Generative Model Driven Design for Agile System Design and Evolution: A Tale of Two Worlds

Tiziana Margaria(✉)

Chair of Software Systems, University of Limerick, and Lero, Limerick, Ireland
tiziana.margaria@ul.ie

Abstract. In order to mainstream the production and evolution of IT at the levels of speed, scale, affordability and collaborative effort needed to truly make IT enter the fabric of every economical and societal endeavour, as is the projected future of our society in the next decade, the ease of learning, understanding, and applying new disruptive technologies must drastically improve. We argue that the needs of the people, the economical sectors, and the large-scale trends can only be met if the IT professions embrace and adopt a new way of producing and consuming IT, based on more formal descriptions, more models, more reasoning and analysis before expensive implementations are incurred, coupled with automatic transformations, generations, and analyses that take advantage of the models and formalized knowledge.

We analyse briefly the various dimensions, derive a specification for the new IT and IT platforms, and provide a few examples of how the new thinking can disrupt the status quo but empower a better understanding, a more efficient organization, and a more automatic management of the many cross-dimensional issues that future connected software and systems will depend upon.

1 Introduction

In the increasingly connected and heterogeneous world in which the modern and future industrial critical systems will operate, agility and evolution are of paramount importance. As in Alice in Wonderland, solution providers and technology providers need to run fast in the technology and context evolution race in order to not fall back. The steady evolution of products, infrastructure, as well as design and implementation/manufacturing environments is a continuous source of change. It is additionally topped by disruptions: examples are the inception and then the steady adoption of online-X, self-X and now smart-X approaches across the economy sectors. These disruptions are pervasive and irreversible trends that subvert, one after the other, the well established power and dominance structures in the sectoral, local and global economy. This happens over and over again: in little more than a decade we have seen the inception and then mainstreaming of online communication, marketing, service, and commerce

F. Howar and J. Barnat (Eds.): FMICS 2018, LNCS 11119, pp. 3–18, 2018.
https://doi.org/10.1007/978-3-030-00244-2_1

channels, to the point that Amazon has been for years among the most valuable companies worldwide. The self-X economy has eliminated many service professions though the adoption of online access and individual recognition of the user: travel agencies, ticketing services, booking platforms, and the corresponding service desks of the large providers (like airlines, railways, and event managers like ticket.com or eventbrite), or in the case of science and research, online paper submission systems and online conference management systems. The smart-X economy is the new incipient wave, fuelled by data collection and analysis readily available in a connected and cloud based fashion. The "Smart Anything Everywhere"[1] paradigm builds upon increasingly cheap storage, increasingly cheap and efficient computation power, increasingly powerful and pervasive communication networks, and various advances in traditional algorithms for analysis and optimization, and now also AI/ML style reasoning. Clearly, all these changes separate the current world, as we know it, from a future world where all these connections and enhancements will be accepted, considered normal, and essential part of the established "business practices" for all organisations and companies.

So in such an accelerating, convergent, and individualized socio-technical and economical world, what kind of design and implementation technologies for software and systems will be needed in the future to survive and possibly thrive?

The specification for the fundamental traits of a new generation of technologies comes from the kind of changes demanded by customers and users (at the individual and corporate level). These are either enforced by the economic actors (like the providers of components, systems, the integrators, and the various granularities of cooperation practiced in complex and global supply chains), or mandated by those entities that are responsible for policies and regulations (like the EU in GDPR, the governments, or oversight and standardization bodies, etc.).

So let us have a look at these dimensions of change and forces (Sects. 2–4), in order to derive a characterization of what needs to fundamentally become different if we wish to be ready for the new roaring twenties ahead (2020–2030) in Sect. 5 and propose a new paradigm in Sect. 6.

This space is characterized by a large prevalence of contradictory "needs" and desires, that can only be faced by thinking in a fundamentally different way. Putting a new thinking into practice requires a welcoming adoption of innovations, in spite of the fact that innovations are by definition new unproven paths and means, and as such scary and risky. The lines of resistance to innovation are accordingly high-profile and deeply rooted in the individual fear and organisational inertia. The specific micro- to macrolevel contexts I am prevalently looking at and from which I draw the observations are those of Lero and Confirm, two Irish national research centres that include 8 resp. 10 universities and research institutions, and over 40 companies (SME to multinationals) each. Such centres run 6 years research programs comprising tens of projects, and are embedded in the various layers of decision and management at the single partner level, centre level, national level and EU/international/global level. The global level is due

[1] See the EU initiative at https://smartanythingeverywhere.eu.

to strategic partnerships, e.g. with Fraunhofer in Germany, NII in Japan, and CSIRO in Australia and the practice of international collaborations, but also to the fact that several industry partners are Irish branches of multinational corporations.

