Tool Integration and Interoperability Challenges of a System-Level Design Flow: A Case Study

Andy D. Pimentel[1], Todor Stefanov[2], Hristo Nikolov[2], Mark Thompson[1], Simon Polstra[1], and Ed. F. Deprettere[2]

[1] Computer Systems Architecture group
Informatics Institute, University of Amsterdam, The Netherlands
{andy,thompson,spolstra}@science.uva.nl
[2] Leiden Embedded Research Center, Leiden University, The Netherlands
{stefanov,nikolov,edd}@liacs.nl

Abstract. Daedalus is a system-level design flow for the design of multiprocessor system-on-chip (MP-SoC) based embedded multimedia systems. It offers a fully integrated tool-flow in which design exploration, system-level synthesis, application mapping, and system prototyping of MP-SoC architectures are highly automated. In this paper, we describe Daedalus from a software perspective, explaining its supporting software infrastructure and the way the various tools interoperate. Moreover, we discuss the lack of support for achieving tool interoperability that we have encountered during the development of Daedalus, and present several ideas of future research directions to address this issue. More specifically, we argue that a so-called Common Design Flow Infrastructure (CDFI) for system-level design flows is needed to improve and stimulate research and development in the area of system-level design methodology.

1 Introduction

The concept of system-level design of embedded systems, which raises the abstraction level of the design process to cope with design complexity, has been around for more than a decade now and has shown a lot of potential. Despite of this, system-level design still involves a substantial number of challenging design tasks. This is especially true for the design of MultiProcessor-SoC (MP-SoC) architectures, which become increasingly popular target platforms for modern embedded systems. For example, applications need to be decomposed into parallel specifications so that they can be mapped onto the multiple processing elements inside MP-SoC architectures [1]. Subsequently, applications need to be partitioned into HW and SW parts since MP-SoC architectures often are heterogeneous in nature. To this end, MP-SoC platform architectures need to be modeled and simulated to study system behavior and to evaluate a variety of different design options. Once a good candidate architecture has been found, it needs to be synthesized, which involves the synthesis of its architectural components as well as the mapping of applications onto the architecture. To accomplish all of these tasks, a range of different tools and tool-flows is often needed, potentially leaving designers with all kinds of interoperability problems. Moreover, there typically remains a large gap between the deployed system-level specifications (or models) and actual implementations of the system under study, known as the *implementation gap* [2]. Currently, there exist no mature

M. Berekovic, N. Dimopoulos, and S. Wong (Eds.): SAMOS 2008, LNCS 5114, pp. 167–176, 2008.

methodologies, techniques, and tools to effectively and efficiently convert system-level system specifications to RTL specifications.

Recently, we presented our Daedalus system-level design framework which addresses the above design challenges [3,4,5]. Daedalus' main objective is to bridge the aforementioned implementation gap for the design of multimedia MP-SoCs. It does so by providing an integrated and highly-automated environment for system-level architectural exploration, system-level synthesis, programming and prototyping. The Daedalus design flow, starting from sequential application to an implemented MP-SoC system on an FPGA with a parallelized application mapped onto it, can be traversed in only a matter of hours. Evidently, this offers great potentials for quickly experimenting with different MP-SoCs and exploring design options during the early stages of design.

In this paper, we describe Daedalus from a software perspective, providing insight of how the different tools in the design flow interoperate and describing the supporting tool infrastructure that improves the actual deployment of the design flow. Moreover, we discuss the lessons that we have learned from the development of Daedalus, mostly recognizing the lack of support for achieving tool interoperability, and present several ideas of future research directions to address this issue. More specifically, we argue that a so-called Common Design Flow Infrastructure (CDFI) for system-level design flows is needed, which surpasses ongoing efforts in this direction, in order to improve and stimulate research and development in the area of system-level design methodology.

