Challenges in Software Development for Multicore System-on-Chip Development

Ian Gray, Neil C. Audsley
Real-Time Systems Research Group, Dept. of Computer Science, University of York, U.K.

***Abstract*—Multiprocessor Systems-on-Chip (MPSoC)-based platforms are becoming more common in the embedded domain. Such systems are a significant deviation from the homogeneous, uniprocessor architectures that have been traditionally employed by embedded designers, thereby making the software development process to effectively target the platform more challenging. Low-resource embedded systems rely on efficient implementations that are not well supported by traditional solutions based on architecture virtualisation or middleware. Within this paper we examine these challenges and discuss ways in which they can be mitigated. In particular, we focus on the contributions made by two recent approaches based on Model-Driven Engineering (MDE). We also discuss challenges for future research.**

***Keywords*-MPSoC, Embedded Systems, Model-Driven Engineering, Real-Time Systems, MADES, Touchmore, T-CREST.**

I. Introduction

Many modern embedded systems target Multiprocessor Systems-on-Chip (MPSoC)-based platforms. The architectures of these systems are becoming increasingly complex and are poorly supported by existing languages and toolchains. This paper explores the ways in which language and tool support is lacking and enumerates some of the major challenges facing the development of industrial MPSoC-based systems. Motivated by this, the paper then describes two recent projects that attempt to tackle these issues and the experience gained from them.

The architectures of MPSoCc are a significant deviation from the homogeneous, uniprocessor platforms that have traditionally been the main component of embedded architectures. As MPSoCs become more common, this difference is complicating the process of creating effective and efficient software systems for the platform, which is vital given the tight resource constraints of many embedded systems. In addition to this, in many domains embedded software must be certified before deployment (such as in the avionics or automotive domains). This may require code which can be analysed for worst-case execution time (WCET) and worst-case response time (WCRT); that supports traceability from specification to code; or that can be verified using techniques such as model-checking or formal proof. Finally, the integration of legacy code is important due to the high cost of redeveloping and recertifying systems. Within this paper we examine these software development challenges and examine how they are being mitigated by current Model-Driven Engineering (MDE) approaches; and which significant challenges remain for future research.

This work was supported by EU Framework 7 research contracts FP7-248864, FP7-288008 and FP7-288166

The move to such MPSoC-based designs has been driven by large increases in transistor density coupled with relatively modest increases in maximum clock rates [17]. This has forced the exploration of multi-processor architectures with heterogenous processing components in order to meet increasing application performance requirements.

MPSoCs are differentiated from more simple multiprocessor architectures by their tendency to contain multiple, *heterogeneous* processing elements [20], non-uniform memory structures [1], and non-standard communication facilities such as on-chip networks (e.g Network-on-Chip (NoC) communications structures seen on the Tilera 64-core TILEPro64 processor [35] and the Intel 48-core Single-Chip Cloud Computer [21]). MPSoCs make extensive use of application-specific hardware (i.e. DSP cores, function accelerators, or configurable processors [10]). For example, the recent OMAP 5 range of MPSoC from Texas Instruments [28] contains a dual-core ARM Cortex A15, two other smaller ARM cores, DSPs, and a GPU core. For MPSoCs there are no accepted *standard platform architectures*, unlike previous uniprocessor / SMP multiprocessor architectures. This means that MPSoCs are not well-supported by the standard toolchains and languages that have been developed over the past forty years or so.

The major challenges facing the development of industrial MPSoC-based systems are explored in section II. Section III then describes the experience of addressing some of these challenges during two recent research projects, and section IV discusses future challenges and how they may be solved.

II. Major MPSoC Challenges

The challenges detailed in this section have been identified through embedded systems research and over a range of projects in collaboration with industry partners. This list does not aim to be exhaustive, and is focussed towards the development of software rather than the additional challenges associated with hardware development and verification.

A. The Programming Model Gap

MPSoCs present unique challenges for software development due to the change from uniprocessor to many or multiprocessors (implying parallelism) and non-uniform memory structures. The following issues are identified.

Issue: Programming Model and Platform Mismatch
According to the TIOBE index [33], an informal ranking

of language popularity, the three most commonly-used programming languages in the world are C, Java and C++. In embedded and safety-critical settings, (subsets of) C and C++ dominate. However, the programming model of all of these languages assumes a homogeneous implementation architecture with a uniform, shared memory space. This model is incompatible with the application-specific, heterogeneous architectures of MPSoCs. This results in a mismatch between the programmer's conceptual model and the underlying implementation.