Examples of the new, future-oriented way of dealing with complex interfaces, new architecture paradigms and security-injecting generative approaches are then provided in Sect. 7.

2 The People

Looking at users as consumers of technology and products, the demand is for significant change already in the close and even immediate future: how can people achieve more complex, more specialized and technically advanced goals and operations by using "simpler" and more supportive tools and infrastructure? In other words, by using tools and systems that "know" more themselves, humans with less and less technical mastery should be quickly enabled to achieve in a more reliable way and at higher quality a wealth of more complex, more precise, more secure and higher quality design, implementation, production, maintenance goals. At the same time, the need for engineers and specialists is growing much faster than their production along the established paths. Along the universities and the universities of applied science, here I see as "producers" of IT specialists also the technical courses in high schools for those countries like e.g. Italy and Germany that have a rich vocational school offering[2].

The requests for producing more specialists and faster are increasingly insistent. These requests come from companies and from professional organizations, both nationally, e.g. from the Irish Computer Society, and internationally, from ACM and IEEE. They also come from the governments and their proxies, like the Irish Higher Education Authority, that design and enforce the policies and programs for the workforce of the future. They ask Universities and research centres to train more people with a large variety of non-traditional backgrounds in these new and wonder-achieving technologies. They dream of an educational system that (a) in a short time span of typically a few weeks to a few months, (b) possibly without direct contact with the teachers nor in-person monitoring (i.e. through online education, or blended forms whereby the on-site presence is extremely compact - one weekend to one week per academic year), and (c) largely virtually, i.e. surely without the need of daily physical presence in dedicated equipped spaces (like classrooms and laboratories), these "fast tracked" individuals become proficient professionals in the new technologies, and future-proof employees of these advanced, leading edge companies.

[2] In Italy there is a rich tradition of excellent Istituti Tecnici that form at thousands of qualified experts at the upper secondary school level, leading to chartered professional profiles (e.g. Perito tecnico) under the control of professional Charters (Albo Professionale). The German system has a strong tradition of dual education (duale Berufsausbildung) combining formal education with a training on the job component.

The characteristics and slogans we hear in this context span typically *agile workforce training; training on demand; workforce evolution; off-site approach; flexible, adaptable and smart-sized education; ad-hoc education pills; life-long upskilling; competence building; capability-oriented approaches.*

3 The Economic Domains and Their Convergence

Referring to the Irish and global situation, a number of "needs" are mentioned over and over again:

- the need to integrate across specialization 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.

Such demands are brought up consistently across all the economic domains: when we talk about large scale software development and global software development as in Lero or at the EU level, when we address smart advanced manufacturing as in the Irish Confirm or Industry 4.0 in Germany, or smart bio and smart energy (as in the MAREI centre in Ireland), or smart agri (as in an ongoing EU initiative), or new smart materials (as in the Bernal Institute at UL and the ADAPT research centre in Ireland). This uniformity indicates very clearly the convergence of these domains not only in the factual collaboration (e.g. integration to embed smart energy aspects in smart manufacturing and smart agri) but also a higher-level, strategic convergence when anticipating and forecasting what capabilities will be essential to thrive in the next decade.

The characteristics and slogans we hear in this context span *agile development; continuous quality control; evolution-driven design; seamless integration; seamless evolution; continuous development; continuous integration, data-driven development; technology and vendor lock-in.*