The next section provides a birds-eye, conceptual overview of the Daedalus design flow. Section 3 describes the software infrastructure of Daedalus, after which Section 4 discusses some of the lessons we have learned from Daedalus' development. In Section 5, we present several initial ideas about a Common Design Flow Infrastructure which aims at significantly improving the process of developing system-level design flows. Section 6 describes related work, and Section 7 concludes the paper.

2 The Daedalus Design Flow

In Figure 1, the conceptual design flow of the Daedalus framework is depicted. As mentioned before, Daedalus provides a single environment for rapid system-level architectural exploration, high-level synthesis, programming and prototyping of multimedia MP-SoC architectures. Here, a key assumption is that the MP-SoCs are constructed from a library of pre-determined and pre-verified IP components. These components include a variety of programmable and dedicated processors, memories and interconnects, thereby allowing the implementation of a wide range of MP-SoC platforms.

Starting from a sequential application specification in C, the KPNgen tool [6] allows for automatically converting the sequential application into a parallel Kahn Process Network (KPN) [7] specification. Here, the sequential input specifications are restricted to so-called static affine nested loop programs, which is an important class of programs in, e.g., the scientific and multimedia application domains. By means of automated source-level transformations [8], KPNgen is also capable of producing different input-output equivalent KPNs, in which for example the degree of parallelism can be varied. Such transformations enable application-level design space exploration.

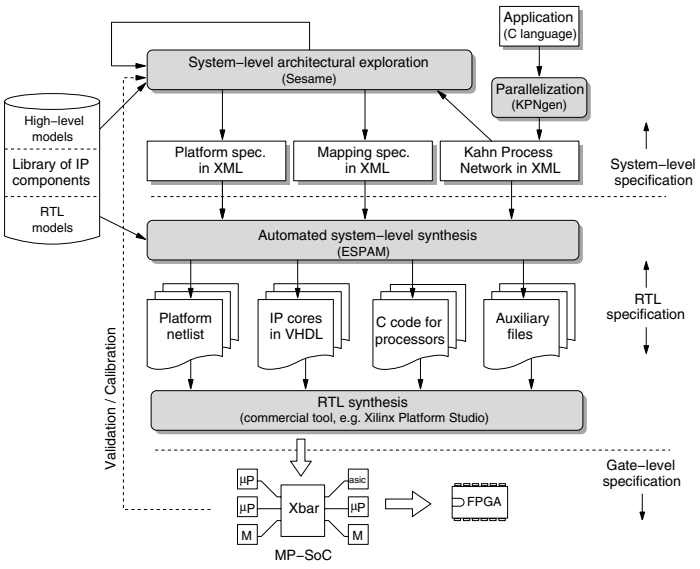

Fig. 1. The Daedalus design flow

The generated or handcrafted KPNs (the latter in the case that, e.g., the input specification did not entirely meet the requirements of the KPNgen tool) can subsequently be used by our Sesame modeling and simulation environment [9,10] to perform system-level architectural design space exploration. To this end, Sesame uses (high-level) architecture model components from the IP component library. Sesame allows for quickly evaluating the performance of different application to architecture mappings, HW/SW partitionings, and target platform architectures. Such exploration should result in a number of promising candidate system designs, of which their specifications (system-level platform description, application-architecture mapping description, and application description) act as input to the ESPAM tool [11,12]. This tool uses these system-level input specifications, together with RTL versions of the components from the IP library, to automatically generate synthesizable VHDL that implements the candidate MP-SoC platform architecture. In addition, it also generates the C code for those application processes that are mapped onto programmable cores. Using commercial synthesis tools and compilers, this implementation can be readily mapped onto an FPGA for prototyping. Such prototyping also allows for calibrating and validating Sesame's system-level models, and as a consequence, improving the trustworthiness of these models.

3 Daedalus' Software Infrastructure

Daedalus does not only consist of the three core tools KPNgen, Sesame and ESPAM, but also features several supporting tools to improve the user-friendliness, and therefore also the deployability, of the framework. This section provides an overview of Daedalus' software infrastructure.

