required to match and surpass them all. Still, it is useful to summarise what we know that we can do.

The acceptance of formality as a means to clarify interfaces, behaviours and properties and as a precondition to verify and prevent, instead of implementing, testing, and then repairing case by case, is essential to meet the challenges and demands for the future platforms of IT provision and use. In this sense, we need both Archimedean points [63] and future oriented knowledge management for change management [31] for the fundamental attention to usability by non-IT specialists. With this work, we intend to leverage the XMDD approach [38,42] and the previous work on evolution-oriented software engineering, under the aspect of simplicity [41] and continuous systems engineering [40] especially from the perspective of reconciling domain experts and software professionals [30]. Models with a formal underpinning are likely to be central to the wish of many manufacturers to be able to reconfigure production, which demands simplicity and predictability in evolution and reconfiguration.

We build upon over a decade of previous experiences gathered in various application domains. Specifically, our own work in scientific workflows summarized in [27] included experiences gathered from the initial projects in 2006 [33] to building platforms for the access to complex genetic data manipulations in the bioinformatics domain (the Bio-jETI platform of [23,34] and the agile Gene-Fisher-P [24]). These platforms, and other similar experiences in geoinformation systems, have proven that we are able to create platforms that virtualise and render interoperable collections of third-party services and tools that were not designed for interoperability. This happened with essentially the technology exemplified in the tiny UR3 remote controller. We hope that this previous experience may raise in manufacturing designers and producers the confidence that such a platform is indeed feasible also for the Digital Thread.

The tools we intend to use span from the Cinco-products [43] DIME and DyWA to the most work on DSLs for decision services [12]. The availability of well characterised service collections makes a semantic web-like approach feasible for these DSLs: their classification in terms of taxonomies and labelled properties brings semantics and semantics-based composition within reach. In these respects, we have a long experience of modelling, integration, and synthesis [35,37,60] up to entire synthesis-based platforms and applications [25,26]

as well as benchmark generation platforms [52]. The domain specific knowledge captured in native DSLs and their taxonomic classification as well as the feature-level DSLs and collections of properties may in fact lead to synthesizable workflows and processes within the Digital Thread.

We have experience of various tool integration techniques [29] and service integration platforms, and in the Semantic Web Challenge [48,49]. This past experience has taught that various technology choices in the interfaces, design choices in and degrees of uniformity in the structure and presentation of APIs make a huge difference in the degree of automation of producing native DSLs from such native libraries and APIs. Here we expect unfortunately a large manual effort, as from the current experience with the IoT devices, sensors and actuators we expect to need bespoke solutions for each device, manufacturer, and constant changes across a product's evolution. Interfaces and APIs design appear today in fact almost accidental, they are at the moment one of the most severely undermanaged assets in the Industry 4.0 landscape.

We used features for a long time to model variability [20], introduced various categories of constraints to define structural and behavioural aspects of variability [18] and provided constraint-driven safe service customization adapting features to various contexts [6], up to higher order processes in [46]. On the effects of knowledge on testing, we addressed specifically efficient regression testing supported by models [15] and the hybrid test of web applications with various tools and approaches [50]. We used many techniques that leverage the mathematical nature of the models in order to prove the correctness of the dataflow [22], the control flow [3], the use of games to enhance diagnosis capabilities in model-driven verification [4]. We expect these techniques to be applicable also to the Digital Twin integration in a Digital Thread, and we expect that their impact on shorter design times (i.e., quick prototyping and validation of a new application) and increased quality (i.e. less testing) will lead to a significant speed up in comparison with the current patchwork of ad-hoc code.

We are convinced that all these abilities will be essential for the realisation of an integrated, efficient and correct Digital Thread for the connected and evolving industrial critical systems of tomorrow, to achieve true *Sustainable Computing in Continuous Engineering*[1] platforms for Industry 4.0.

Acknowledgments. Thanks are due to Romain Poussin and Jean-Baptiste Chanier, who implemented the Proof of Concept of the UR3 Controller demo.

This work was supported, in part, by Science Foundation Ireland grant 16/RC/3918 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Confirm, the Smart Manufacturing SFI Research Centre (www.confirm.ie).

[1] For SCCE see https://scce.info.