4 The Big Picture Context

At a higher abstraction level, the strategists think in terms of paradigms, megatrends, and metaforces that guide the 5 to 10 years cycle length of the market

movement and technology adoptions. The moves to *connected*, *mobile*, *online*, *individual*, *social*, *green* and now also *smart* fall in this category of strategic, long term and large scale concerns. These cycles are slower, and as such they are both slower to introduce and also slower to pace out. Some consider them to be generation-defining: they lock into the fundamental imprinting and belief system an individual human feels comfortable with. Therefore they are almost impossible to "undo" or to transition to the "next generation values". The *digital natives* vs. *digital immigrants* distinction was an example of this generation-gap defining skill at the turn of the century, and the internet (i.e. the pervasiveness of online, mobile) followed short thereafter. The problem with the big picture is that it suffers disruption too. In a low frequency, long cycle system, a wave of disruption can be even more subversive than in the "normal" tech world. Examples of such fundamental discontinuities have been the oil crisis in the '70s, the internet bubble burst in the '00s, the 9/11-induced shock, freeze and consolidation in a number of markets, like e.g. aviation. Now, we face the potential de-globalization of supply chains following the trade changes due to e.g. Brexit, the current uncertainty concerning trade agreements like the TPP, NAFTA or the Iran deal, and the quick and unexpected resurgence of tariffs.

What we hear in this context sounds like *survival of the fittest is survival of the fastest; agility and quick evolution; flexible contracts; globalization and de-globalization; reframing of supply chains; glocalization.*

5 The Tools and Techniques for the New World

In the new world, the answer in terms of which IT tool and techniques should be researched, produced, and then studied, adopted, and taught today to the traditional and non-traditional students and professionals in order for them to be future-ready needs to capture the essence of these "keywords", which can be summarized in *speed, uncertainty, and change.* Accordingly, the new IT needs to deliver a correspondingly updated, simplified and flexible approach to producing software and systems. It also needs to provide a much simplified and technology-shielding software infrastructure and platform for software and system development itself. This need induces a significant disruption from the past and current culture, where IT production was and is in the hands of (trained or self-taught) specialists who (need to) master coding and (need to) know the details of development systems, programming frameworks, operating systems, communication systems, virtualization systems down to the hardware. In the new world, there is no time and no long-term value anymore in mandating all this knowledge from whoever professionally uses IT and systems and produces IT and systems. So we face deep change, disruptive of how we teach and educate.

As long as we as a community keep practicing and teaching a **code-driven** and **test-driven** approach to (complex) system design, understanding what a system does based on source code will remain a challenge. It is a challenge already now for professionals, as witnessed for meanwhile over 50 years at general software engineering conferences like ICSE and OOPSLA as well at conferences

and workshops on software testing, software quality and software maintenance. Works like [5] show that even recovering just the feature level architecture of existing software systems is hard work: it requires both specialized tools and quite some detective work, coupled with technical and domain knowledge, affinity for experimentation and a good pinch of intuition. Few people, even in Software Engineering, satisfy this profile. The mass of people and professionals in need of a system's refactoring, extension, and integration described as "the people" in Sect. 2 isn't equipped to add all these skills to their professional portfolio. The educational system currently for IT is not made either to mainstream deep technology competences. The outcome is that for most IT change projects the need is recognized but there is no follow up. This freeze into past solutions locks entire companies and organisation into outdated systems that impose outdated working patterns. This status quo counters the increasingly flexible workflows and collaborations mandated by both the collaborative economy and the many disruptive opportunities, trends and threats.

The problem is not solved by the run to open source: even if the source code is open, and in theory reusable and modifiable, it still requires comprehension first, then mapping to the concrete situation, and thus all the capabilities mentioned above: it is in practice like starting with legacy. The big boost to adoption of open source has been extremely beneficial for the sharing economy, the commoditization of essential layers of software, the globalisation and mass accessibility of technology and its related knowledge in ways unthinkable even 20 years ago, when entire domains were in the hands of proprietary systems with closed APIs. So, OSS and FOSS [37] work very well, but again they are for specialists.

The Agile movement has sparked a new attitude towards collaborative and fast paced system development, including frequent and consensual meetings with stakeholders that are users and customers, and it is gladly adopted in industry. However, agile thinking addresses the software development process and is agnostic to the means and artefacts. In practice, software is developed again within the "standard best practice": starting from code and with "test first" as a maxime for quality assurance. What this means is twofold:

- Because it is **code**, the artefacts are out of the cultural reach for most of these users and customers, as well as for most of the professional roles essential to the software business that are not directly code-related: sales, marketing, legal, etc. These people have to rely on expert "translation" from the artefacts to the understandable (but possibly incomplete, ambiguous, biased) interpretation of those artefacts and their characteristics into some IT-layman description, that most often is in some natural language prose. We are still trapped here in the **business-IT translation gap**.
- Because quality assurance is prevalently if not exclusively handled via **testing**, it can take place only post-factum: a system is implemented to the code level, and only then it is amenable to validation, verification etc. Given that it is very expensive to write code as well as to debug and test code, this is an inherently and **systematically wasteful** way of managing the production of

software and systems. Also simulation based approaches still require writing simulation code that is quite akin to the production code. Also the verification is still conducted case by case, configuration by configuration, run by run. Once a system is out, any change requires reassessment through regression testing, possibly recertification, skewing even more toward the testing costs the already onerous ratio of cost to develop an increment vs. cost to test that increment.