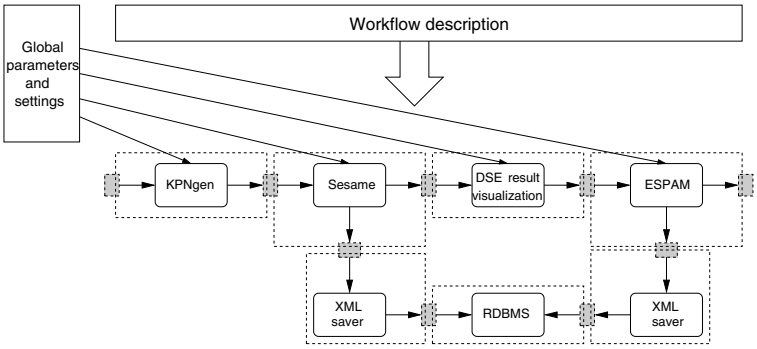

Fig. 2. Building design flows with Daedalus

Integrated RDBMS. In Daedalus, most design information (e.g., structural descriptions of the application, architecture, and the mapping of the former onto the latter) as well as experimental results are described using XML-based descriptions. Daedalus therefore contains the Oracle Berkeley DB XML relational database management system (RDBMS) to store all information (models, parameters and results) related to designs and experiments. Daedalus also features a graphical user interface (GUI) to this RDBMS, which provides the designer with a powerful tool to e.g. explore and visualize the large amounts of data generated by Daedalus' design space exploration. Moreover, it guarantees the reproducibility of experiments at all times.

Workflow control. The vision behind the Daedalus software infrastructure is that it should be open for integration of new tools as well as that it should allow for customization of the design flow. Therefore, the design flow (or tool flow) in Daedalus is composable and constructed from 'design flow blocks'. These design flow blocks, which are illustrated as the dashed boxes in Figure 2, are the tools that take part in the design flow together with their input- and output descriptions. The latter descriptions, illustrated by the gray boxes in Figure 2, provide information about what input/output data a tool consumes/produces and from/to where it reads/writes this data. This allows us to describe a design flow as a simple composition of the design flow blocks, specified in the *workflow description*. For example, Figure 2 shows a design flow which includes a visualization block to visualize Sesame's DSE results and which stores both the DSE and ESPAM's prototyping results in the RDBMS (using the so-called 'XML saver' tool). Evidently, this composability of the design flow allows for easily adding new design steps to a design flow, as well as to customize design flows for specific design domains.

Control and monitoring of MP-SoC prototypes on FPGAs. We have also developed control and monitoring software utilities to facilitate the process of setting up and executing experiments on the FPGA-based prototypes of MP-SoCs generated by Daedalus. Such utilities are necessary and very useful for: (i) conducting an effective and efficient design space exploration at implementation level of abstraction with 100% accuracy on a narrow design space defined by Sesame; (ii) measuring real performance and cost

numbers used for calibration of the Daedalus' high-level architecture models [13]; (iii) preparing real HW/SW demonstrators. The control and monitoring utilities include a configuration manager, an execution control panel, and an on-line monitoring console, all supported by a GUI which allows users, unfamiliar with the FPGA prototyping board, to perform experiments with the MP-SoCs. The configuration manager is used to setup the prototyping FPGA board for a given experiment. The execution control panel allows to define and execute a sequence of instructions (e.g. initialize, start, stop, etc.) that control the interaction of the MP-SoC prototype with the surrounding environment (e.g. the user). The on-line monitoring console displays and stores the data streams that go in and out of the MP-SoC, the content of the status registers of the MP-SoC prototype, and the content of timers and counters that measure the real performance of the prototype.