References

1. Asirelli, P., ter Beek, M.H., Gnesi, S., Fantechi, A.: Formal description of variability in product families. In: 15th International Software Product Line Conference (SPLC 2011), pp. 130–139 (2011)
2. Bacchiega, G.: Creating an embedded digital twin: monitor, understand and predict device health failure. In: Inn4mech - Mechatronics and Industry 4.0 Conference Presentation (2018). https://irsweb.it/pdf/Embedded_Digital%20Twin_v2.pdf
3. Bakera, M., Margaria, T., Renner, C., Steffen, B.: Verification, diagnosis and adaptation: tool-supported enhancement of the model-driven verification process. In: Revue des Nouvelles Technologies de l'Information (RNTI-SM-1), pp. 85–98, December 2007
4. Bakera, M., Margaria, T., Renner, C., Steffen, B.: Tool-supported enhancement of diagnosis in model-driven verification. Innov. Syst. Softw. Eng. **5**, 211–228 (2009). https://doi.org/10.1007/s11334-009-0091-6
5. Boßelmann, S., et al.: DIME: a programming-less modeling environment for web applications. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9953, pp. 809–832. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_60
6. Braun, V., Margaria, T., Steffen, B., Yoo, H., Rychly, T.: Safe service customization. In: Intelligent Network Workshop, IN 1997, vol. 2, p. 4. IEEE, May 1997
7. Buckley, J., Rosik, J., Herold, S., Wasala, A., Botterweck, G., Exton, C.: FLINTS: a tool for architectural-level modeling of features in software systems. In: Proceedings of the 10th European Conference on Software Architecture Workshops, ECSAW 2016, pp. 14:1–14:7. ACM, New York (2016). https://doi.org/10.1145/2993412.3003390
8. Chhetri, M.B., Krishnaswamy, S., Loke, S.W.: Smart virtual counterparts for learning communities. In: Bussler, C., et al. (eds.) Web Information Systems - WISE 2004 Workshops WISE 2004. LNCS, vol. 3307, pp. 125–134. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30481-4_12
9. Classen, A., Heymans, P., Schobbens, P.Y., Legay, A., Raskin, J.F.: Model checking lots of systems: efficient verification of temporal properties in software product lines. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, pp. 335–344. ACM, New York (2010). https://doi.org/10.1145/1806799.1806850
10. Codeless, Platforms: Codeless platforms homepage. Technical report, ex Orbis Software (2019). https://www.codelessplatforms.com
11. Dmitriev, S.: Language oriented programming: the next programming paradigm. JetBrains onBoard Online Magazine(2004). http://www.onboard.jetbrains.com/is1/articles/04/10/lop/
12. Gossen, F., Margaria, T., Murtovi, A., Naujokat, S., Steffen, B.: DSLs for decision services: a tutorial introduction to language-driven engineering. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11244, pp. 546–564. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03418-4_33
13. Gould, L.S.: What are digital twins and digital threads? (2018)
14. Graf, S., Quinton, S., Girault, A., Gössler, G.: Building correct cyber-physical systems: why we need a multiview contract theory. In: Howar, F., Barnat, J. (eds.) FMICS 2018. LNCS, vol. 11119, pp. 19–31. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00244-2_2
15. Hagerer, A., Margaria, T., Niese, O., Steffen, B., Brune, G., Ide, H.D.: Efficient regression testing of CTI-systems: testing a complex call-center solution. Ann. Rev. Commun. Int. Eng. Consortium (IEC) **55**, 1033–1040 (2001)