These languages all make the following *assumptions of universality*, that are desirable to the programmer but are difficult to implement in a highly-parallel, non-uniform MPSoC:

- **Universal communications:** Scope rules aside, most languages assume it is possible to call functions or methods in any part of the program from any other part. On a multicore embedded system, this can be difficult to implement as on-chip communications must be considered. Even in a system where communications between all nodes can be supported, latency and routing are issues. Systems tend to employ weighty middleware systems such as Real-time CORBA [37] to provide this, or rely on the routing of an on-chip network.
- **Universal memory:** Related to the problem of universal communications, these language models assume that data is stored in a single, logical memory space which is coherent and available at all points in the application. Such a model does not scale to support large numbers of memories/caches [3]. On application-specific embedded platforms this is especially wasteful as the programmer frequently only needs to keep a few small areas of their target application coherent. Some recent work has considered ways to limit this problem [8], [15] based on limiting the programming model and allowing greater programmer expressibility.
- **Universal services:** All standard languages assume the availability of operating system services. For example, Java assumes threading, synchronization, etc. and even basic C assumes the presence of a memory allocator. Maintaining a coherent view of such services over an MPSoC requires a distributed OS [3] and can be very expensive to implement without the programmer limiting the required sharing and coherency [4].

Given the large number of processing cores on recent MPSoCs, such as Tilera's 64-core TILEPro64 processor [35] or Intel's 48-core Single-Chip Cloud Computer [21], scalability is a huge concern for developers. These assumptions of universality are a barrier to adopting scalable implementations in the general case.

To attempt to reduce these assumptions, the Multicore Association has proposed a set of programming APIs [14], [30] aimed at embedded systems. The purpose of these APIs is to extend the programming model of the source language (normally C) to include multicore concepts (communication, resource management etc.) in a more scalable way.

Issue: Virtual Platform Assumption

Aside from the universality assumptions, a further weakness of existing programming models is that the programmer is not able to argue about the mappings of their software to the target architecture. Hardware features such as MMUs and software OSs hide implementation details to present a simple and uniform hardware model to aid software development. However, this virtualisation also prevents the programmer from accessing the actual hardware. Some languages have started to allow this in a limited capability. For example, Java's Real-Time Specification [11] (RTSJ) and POSIX allow basic mapping of threads to processors and to model physical memory, but as described above a largely homogeneous architecture is still assumed. Developers must rely on language extensions or extra-linguistic techniques (e.g. custom tools and linker scripts) to fully exploit complex hardware.

New programming languages are often proposed as a way of solving this problem. Languages exist to target complex memory systems [9], highly-parallel architectures [6] and many others. However, until these become a de facto standard they are unlikely to be adopted by industry, and their new languages and tools are a barrier to certification.

B. Rapid Design Space Exploration

A number of issues exist that slow the rate at which the solution space can be explored in a MPSoC design cycle.

Issue: Unpredictable Design Performance

As systems become more complex, it is becoming harder to reason about the optimal implementation of a given system. Even after algorithms have been chosen and implemented many factors still need to be considered. For example; the location of tasks/threads, the use of communications or the scheduling of computation and communications. Furthermore, offline analysis of systems has not yet caught up with the flexibility of the available architectures, so designers rely on virtual implementation platforms [7], simulation, in-circuit measurement, or designer intuition, to guide development.

Issue: Drivers and Low-Level Code

Design space exploration (DSE) is slowed due to the programming model gap that requires developers to create large amounts of "glue" code manually (drivers etc.), and to refactor existing code for every reallocation. Co-design approaches [26], [27] have attempted to automate this process to varying degrees of success, however many suffer from the lack of a suitable input specification. Co-design requires that the units of the system can be implemented interchangeably as either hardware or software. As discussed in section II-C code translation poses problems for high-integrity industries and section II-D describes how the use of new languages is undesirable.