6 Generative Approaches as the Next Wave

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 recognize 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.

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

- The **cultural aspect** is made accessible by resorting to the description of artefacts within domain models, rather than code: model driven design and development, and model driven and model based testing are already going in this direction.
- The **code aspect** is addressed by separating the (application specific or reusable) "logic" from the implementation of the operations and the system. Service oriented computing leverages component based design together with a high level description of the interfaces and properties of the components as well as the behaviours (described as processes or as APIs) and data they operate upon.
- The **testing aspect** is streamlined by choosing modelling languages that facilitate an early stage "checking" of the model structure, of the architectural and behavioural compatibilities. Architecture Analysis and Description Languages (AADLs) cover the architectural and static aspects, while the use of formal models like the KTS used in jABC [24], DIME [2] and in general graph-based models supported by CINCO [29] allow also a behavioural analysis e.g. by model checking and in some cases a correct-by-construction synthesis of property-conform models, thus delivering the "speed" of early detection and even avoidance of errors that makes then testing on the code much faster.
- The **dissemination aspect** is taken care of by sharing such models (understandable to the domain experts) and possibly also the implementations of the building blocks, this by using for example the existing OSS facilities and structures. Libraries of services have been in use in the telecommunication domain since the '80s [35], they are increasingly in use in bioinformatics, geoinformation 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, adding one or more model-to-code layers on top of the already long chain of compilers that take UML classes and create skeletons, or Java code and produce bytecode, or C/C++ code and produce executables, and from the executables goes down to the specific instruction set and firmware of the processors, FPGAs and other ASICS. Generation can be partial, as in the UML community, or more extended as e.g. in the Grammatical Evolution approach by [27], that creates in fact programs by successive approximation based on a specific flavour of genetic algorithms that operate along a grammar-driven DSL of the specific domain and operations under consideration. In jABC, CINCO and DIME we use both model-to-model and model-to-code transformations, starting from the Genesys approach of [16,17], up to the generalized approach that Cinco adopts also at the tool metalevel [29].
- 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 [6,39] or language-driven engineering [31].
- 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 [1,20], postgraduate students with a background in biology and geography [19], and more recently first year students and mature CS students that take a 1 year Higher Diploma [11].
- 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 and One Thing Approach, applied to models of systems but also of test cases [30].
- The **structuring approaches** based e.g. on hierarchy [32,33], on several notions of features like in [5,14,15,18,25,34], or contracts for abstraction and compositionality as in [13]. These structures allow a nice and incremental factoring of well characterized system components to aide the hierarchical and collaborative organisation of complex or large systems, while supporting the intuition of the domain experts, and the most opportune units of reuse (for development, evolution and testing), and units of localized responsibility, e.g. for maintenance, evolution and support.

Some of this is already in the making even at the level of hardware and platforms, which are essential for the connected and evolving industrial critical systems of tomorrow.

7 Examples of Future-Oriented Rethinking

On the hardware side the foundation of all software executions is the computer architecture, possibly enhanced by various layers of virtualization and resource management. In terms of properties, guarantees, contracts, service level agreements, upper layers of the software and system management stack must rely upon what is known and guaranteed by the underlying layers. So, what is known today about computer architectures and the software layers that enforce essential platform properties? We look at three recent contributions that provide a glimpse of the promising research directions that are amenable to bring formal thinking and generative approaches closer to the mainstream.

7.1 Rich, Formally Specified Interfaces from the Bare Metal Upward

Margaret Martonosi, recent recipient of the IEEE Computer Society Technical Achievement Award "for contributions to power-aware computing and energy-constrained mobile sensor networks", reflected in her recent keynote at IEEE COMPSAC that computer architects are until now required yet to a good extent also restricted to measure in architectures only execution performance, thus sticking to a one-dimensional characterisation of quality that is short of today's architectural demands. Her plea for modern computer systems is towards a far richer and multifaceted characterisation. Such a profile-like characterisation includes next to figures of energy consumption also values for reliability, fairness, portability, scalability, security, and many more. These rich descriptions of the Computer Architecture interfaces should be formulated in an unambiguous language, and she advocates using formal methods for describing and reasoning about interface specification, modeling and metrics [26].