16. Jonsson, B., Margaria, T., Naeser, G., Nyström, J., Steffen, B.: Incremental requirement specification for evolving systems. Nordic J. Comput. **8**, 65–87 (2001). http://dl.acm.org/citation.cfm?id=774194.774199
17. Jörges, S.: Construction and Evolution of Code Generators. LNCS, vol. 7747. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36127-2
18. Jörges, S., Lamprecht, A.L., Margaria, T., Schaefer, I., Steffen, B.: A constraint-based variability modeling framework. Int. J. Softw. Tools Technol. Transf. (STTT) **14**(5), 511–530 (2012)
19. Jörges, S., Margaria, T., Steffen, B.: Genesys: service-oriented construction of property conform code generators. Innov. Syst. Softw. Eng. **4**(4), 361–384 (2008)
20. Karusseit, M., Margaria, T.: Feature-based modelling of a complex, online-reconfigurable decision support service. Electron. Notes Theor. Comput. Sci. **157**(2), 101–118 (2006). http://www.sciencedirect.com/science/article/pii/S1571066106002489
21. Lamprecht, A.-L., Margaria, T. (eds.): Process Design for Natural Scientists. CCIS, vol. 500. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45006-2
22. Lamprecht, A.-L., Margaria, T., Steffen, B.: Data-flow analysis as model checking within the jABC. In: Mycroft, A., Zeller, A. (eds.) CC 2006. LNCS, vol. 3923, pp. 101–104. Springer, Heidelberg (2006). https://doi.org/10.1007/11688839_9
23. Lamprecht, A.L., Margaria, T., Steffen, B.: Bio-jETI: a framework for semantics-based service composition. BMC Bioinf. **10**(Suppl. 10), S8 (2009)
24. Lamprecht, A.L., et al.: variations of GeneFisher as processes in Bio-jETI. BMC Bioinf. **9**(Suppl. 4), S13 (2008). http://www.ncbi.nlm.nih.gov/pubmed/18460174
25. Lamprecht, A.L., Naujokat, S., Margaria, T., Steffen, B.: Synthesis-based loose programming. In: Proceedings of the 7th International Conference on the Quality of Information and Communications Technology (QUATIC 2010), Porto, Portugal, pp. 262–267. IEEE, September 2010
26. Lamprecht, A.L., Naujokat, S., Margaria, T., Steffen, B.: Semantics-based composition of EMBOSS services. J. Biomed. Seman. **2**(Suppl. 1), S5 (2011). http://www.jbiomedsem.com/content/2/S1/S5
27. Lamprecht, A., Steffen, B., Margaria, T.: Scientific workflows with the jABC framework - a review after a decade in the field. STTT **18**(6), 629–651 (2016). https://doi.org/10.1007/s10009-016-0427-0
28. Margaria, T.: Components, features, and agents in the ABC. In: Ryan, M.D., Meyer, J.-J.C., Ehrich, H.-D. (eds.) Objects, Agents, and Features. LNCS, vol. 2975, pp. 154–174. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-25930-5_10
29. Margaria, T.: Web services-based tool-integration in the ETI platform. Softw. Syst. Model. **4**(2), 141–156 (2005). https://doi.org/10.1007/s10270-004-0072-z
30. Margaria, T.: Service is in the eyes of the beholder. IEEE Comput. **40**(11), 33–37 (2007)
31. Margaria, T.: Knowledge management for inclusive system evolution. In: Steffen, B. (ed.) Transactions on Foundations for Mastering Change I. LNCS, vol. 9960, pp. 7–21. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46508-1_2
32. Margaria, T.: Generative model driven design for agile system design and evolution: a tale of two worlds. In: Howar, F., Barnat, J. (eds.) FMICS 2018. LNCS, vol. 11119, pp. 3–18. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00244-2_1
33. 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), pp. 169–176. IEEE Computer Society, Los Alamitos, August 2006

34. Margaria, T., Kubczak, C., Steffen, B.: Bio-jETI: a service integration, design, and provisioning platform for orchestrated bioinformatics processes. BMC Bioinf. **9**(Suppl 4), S12 (2008)
35. 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
36. Margaria, T., Steffen, B.: Lightweight coarse-grained coordination: a scalable system-level approach. Softw. Tools Technol. Transf. **5**(2–3), 107–123 (2004)
37. Margaria, T., Steffen, B.: LTL-guided planning: revisiting automatic tool composition in ETI. In: Proceedings of the 31st Annual IEEE/NASA Software Engineering Workshop (SEW 2007), Columbia, MD, USA, pp. 214–226. IEEE Computer Society (2007). http://portal.acm.org/citation.cfm?id=1338445.1338873&coll=GUIDE&dl=GUIDE
38. Margaria, T., Steffen, B.: Agile IT: thinking in user-centric models. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 490–502. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88479-8_35
39. Margaria, T., Steffen, B.: Business process modelling in the jABC: the one-thing-approach. In: Cardoso, J., van der Aalst, W. (eds.) Handbook of Research on Business Process Modeling. IGI Global (2009)
40. Margaria, T., Steffen, B.: Continuous model-driven engineering. IEEE Comput. **42**(10), 106–109 (2009)
41. Margaria, T., Steffen, B.: Simplicity as a driver for agile innovation. Computer **43**(6), 90–92 (2010)
42. 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
43. Naujokat, S., Lybecait, M., Kopetzki, D., Steffen, B.: CINCO: a simplicity-driven approach to full generation of domain-specific graphical modeling tools. Softw. Tools Technol. Transf. **20**(2), 1–28 (2017)
44. Naujokat, S., Neubauer, J., Lamprecht, A.L., Steffen, B., Jörges, S., Margaria, T.: Simplicity-first model-based plug-in development. Softw. Pract. Exp. **44**(3), 277–297 (2013)
45. 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
46. Neubauer, J., Steffen, B., Margaria, T.: Higher-order process modeling: product-lining, variability modeling and beyond. Electron. Proc. Theor. Comput. Sci. **129**, 259–283 (2013)
47. Niese, O., Steffen, B., Margaria, T., Hagerer, A., Brune, G., Ide, H.-D.: Library-based design and consistency checking of system-level industrial test cases. In: Hussmann, H. (ed.) FASE 2001. LNCS, vol. 2029, pp. 233–248. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45314-8_17
48. Petrie, C., Küster, U., Margaria, T., Zaremba, M., Lausen, H., Komazec, S.: Status, perspectives, and lessons learned. In: Semantic Web Services Challenge, Results from the First Year, pp. 275–284 (2009). https://doi.org/10.1007/978-0-387-72496-6_17
49. Petrie, C., Margaria, T., Lausen, H., Zaremba, M. (eds.): Semantic Web Services Challenge. Results from the First Year, Semantic Web and Beyond, vol. 8. Springer, US (2009)