C. Criticality

Many MPSoCs are subject to real-time or safety-critical constraints, and so their development must consider issues:

Issue: Predictability and Analytical Complexity
Hard real-time systems must be amenable to WCET and WCRT analysis so that their worst-case timing behaviour can be identified and accounted for. This requires predictability at all stages of the design, from language choice (frequently a high-integrity subset such as Ravenscar Ada [5] or Java [19]) through a real-time OS / kernel (such as MARTE OS) to real-time hardware features (such as the CAN bus, or SoCBUS [36]).

Safety-critical systems must be certified to the highest levels, such as DO178-B in the avionics domain. This requires code that can be statically-analysed, tested, or proved correct. This is a great challenge on its own, but it is made even more difficult in an MPSoC environment due to the programming model gap detailed in section II-A. The architecture assumed by the standard programming model is a homogeneous uniprocessor or SMP system with predictable, uniform, memory timing, no bus contention, bounded blocking and a predictable, coherent, OS layer that provides universal services. In reality, the programmer must use low-level drivers and techniques outside of the programming model (custom assembly, manual memory allocation, code splitting etc.) to target the hardware. This makes code analysis more challenging.

Also MPSoC hardware frequently includes unpredictable elements. Caches are very difficult to analyse and real-time network routing is still an area of highly active research [36]. Finally, the layers of middleware, virtualisation, or distributed OSes introduce huge amounts of software to the system that have to be analysed for timing, path coverage, functional correctness or whatever is specified by the certification standard. For the highest levels of certification this is currently infeasible.

Issue: Traceability
Related to the issue above, many high-level certification standards require *traceability*, which requires every source line to be traceable back to the requirement (or model element) that generated it. The "glue code" and layers of middleware discussed in the previous section all lack this traceability from the source specification, and so need to be considered specially. This makes certification harder, and so their use is minimised.

D. Industrial Applicability

Industry is generally reluctant to switch to new programming languages and toolchains, as this imposes a revolution in development approach, with implicit problems of risk, acceptance and difficulties with legacy systems. Hence a key issue is as follows:

Issue: Evolutionary Adoption of MPSoC Toolchains
For MPSoC platforms to be successfully exploited by industry users, there is a need to *evolve* toolsets, programming languages and development approaches from the current uniprocessor biased approaches to having an MPSoC focus. This is required because:

- Reuse of legacy code is important, as the development and verification costs have already been paid. Switching to a new language may require reimplementation and retraining of developers.
- In a high-integrity domain, tools (compilers, linkers etc.) have to reach a 'trusted' status before they can be used for strict certification. This can be a problem for many academic tools.

In general, existing industrial methodologies must be supported rather than supplanted. Model-driven engineering (MDE) is becoming more frequently used in industrial projects [22] and represents a common way of tackling the higher abstractions of modern embedded systems [16]. However, as with programming languages it is desirable to remain with existing modelling standards (such as SysML [34] or MARTE [23]) and tooling wherever possible. Another parallel with restricted programming languages is that UML and profiles like MARTE are very complex and there are many different ways to model the same concept, so restricted and more focussed subsets can help with productivity.

III. Tackling challenges using Model Driven Engineering

This paper will now describe experience gained from recent efforts to overcome some of the challenges identified in section II. MADES [29] and TouchMore [32] are two EU Seventh Framework projects that are using approaches based around Model-Driven Engineering (MDE) to assist the development of modern embedded systems. MADES is approaching final review so will be concentrated on by this paper, whereas TouchMore is in its first year. Section III-A will describe the MADES project, and section III-B will describe TouchMore. Then the following sections detail how the projects approach the challenges of industrial applicability, complex architectures and criticality.

A. The MADES Approach

The MADES approach [29] was developed as an EU Seventh Framework project to provide a fully MDE approach for the design, validation, simulation, code generation and deployment of multi-processor embedded systems. In MADES, high-level system models (in a combination of SysML [34] and MARTE [23]) provide information about the target architecture and the structure of the input software. MADES focusses on verticality and applies all the way from requirements to implementation. Input models and diagrams are transformed in traceable and certifiable steps from initial specification to the final deployed system using the Epsilon model transformation tools [18].