In recent work with Sharad Malik and Aarti Gupta, she used such formal specifications, albeit simple, first to describe the capabilities of an instruction set, an API, and equivalently specifiable behavioural mechanisms for the many other components that do not have an instruction set (called non-ISA components, like most Internet of Things devices), Subsequently they used those descriptions to analyze both the real and the possible behaviours of systems in isolation and in a friendly or hostile context. They used these capabilities and, in a linearized world, the "happens before" behavioural relation that allows to think in terms of system behaviours over sequences of operations, in a trace semantics approach. This led to the development of simple but effective and efficient tools that work as a set of concern-specific lenses to talk about global behaviours at the system level. Tools like Wattch [4], PipeCheck [21] and COATCheck [22] have made the **systematic generation and exploration of architecture-level behavior** both **possible** and **easy**, regarding power optimisation, memory consistency, and memory ordering at the hardware-OS interface, respectively.

Thanks to the rich interface formalisation, it is possible to describe exactly what is intended, what is forbidden, and what happens in a situation-dependent context, and it is consequently possible for the first time to build tools, in

this case based on SAT solvers, that systematically explore and classify the behavioural traces.

Eminent achievements have been the automatic and easy reproduction of the Meltdown and Spectre security attacks using the above tools. Not only were such attacks easily "discovered": the discovery came together with an understanding of which design features and mechanisms of interaction led to them, and this understanding of course led to an entire design space of ways to detect and prevent such and similar attacks. Additionally, they were able to generate a large number of other so far unreported attacks that are possible on the same architectures and mechanisms, some of which so relevant that they were reported back to the companies that designed such systems.

Thanks to such behavioural analysis tools, it is made easy for designers of higher level infrastructure software and of applications, to check and recheck that their own designs do not incur in behavioral risks or inconsistencies. Weighted edges on automata-like models can represent latency, power or reliability figures, and provide rich analysis models down to the instruction set layer, which is so far not possible. Upper layers can then rely on the data measured at the lower layers, and enable a systematic design space exploration driven by rich models that embody detailed and multifaceted knowledge at the underlying levels: still a dream for today's system designers, as we will see in Sect. 7.3.

7.2 Ad-Hoc Computing in the World Beyond von Neumann's Architectures

However, considering the future needs, it is even likely that the Von Neumann architectural paradigm will be superseded. As HP Labs Dejian Milojicic foresees [28], architectures for large scale in-memory computing will revolution the operating systems as we know them today and change the entire organization of what is a computer due to he emergence of new nanocomponents. Current research on emerging technologies for memory design, like memristors in HP but also other emergent memory components like Magnetic Random Access Memory (MRAM)[3], studied among others in the TRUDEVICE EU COST Action UL belonged to, are hot candidates to successfully replace actual DRAM-based main memory technology. On the MRAM technology roadmap, the goal is to design and develop ?Service-Oriented? emerging memory devices based on non-volatile MRAM technology, with characteristics and mode of functioning tunable in an ad-hoc fashion by individual application designers, as needed to meet their specific project/work needs. Rapid reconfigurability of the work is here the main benefit, but configurable and controllable (1) reliability, (2) variability, (3) endurance, (4) access time, (5) bandwidth, (6) latency, and (7) power consumption will allow application designers and programmers direct QoS control on memory access services. On the memristor technology roadmap, research leads mainly towards the development of a new kind of passive storage. However, it is known how to produce memristor based devices that implement negation

[3] ITRS 2013, http://public.itrs.net/.

and implication and thus generic logic functions. Accordingly, we can envisage scalable architectures where memristor-based memories manage the data, and efficient tiling and interconnections of memristor-based logic organized in crossbar architectures (like for FPGAs) provide easily reconfigurable "control" units that supersede the current Von Neumann architectures. The consequence is a new generation of architectures that provide a memristor-based generic *memory and control* capability that can be configured and used completely on-demand.