The Open Source philosophy of Daedalus. The entire Daedalus framework has been developed as high-quality software distributed under Open Source licenses such as GPL or CPL (see `http://daedalus.liacs.nl/Site/Download.html`). This provides many advantages and opportunities (e.g., more easy take-up of the technology since no expensive licenses are required, possible world-wide contributions to the technology, etc.) but it also poses challenges related to software maintainability and tool interoperability. For example, regarding the maintainability, we have developed a configuration and installation utility for the whole Daedalus software framework. At a glance, this task seems to be trivial but our experience shows that it is not, especially when our goal is a fully automated installation process on all major Linux OS distributions. Daedalus consists of many tools that depend on other tools and libraries that have to be installed because they are not available on all or some of the Linux distributions. Identifying, documenting, and maintaining all these tool and library dependencies is a continuous process.

4 Lessons Learned: The Tool Interoperability Problem

A central problem for any design flow addressing the development of embedded systems is that it typically consists of a number of tools that need to inter-operate with each other for the design flow to be efficient and effective. From the experience with Daedalus, we found that tool interoperability is a major problem, which consumes an unnecessary amount of (software engineering) effort. In general, the lack of support for achieving interoperability between tools is becoming one of the big showstoppers for the much-needed productivity improvement in the embedded systems design area, which may seriously endanger the ability to cope with the rapidly growing design complexity. This lack of good tool-infrastructure is a problem that both concerns the embedded systems industry as well as academia. Many research groups develop algorithms and solutions for specific design problems and issues that are not (yet) addressed by commercial tool providers. But since commercial tool providers often refrain from publishing interfaces to their tools, research groups are typically left with the only option of building tool support for the whole design flow themselves, including very basic elements such as editors, graphical UI's, etc. Daedalus was no exception here. Also since there is no common well-defined notion of tool infrastructure, research groups find it

often difficult to cooperate on tool research and development. The flow of ideas from academia to industry is also made more difficult, because it is difficult to deploy new algorithmic innovations into industrial design flows since the tools are not interoperable in any reasonable way.

Moreover, there do not exist good standard case studies and benchmarks for system-level design. We believe that this is due to the fact that (too) much effort is spent on the tool-building part of system-level design research projects instead. This tool development typically involves a significant (software) engineering effort, at the cost of the scientific content of such projects. With a tool infrastructure that fosters the re-use of design tools, this effort could be redirected to the development of good benchmarks and case studies. This would invigorate design flow research as it enables the comparison of research results. Currently, such a comparison of the various achievements in system-level research is not or hardly possible. Finally, we believe that good benchmarks and case studies will provide profit to the flow of ideas from academia to industry, because the design flow improvements can be demonstrated on industrially relevant examples, thus making them much more realistic.

5 Towards a Common Design Flow Infrastructure

To address the tool interoperability problem in system-level design flows, we argue that it is highly desirable to have a tool infrastructure that supports system-level design flows. This infrastructure, which should go beyond efforts such as OCP-IP [14] and SPIRIT's IP-XACT [15], would be a kind of meta-tool for developing system-level design flows, having design flow steps as "plug-ins". This requires the definition of standardized tool, model and data descriptions and file formats to allow the interchange of information between the framework and external tools (i.e., plug-ins). Moreover, the framework should also allow for explicitly defining design flows. This will make it possible to build pre-packaged standardized or customized design flows.

This Common Design Flow Infrastructure (CDFI), that should facilitate the construction and/or adaptation of complex system-level design flows, is conceptually shown in Figure 3. Central to the CDFI is a repository on which all participating tools operate and in which the key elements of system-level design flows (such as application specifications, application and architecture models in various models of computation and at various levels of abstraction, input/output data, simulation results, IP blocks, etc.) are stored in a structural manner. The tools that participate in the CDFI and operate on its repository could either directly belong to the implemented design flow, or have a more supporting role such as translators that, e.g., perform model refinement or translation between models specified in different models of computation.