The MADES Consortium consisted of two major industrial partners from the avionics and automotive domains, so they are familiar with certification and real-time issues. As will be detailed, this was key to gaining useful insight from the project. All aspects of the toolflow are fully validated on real-life case studies in the surveillance and avionic domains by MADES' industrial partners in order to ensure industrial

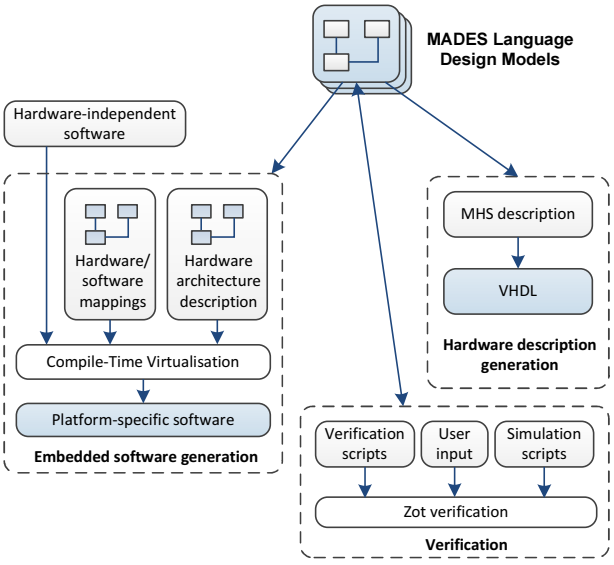

Fig. 1. Overview of the artifacts in the MADES approach

applicability (a challenge discussed in section II-D). MADES incorporates three parallel toolchains for performing hardware development, temporal verification/validation/simulation, and software development. The benefits of each of these toolchains will be discussed in terms of how they relate to the challenges of MPSoC development.

B. The TouchHMore Approach

TouchHMore [32] is also an EU Seventh Framework project focussed on using a model-driven approach to capture the key characteristics of the platform in order that the toolchain may be customised to meet its specific requirements. Given the enormous potential variation in MPSoC architectures and platforms and the inability of programming models to capture this (the 'programming model gap' of section II-A), there is a fundamental need to support customisable toolsets that can target a wide range of such platforms. This supports industry because no change is required in the toolset to target a new platform, unlike today where new development processes, modelling processes and tools must be built. The aim to to provide a better programming model that supports the challenge of providing rapid DSE (section II-B).

C. Evaluation Approach

In order to evaluate the effectiveness of these approaches, the consortiums of both projects include industrial partners who generate realistic, industrially-relevant case studies.

The Car Collision Avoidance System (CCAS) was one of two major MADES case studies. The CCAS is a real-time, safety-critical system that uses a car-mounted radar to apply braking when a collision is imminent. The system has been specified as a set of initial system requirements in the MADES modelling language using requirements and use case diagrams. These are refined to system class diagrams then to software, hardware, and allocation specifications. These can then be verified, and implemented, using the MADES approach.

The Avionics Weather System (AWS) is the second MADES case study. The AWS is an avionics weather system which uses radar imaging and computational models to create a GUI image of the surrounding weather patterns. Unlike the CCAS, the AWS is a simpler system designed to be deployed on a single processor, although it has the same high certification requirements. The AWS is also specified using the MADES language.

In TouchHMore, the Audio Manager case study is a system that provides complex audio management for automotive and similar environments. The Audio Manager provides a multi-channel mixer with dynamic, routable audio channels. The application has strict timing requirements and low memory constraints.

Numerous synthetic studies have also been developed in order to specifically exercise individual sections of the respective toolflows. Experience gained from these case studies has allowed the evaluation of the attempt to overcome MPSoC development challenges. This is discussed in the following sections.

D. Industrial Applicability

A key requirement of both approaches is that their results must be usable by industry. This is a major challenge (discussed in section II-D) and indeed during the initial stages of the MADES project the academic and industrial partners had contrasting views. Academic partners did not appreciate the value of adapting existing toolchains rather than replacing them, and the industrial partners were not aware of the potential benefits that can be gained by augmenting their existing flow. The following points were most evident:

- Standards are essential for certification, but too permissive. The MARTE profile was deemed far too large to work with. Consequentially, a new modelling language was developed as a subset of SysML and MARTE that captured only the semantics required for the toolchain. The aim was to deliberately restrict developers, thereby reducing training time and assisting certification. This subset is supported by an industrial-quality open-source modelling tool, Modelio. A full discussion of the MADES subset is outside the scope of this paper but can be found in [25].
- Requirements capture is necessary for certification and not handled well by UML/MARTE. This why SysML concepts are included in the MADES language.
- Standardisation requires traceability from requirements through model elements to output software. The traceable model transformations of Epsilon allow a safety case to be built in which all output software can be traced back to the elements responsible for its creation.