Fully exploiting either the Service Oriented MRAM Memory device or the memristor-based architecture requires revolutionizing the current programming model, and the way we organize and handle data, control, and even the set of instructions at the hardware level. In such a post-Neumann architecture, computations consist of

- the data, to be stored e.g. in the memristor storage,
- the "program", to be e.g. "loaded" on the memristor-based control architecture, but whereby
- the Instruction Set of the architecture (i.e., the primitives of the programming language) is not predefined and static like in current architectures, but easily and fully reconfigurable beyond the level of reconfigurability that we know today from FPGAs. In this future, the definition of the instructions in terms of hardware implementation of the primitives will be provided together with the "program", and the instructions will be "mapped" themselves on the control architecture in order to define the set of functionalities available to the specific program.

Taken together, we see here the proposal of a completely ad-hoc and dynamic computation model and system: a Domain Specific Language, and potentially even an Application Specific Language, can be "deployed" case by case, program by program, onto an instruction set-agnostic architecture. By configuring a sea of memristors into instruction-executing logic components, this new paradigm effectively creates a Domain Specific Computer. This computer is then asked to execute a program in that DSL onto given data. When that execution terminates it is ready for a complete reconfiguration to another domain or application. In other words, this future paradigm ditches the inherent dicothomy of the Von Neumann paradigm, and embraces a thorough and generic service-orientation from the hardware up. Once such an architecture is available, implementing language-oriented programming [6,39] and language-driven engineering [31] throughout the entire stack of software layers, down to the hardware, becomes natural and obvious.

Once this is in place, it will be then finally possible to have formally characterized capabilities and interfaces for a new kind of service-oriented general purpose efficient hardware, that is pliable to fast and efficient reconfiguration, and that brings a rich description of both functionality and quality of service in terms of properties and behaviours. The approach just described applies not only to instruction set-based hardware like today's CPUs, but also to API-based components like e.g. Smartcards, which support at different abstraction layers

interfaces like the Transmission Protocol Data Units and the Application Protocol Data Units. It applies even to non-ISA components like the FPGAs and most accelerators and IoT devices.

7.3 Security-Injecting Compilers

This is exactly the situation we are currently facing in the project with Blu5 Labs: the SEcube (TM) chip they provide includes three open source components (a CPU, an FPGA and a Smartcard) [38] and is therefore an advanced and open System-on-a-Chip specialised for security applications from the hardware up. However, no formal description of the interfaces of these components is available, as demanded in Sect. 7.1, no formally robust behavioural characterization is available, and as a consequence we cannot bootstrap the formal reasoning from the hardware up as should be the case for an SoC component that is central to a holistic security concept, and responsible for the hardware layers of a security architecture used commercially for top-level security applications.

On the basis of the open source descriptions of the three components and through testing with the SoC programming boards, it is possible to manually write drivers (in machine language and C) that realize several layers of services [8], that are subsequently thoroughly tested in the laboratory. This is still the current state of the art in security architectures: at the hardware interface and at the driver and relative protocols nothing is formal, architectures are designed case by case, software for protocols and low level features is implemented in "coding first" fashion, and architects and designers mostly do not feel the need for introducing formality at all.

In this situation, we built our own model driven, service oriented and generative approach on top of the software layers produced manually by the partners. We produced a DSL library for the multifaceted characterization of security primitives [3], a service library for the primitives at the programming language level, in this case the C language, embedded in the C-IME, our C-applications Integrated Modelling Environment [10,12], and we provided libraries for other domains as in [7,9]. On this basis it is also possible to provide a smart compilation platform, itself realized in a model driven paradigm, that is a model-to-code compiler to normal C or to C with security added through the SEcube platform. This compiler takes security-agnostic models of applications designed in the C-IME environment, and injects during compilation where needed the appropriate security primitives. The result is the same C application, but where all the communication is correctly secured using through the C language libraries and the SEcube security libraries [12].

This generative and model driven approach works in a model driven, service oriented and generative fashion, on security-agnostic application models that are far away from the level of programming, hardware and security competence needed today to create comparably secure software. It successfully frees the application designer from the need of specialized knowledge about security mechanisms and their implementation details. However, in today's world, this ability of modelling, analyzing and proving properties of behaviours is still

limited to aim at the provision of executable functionality (of an application). We'd rather like instead to provide a provably bug-free and attack-resilient hardware/software stack. This limitation is due to the informal treatment of hardware interfaces, and to the customary code-based approach to the production and documentation of the low level layers of the SEcube-close software, which are directly implemented in machine language and C and are not accompanied by specifications nor models and are thus in practice not verifiable.