50. Raffelt, H., Margaria, T., Steffen, B., Merten, M.: Hybrid test of web applications with webtest. In: TAV-WEB '08: Proceedings of the 2008 Workshop on Testing, Analysis, and Verification of Web Services and Applications, pp. 1–7. ACM, New York (2008)
51. 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: State of the Art and Perspectives. LNCS, vol. 10000, pp. 311–344. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91908-9_17
52. Steffen, B., Isberner, M., Naujokat, S., Margaria, T., Geske, M.: Property-driven benchmark generation: synthesizing programs of realistic structure. Softw. Tools Technol. Transf. **16**(5), 465–479 (2014)
53. 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
54. Steffen, B., Margaria, T., Braun, V., Kalt, N.: Hierarchical service definition. Ann. Rev. Commun. ACM **51**, 847–856 (1997)
55. Steffen, B., Margaria, T., Claßen, A.: Heterogeneous analysis and verification for distributed systems. Softw. Concepts Tools **17**(1), 13–25 (1996)
56. Steffen, B., Margaria, T., Claßen, A., Braun, V.: Incremental formalization: a key to industrial success. Softw. Concepts Tools **17**(2), 78–95 (1996)
57. Steffen, B., Margaria, T., Claßen, A., Braun, V.: The METAFrame'95 environment. In: CAV, pp. 450–453 (1996)
58. Steffen, B., Margaria, T., Claßen, A., Braun, V., Nisius, R., Reitenspieß, M.: A constraint-oriented service creation environment. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 418–421. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61042-1_63
59. Steffen, B., Margaria, T., Claßen, A., Braun, V., Reitenspieß, M.: An environment for the creation of intelligent network services. In: Intelligent Networks: IN/AIN Technologies, Operations, Services and Applications - A Comprehensive Report, pp. 287–300. IEC: International Engineering Consortium (1996)
60. Steffen, B., Margaria, T., Freitag, B.: Module Configuration by Minimal Model Construction. Technical report, Fakultät für Mathematik und Informatik, Universität Passau (1993)
61. 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
62. Steffen, B., Narayan, P.: Full life-cycle support for end-to-end processes. IEEE Comput. **40**(11), 64–73 (2007)
63. Steffen, B., Naujokat, S.: Archimedean points: the essence for mastering change. LNCS Trans. Found. Mastering Change (FoMaC) **1**(1), 22–46 (2016)
64. Steinmacher, I., Robles, G., Fitzgerald, B., Wasserman, A.I.: Free and open source software development: the end of the teenage years. J. Internet Serv. Appl. **8**(1), 17:1–17:4 (2017). https://doi.org/10.1186/s13174-017-0069-9
65. Talkhestani, B.A., Jung, T., Lindemann, B., et al.: An architecture of an intelligent digital twin in a cyber-physical production system. Automatisierungstechnik **67**(9), 762–782. 101515/auto-2019-0039 2019
66. Ward, M.P.: Language oriented programming. Softw. Concepts Tools **15**(4), 147–161 (1994)

67. Wikipedia: Digital twin - including 10 definitions. https://en.wikipedia.org/wiki/Digital_twin
68. Wortmann, N., Michel, M., Naujokat, S.: A fully model-based approach to software development for industrial centrifuges. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9953, pp. 774–783. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3_58