These results have fed into the modelling used in the TouchHMore project, which uses an entirely SysML-based language subset that is restricted to only the concepts required by the project. All tools developed also have strong traceability

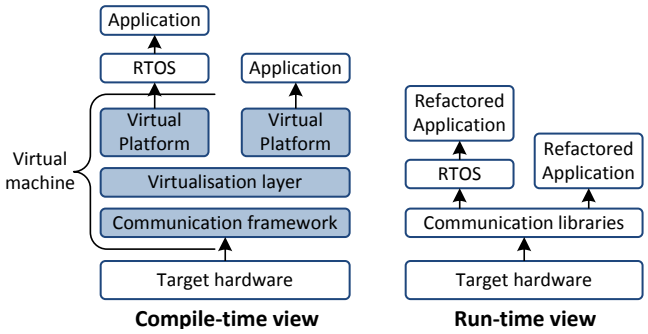

Fig. 2. Compile-Time Virtualisation hides complex hardware, but only at compile-time.

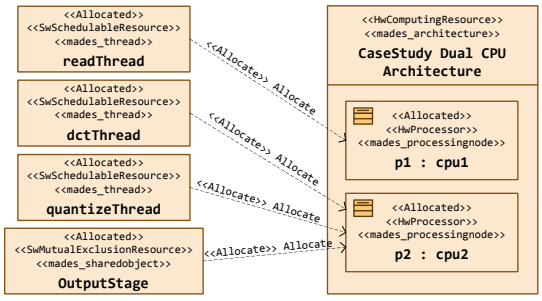

Fig. 3. CTV is integrated into the MADES modelling toolchain

requirements to aid the certification of systems developed using the approach. Sections III-E and III-F talk about how other aspects of the MADES project were modified to retain industrial applicability.

E. Software Generation Using Virtualisation

MADES attempts to tackle the programming model gap challenge (section II-A) by using virtualisation to hide the underlying complexity of the target architecture and allow software developers to write code for a virtual architecture that matches their programming model. However, through discussion with industrial partners, the criticality challenge (section II-C) mandates the following requirements:

- No new programming languages or tools because of certification requirements
- No large runtime layers, or complex translated code
- Integration with model-driven development to aid developers

As a result, a virtualisation technique called Compile-Time Virtualisation (CTV) is used. [12] CTV is a source-to-source translation technique that provides the benefits of virtualisation without the imposing a large runtime layer because its virtualisation is applied only at compile-time (see figure2). AnvilJ, the implementation of CTV used in MADES, has been shown to be amenable to resource-constrained, high-integrity, real-time systems. [13]

CTV is integrated fully into the MADES modelling language. Figure 3 shows an allocation diagram that allocates parts of legacy code from one of the MADES case studies to an

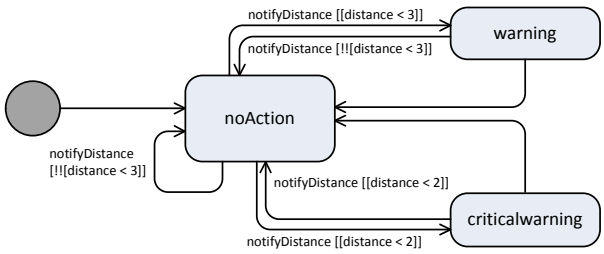

Fig. 4. Constraints on Interaction Overview Diagrams allow the modeller to provide verification properties

example implementation architecture. The designer can obtain extremely rapid DSE because moving code and data through the target architecture is a simple as editing an allocation diagram. CTV works in the toolchain to insert "glue code" and generate custom, minimally-sized microkernels for each processor that implement the modelled system.

Considering certification, in general new languages are avoided because their compilers must be certified (or trusted) and this is frequently too difficult a task. With CTV, two approaches can be taken. When compared to a compiler the CTV refactoring engine is a lot simpler and certification is believed to be more tractable. However, CTV's main benefit is that it only makes a few, small, traceable, readable, and well-defined changes to the input code and so certification effort remains the same. The only extra requirement is that CTV's microkernel must be certified. It is unavoidable that some runtime must be included, but as shown perviously [13] this runtime is minimal and is built to be amenable to many kinds of correctness analysis.