8 The Perspectives Ahead

In spite of the difficulties, we are convinced that this is the way to go in order to mainstream the production and evolution of IT at the levels of speed, scale, affordability and collaborative effort needed to truly make IT enter the fabric of every economical and societal endeavour. It needs to be achieved at a level of competence that can be acquired quickly, in a targeted fashion, and such that the collaboration with the IT specialists is likely to become natural and pervasive. Specialists will still exist and be essential for the implementation and maintenance of the code and of the infrastructure. But 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 [36] and future oriented knowledge management for change management [23] in order to make future platforms *easy but powerful for the many*, and *complex yet well structured for the few*.

Acknowledgment. This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

References

1. Bakera, M., Jörges, S., Margaria, T.: Test your strategy: graphical construction of strategies for connect-four. In: Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems, pp. 172–181, ICECCS 2009. IEEE Computer Society, Washington, DC (2009). http://dx.doi.org/10.1109/ICECCS.2009.51
2. 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
3. Boßelmann, S., Neubauer, J., Naujokat, S., Steffen, B.: Model-driven design of secure high assurance systems: an introduction to the open platform from the user perspective. In: Margaria, T., Solo, M.G.A. (eds.) The 2016 International Conference on Security and Management (SAM 2016). Special Track "End-to-end Security and Cybersecurity: from the Hardware to Application", pp. 145–151. CREA Press (2016)

4. Brooks, D., Tiwari, V., Martonosi, M.: Wattch: a framework for architectural-level power analysis and optimizations. In: Proceedings of 27th International Symposium on Computer Architecture, vol. ISSN=1063-6897, pp. 83–94. IEEE (2000)
5. 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). http://doi.acm.org/10.1145/2993412.3003390
6. Dmitriev, S.: Language oriented programming: the next programming paradigm. JetBrains onBoard Online Mag. **1** (2004). http://www.onboard.jetbrains.com/is1/articles/04/10/lop/
7. Farulla, A., Lamprecht, A.L.: Model checking of security properties: a case study on human-robot interaction processes. In: 12th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS), pp. 1–6. IEEE Computer Society (2017). https://doi.org/10.1109/DTIS.2017.7930158
8. Farulla, G.A., Prinetto, P., Varriale, A.: Holistic security via complex HW/SW platforms. In: 12th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS), pp. 1–6. IEEE Computer Society (2017). https://doi.org/10.1109/DTIS.2017.7930156
9. Farulla, G.A., Indaco, M., Legay, A., Margaria, T.: Model driven design of secure properties for vision-based applications: a case study. In: Margaria, T., Solo, M.G.A. (eds.) The 2016 International Conference on Security and Management (SAM 2016). Special Track "End-to-end Security and Cybersecurity: from the Hardware to Application", pp. 159–167. CREA Press (2016)
10. Gossen, F., Tiziana Margaria, J.N.B.S.: A model-driven and generative approach to holistic security. In: Flammini, F. (ed.) Resilience of Cyber-Physical Systems: From Risk Modeling to Threat Counteraction. Advanced Sciences and Technologies for Security Applications. Springer, Heidelberg (2018). ISBN: 978-3-319-95597-1
11. Gossen, F., Kühn, D., Margaria, T., Lamprecht, A.L.: Computational thinking: learning by doing with the Cinco adventure game tool. In: 42nd IEEE Annual Computer Software and Applications Conference (COMPSAC), CELT Symposium, Tokyo, Japan, 24–27 July 2018. IEEE Computer Society (in press)
12. Gossen, F., Neubauer, J., Steffen, B.: Securing C/C++ applications with a secube^{TM}-based model-driven approach. In: 12th International Conference on Design & Technology of Integrated Systems in Nanoscale Era, DTIS 2017, Palma de Mallorca, Spain, 4–6 April 2017, pp. 1–7. IEEE (2017). https://doi.org/10.1109/DTIS.2017.7930157
13. Graf, S., Quinton, S., Girault, A., Gössler, G.: Building correct cyber-physical systems: why we need a multiview? In: Howar, F., Barnat, J. (eds.) FMICS 2018. LNCS, vol. 11119, pp. 19–31. Springer, Cham (2018)
14. Jonsson, B., Margaria, T., Naeser, G., Nyström, J., Steffen, B.: Incremental requirement specification for evolving systems. In: Calder, M., Magill, E.H. (eds.) Feature Interactions in Telecommunications and Software Systems VI (FIW 2000), pp. 145–162. IOS Press, May 2000
15. 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
16. Jörges, S.: Construction and Evolution of Code Generators. A Model-Driven and Service-Oriented Approach. LNCS, vol. 7747. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36127-2

17. Jörges, S., Margaria, T., Steffen, B.: Genesys: service-oriented construction of property conform code generators. Innov. Syst. Softw. Eng. **4**(4), 361–384 (2008)
18. 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
19. Lamprecht, A.-L., Margaria, T. (eds.): Process Design for Natural Scientists. An Agile Model-Driven Approach. CCIS, vol. 500. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45006-2
20. Lamprecht, A., Margaria, T., McInerney, C.: A summer computing camp using ChainReaction and jABC. In: 40th IEEE Annual Computer Software and Applications Conference, COMPSAC Workshops 2016, Atlanta, GA, USA, 10–14 June 2016, pp. 275–280. IEEE Computer Society (2016). https://doi.org/10.1109/COMPSAC.2016.41
21. Lustig, D., Pellauer, M., Martonosi, M.: PipeCheck: specifying and verifying microarchitectural enforcement of memory consistency models. In: 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 635–646. No. ISSN=1072-4451. IEEE (2015)
22. Lustig, D., Sethi, G., Martonosi, M., Bhattacharjee, A.: COATCheck: Verifying memory ordering at the hardware-OS interface. SIGPLAN Not. **51**(4), 233–247 (2016). http://doi.acm.org/10.1145/2954679.2872399
23. 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
24. Margaria, T., Steffen, B.: Lightweight coarse-grained coordination: a scalable system-level approach. Softw. Tools Technol. Transfer **5**(2–3), 107–123 (2004)
25. Margaria, T., Steffen, B., Reitenspieß, M.: Service-oriented design: the roots. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 450–464. Springer, Heidelberg (2005). https://doi.org/10.1007/11596141_34
26. Martonosi, M.: New metrics and models for a Post-ISA era: managing complexity and scaling performance in heterogeneous parallelism and internet-of-things (keynote talk). In: 42nd IEEE Annual Computer Software and Applications Conference (COMPSAC), CELT Symposium, Tokyo, Japan, 24–27 July 2018. IEEE Computer Society (2018, in press)
27. Medernach, D., Fitzgerald, J., Azad, R.M.A., Ryan, C.: A new wave: a dynamic approach to genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, pp. 757–764. GECCO 2016. ACM, New York (2016). http://doi.acm.org/10.1145/2908812.2908857
28. Milojicic, D.: Generalize or die: operating systems support for memristor-based accelerators (keynote talk). In: 42nd IEEE Annual Computer Software and Applications Conference (COMPSAC), CELT Symposium, Tokyo, Japan, 24–27 July 2018. IEEE Computer Society (2018, in press)
29. 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**, 327 (2017)
30. 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

31. 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. Springer, Heidelberg (2018)
32. Steffen, B., Margaria, T., Braun, V., Kalt, N.: Hierarchical service definition. Ann. Rev. Commun. ACM **51**, 847–856 (1997)
33. Steffen, B., Margaria, T., Claßen, A.: Heterogeneous analysis and verification for distributed systems. Softw. Concepts Tools **17**(1), 13–25 (1996)
34. Steffen, B., Margaria, T., Claßen, A., Braun, V.: Incremental formalization: A key to industrial success. Softw. Concepts Tools **17**(2), 78–95 (1996)
35. 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)
36. Steffen, B., Naujokat, S.: Archimedean points: the essence for mastering change. In: Steffen, B. (ed.) Transactions on Foundations for Mastering Change I. LNCS, vol. 9960, pp. 22–46. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46508-1_3
37. 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
38. Varriale, A., di Natale, G., Prinetto, P., Steffen, B., Margaria, T.: SEcube^{TM}: an open security platform: general approach and strategies. In: Margaria, T., Solo, M.G.A. (eds.) The 2016 International Conference on Security and Management (SAM 2016). Special Track "End-to-end Security and Cybersecurity: from the Hardware to Application", pp. 131–137. CREA Press (2016)
39. Ward, M.P.: Language oriented programming. Softw. Concepts Tools **15**(4), 147–161 (1994)