Through the use of CTV, the MADES approach has allowed the consortium to take code from the CCAS and other case studies and deploy it in seconds over a range of architectures without any refactoring or rewriting. Only the deployment and hardware diagrams need to be changed. Analysis and testing must still be performed on the final system, but this significantly lowers the barrier to such work.

F. Verification and Simulation

Formal verification techniques are well-developed in academic literature, but have failed to become commonplace in most industrial settings. This is due to their very high barrier to entry as developers are usually not trained in their use. However, such verification helps to tackle the challenge of criticality (section II-C), so MADES attempts integrate formal temporal verification with the model-driven development flow in an industrially-relevant way.

In the MADES approach, system behaviour and timing, and the properties to be verified are provided in Interaction Overview Diagrams (IODs) [2] (see figure 4). These are integrated into the normal modelling flow and supported natively by the tools. These models need to be converted to input for a temporal logic solver, in this case ZOT [24]. The input to Zot is a complex Lisp-based format which encodes temporal logic equations. In MADES, this complexity is hidden from the

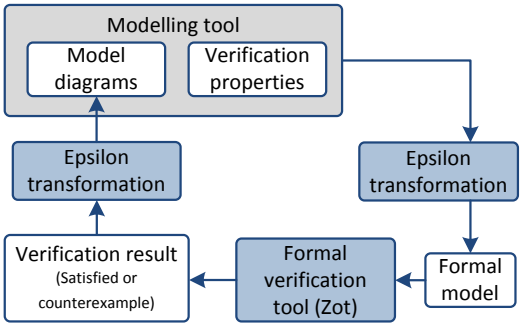

Fig. 5. Consistent metamodels allow model transformations to integrate verification

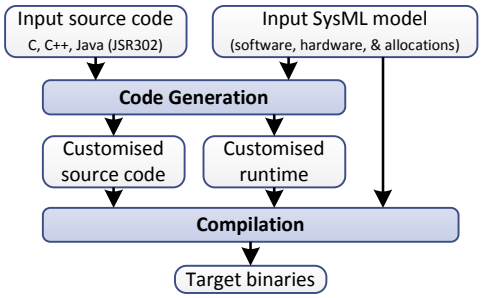

Fig. 6. The TouchHMore toolflow

designer as the scripts are generated using Epsilon's traceable model transformations. This can be done because the IODs and input scripts have a common and consistent metamodel that the translation uses (see figure 5).

When the verification is started, the scripts are generated, the verification tool runs, and the outputs from the tools are then translated back into model elements, according to the metamodel. This allows the integration of a useful verification tool, that otherwise would be inaccessible to many developers.

MADES also integrates Modelica simulation using a similar technique. Modelica is a textual modelling language for modelling complex systems (e.g. electrical, thermal, hydraulic or mechanical systems). The MADES co-simulation approach uses Modelica to simulate the external environment and provide stimulus to the ZOT verification. This allows for verification of systems as they interact with the environment.

G. Variability Awareness

The manufacturing process for many- and multi-processor MPSoC leads to variability in the capabilities of individual chips - eg. due to fabrication variability the CPUs within the chip may have different maximum clock speeds. Also, embedded MPSoCs are frequently battery powered, and may need to reduce their energy usage or thermal output (if cooling is an issue). This can be achieved through a range of techniques, including clock gating, dynamic frequency and voltage scaling (DVFS), offloading processing to DSP cores, and dynamic software that reacts to the design-time and run-time variability of the system. Building awareness of such variability into the design languages and toolchains with the aim of decreasing time-to-market is a key challenge that is currently being considered.

The TouchHMore approach is shown in figure 6. The input software, target platform, and allocations are modelled in SysML and provided alongside a software implementation of the system. The input software may be provided as C, C++ or Java. A code generation phase then customises code in the following ways:

- A TouchHMore runtime is generated for the target architecture that consists of three layers. The API layer is visible to the user program and includes variability calls that support power scaling, power-aware algorithms, DSP offload etc. The OS layer is a kernel that supports the features of the user software. The communications layer allows the OS layer to communicate with other OSes on other cores of the architecture to provide system-wide services. It is based on the Multicore Association's MCAPI [30] and MRAPI [14] APIs.
- User code (for example, methods that can be offloaded to DSPs) is rewritten to call the appropriate functions of the generated TouchHMore runtime.

The customised source code and runtime are then compiled using a compilation process that is itself variability-aware. Based on the architecture and allocations, the compiler will produce different sets of binaries, marshal data for intracore communications, and make use of DSPs, DMAs and shared memory as available.

IV. Conclusions and Future Challenges

This paper has enumerated some of the major challenges that are encountered when developing embedded MPSoC designs. In general, problems are encountered across the entire development spectrum: requirements capture and system modelling; the programming model of software languages; analysis and verification of complex architectures; certifying runtime systems and toolchains; and complex hardware implementations. This must also all be tackled in a manner that remains applicable to current industrial practice.

The MADES and TouchHMore projects have been described, which provide model-driven engineering approaches to the development of embedded systems. The approaches are designed fundamentally to be used by industry and to tackle many of the problems identified in the paper. Whilst the projects go some way to providing solutions for the identified challenges, they also highlight significant future challenges which are being considered within other projects:

- **Support for Predictability:** Worst-case execution time (WCET) analysis for complex embedded architectures is a significant open problem. Almost all of the schedulability and WCET analysis performed for uniprocessor systems no longer applies to multiprocessor systems. Furthermore, the architectures of MPSoCs tend to include difficult to analyse hardware features (caches, networks etc.). Worst-case analytical models of such systems are still not accurate enough for real-world use due to the

amount of pessimism they introduce. These issues are being considered within the T-CREST [31] project which aims to build a time predictable NoC based multiprocessor architecture, with supporting compiler and WCET analysis.

- **Simulation and Debugging:** For lower-criticality systems, simulation and testing can be sufficient. For these systems, simulation and debugging support is essential. Recent advances, such as hybrid simulation or high-level simulation models have helped this, but current approaches must balance accuracy with speed. Future challenges will involve the use of programming models, architecture designs, and simulation systems that are amenable to the slight inaccuracies of high-level simulation and can maintain confidence in the obtained result.

REFERENCES

[1] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad memory: design alternative for cache on-chip memory in embedded systems. In *CODES '02*, pages 73–78, 2002.

[2] L. Baresi, A. Morzenti, A. Motta, and M. Rossi. Towards the UML-Based Formal Verification of Timed Systems. In B. Aichernig, F. de Boer, and M. Bonsangue, editors, *Formal Methods for Components and Objects*, volume 6957 of *Lecture Notes in Computer Science*, pages 267–286. Springer Berlin / Heidelberg, 2012.

[3] Baumann et al. The Multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of SOSP '09*, pages 29–44, New York, NY, USA, 2009. ACM.

[4] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y. Dai, Y. Zhang, and Z. Zhang. Corey: an operating system for many cores. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 43–57, 2008.

[5] A. Burns, B. Dobbing, and G. Romanski. The Ravenscar Tasking Profile for High Integrity Real-Time Programs. In *Ada-Europe '98*, pages 263–275. Springer-Verlag, 1998.

[6] B. Chamberlain, D. Callahan, and H. Zima. Parallel Programmability and the Chapel Language. *Int. J. High Perform. Comput. Appl.*, 21(3):291–312, 2007.

[7] CoWare, Inc. CoWare Virtual Platform - Hardware/-Software Integration and Testing...Without Hardware. http://www.coware.com/products/virtualplatform.php (Accessed Aug 09).

[8] N. D. Enright Jerger, L.-S. Peh, and M. H. Lipasti. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In *MICRO '08: Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 35–46, Washington, DC, USA, 2008. IEEE Computer Society.

[9] K. Fatahalian et al. Sequoia: programming the memory hierarchy. In *SC '06*, page 83, 2006.

[10] R. Gonzalez. Xtensa: a configurable and extensible processor. *Micro, IEEE*, 20, Issue 2:60–70, 2000.

[11] J. Gosling and G. Bollella. *The Real-Time Specification for Java*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[12] I. Gray and N. Audsley. Exposing Non-Standard Architectures to Embedded Software Using Compile-Time Virtualisation. *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '09)*, 2009.

[13] I. Gray and N. C. Audsley. Developing Predictable Real-Time Embedded Systems using AnvilJ. In *Proceedings of The 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2012) Beijing China, April 17-19 2012*, 2012.

[14] J. Holt. Designing an Industry Standard API to Manage Multicore System Resources. http://www.multicore-association.org/webinar/090811_MRAPI.pdf, August 2009.

[15] H. Huang, N. Yuan, W. Lin, G. Long, F. Song, L. Yu, Y. Liu, L. Liu, Y. Zhou, X. Ye, J. Zhang, D. Fan, and Z. Tang. Architecture Supported Synchronization-Based Cache Coherence Protocol For Many-Core Processors. *Chinese Journal of Computers*, 8:1618–1630, 2009.

[16] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen. Empirical assessment of MDE in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 471–480, New York, NY, USA, 2011. ACM.

[17] ITRS. International Technology Roadmap for Semiconductors, 2007 Edition. http://www.itrs.net/, 2007.

[18] D. S. Kolovos, R. F. Paige, and F. A. C. Polack. Eclipse Development Tools for Epsilon. In *In Eclipse Summit Europe, Eclipse Modeling Symposium*, 2006.

[19] J. Kwon, A. Wellings, and S. King. Ravenscar-Java: A High Integrity Profile for Real-Time Java. In *In Joint ACM Java Grande/ISCOPE Conference*, pages 131–140. ACM Press, 2002.

[20] P. Marwedel. *Embedded System Design*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[21] T. Mattson, R. V. der Wijngaart, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, and S. Dighe. The 48-core SCC processor: the programmers view. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2010*, 2010.

[22] P. Mohagheghi and V. Dehlen. Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In I. Schieferdecker and A. Hartman, editors, *Model Driven Architecture Foundations and Applications*, volume 5095 of *Lecture Notes in Computer Science*, pages 432–443. Springer Berlin / Heidelberg, 2008.

[23] Object Management Group. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. http://www.omgmarte.org/, November 2009.

[24] M. Pradella, A. Morzenti, and P. San Pietro. The symmetry of the past and of the future: bi-infinite time in the verification of temporal properties. In *ESEC-FSE '07*, pages 312–320, New York, NY, USA, 2007. ACM.

[25] I. R. Quadri, A. Sadovykh, and L. S. Indrusiak. MADES: A SysML/-MARTE high level methodology for real-time and embedded systems. In *ERTS2 2012: Embedded Real Time Software and Systems*, 2012.

[26] F. Siebert. JEOPARD – Java Environment for Parallel Real-Time Development. *Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on*, 0:28–36, 2009.

[27] E. T. Silva Jr., D. Andrews, C. E. Pereira, and F. R. Wagner. An Infrastructure for Hardware-Software Co-Design of Embedded Real-Time Java Applications. *Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, pages 273–280, 2008.

[28] Texas Instruments Inc. OMAP5430 mobile applications platform. http://focus.ti.com/pdfs/wtbu/OMAP5_2011-7-13.pdf, July 2011.

[29] The MADES Consortium. The MADES Project. http://www.mades-project.org/, 2011.

[30] The Multicore Association. Multicore Communications API Specification V1.063 (MCAPI). http://www.multicore-association.org/workgroup/mcapi.php, March 2008.

[31] The T-CREST Consortium. The T-CREST Project. http://www.3sei.com/t-crest/, 2012.

[32] The TouchHMore Consortium. The TouchHMore Project. http://www.touchmore-project.eu/, 2012.

[33] TIOBE Software BV. TIOBE Programming Community Index for May 2012. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, May 2012.

[34] T. Weilkiens. *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[35] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. Brown, and A. Agarwal. On-chip interconnection architecture of the tile processor. *Micro, IEEE*, 27:15–31, Sept-Oct 2007.

[36] D. Wiklund and D. Liu. SoCBUS: Switched Network on Chip for Hard Real Time Embedded Systems. In *IPDPS '03*, page 78.1, 2003.

[37] V. F. Wolfe, L. C. DiPippo, R. Ginis, M. Squadrito, S. Wohlever, I. Zykh, and R. Johnston. Real-time CORBA. In *RTAS '97: Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium (RTAS '97)*, page 148, Washington, DC, USA, 1997. IEEE Computer Society.