For each tool that wishes to participate in the flow and thus operate on the CDFI repository, one needs to formally specify its preconditions and input requirements, its semantics, and its postconditions and output specification. To give a few examples, please consider Figure 3. Here, the input/output specification for a tool like KPNgen [6] could specify that it requires "sequential static affine nested loop programs" as input, and produces parallel specifications in the form of Kahn Process Networks (KPNs) [7]. Naturally, such specifications need to be formalized and should be based on a model

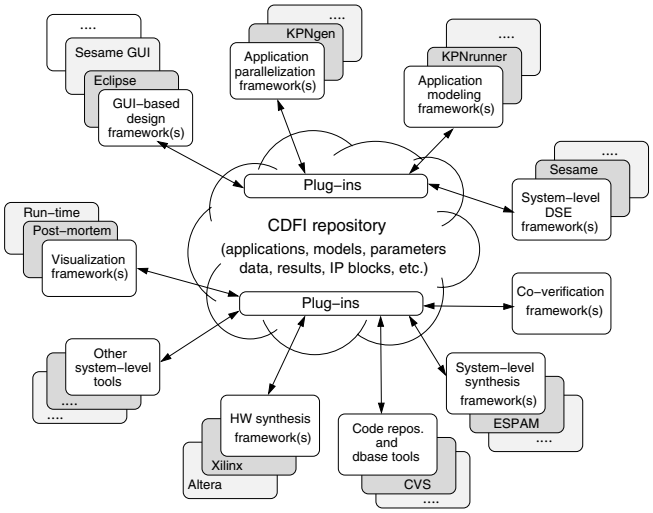

Fig. 3. The CDFI allows the interconnection of System-level Design tools

with clear denotational semantics. Subsequently, for a DSE tool like Sesame [9], it must be specified that it needs application models in the form of KPNs and generates a multitude of performance metrics for a range of architectural implementation instances. Finally, for a visualization tool it may be specified what performance metrics it needs in order to perform post-mortem or run-time visualization of these data. This also means that the output of tools as well as the input parameters must be structurally stored (i.e., described using meta-data) in the CDFI repository in order to allow other tools (such as visualizers) to relate these data, e.g., visualizing cycle-counts, component utilization, etc. for simulation runs with different input parameters. Moreover, we also need to formally relate the already existing models in the CDFI with the models used by any new tool. This specification will show if and how the new tool can be used in combination with other tools in a CDFI-based design flow.

To actually allow for interoperability between different plug-in tools, these tools must have a common understanding of the exact semantics of the CDFI repository elements (e.g., models, data, IP components) they use. This requires standardization with respect to the specification of these repository elements. For example, standardized model specifications (i.e. metamodels) should provide the means to formally relate models and to perform model translations (e.g., via a plug-in translator tool) when required. Essentially, we are looking for a type-system for the tools and their models, in which for example model translations can be seen as type casts. Clearly, developing methods for describing the semantic and input/output behavior of tools as well as for specifying the elements used by these tools (e.g., data, models, IP components) – all with the aim of tool interoperability in mind – is still a formidable research challenge. Evidently, for the actual implementation of the CDFI, existing software technology from e.g. the Model Driven Architecture (MDA) domain could be exploited.

So far, we have only described the CDFI from the perspective of individual tools. In addition, a coordination framework is needed that coordinates the control- and dataflow between the different tools that take part in the design flow. In other words, this coordination framework basically specifies a projection of the implemented design flow on top of the generic CDFI repository and associated tools. Because we explicitly separate the tools and tool specifications (semantic and input/output description) on one hand, and the coordination of the tools to form a specific design flow on the other hand, it should be fairly easy to construct new design flows by re-using tools, extend a design flow, and/or substitute certain tools in a design flow with other tools. Evidently, the development of such a coordination framework (accounting for the specification of control- and dataflow between the different tools in the flow in a generic and flexible manner) also requires substantial research, which could e.g. be inspired by the extensive research that has been performed on workflow frameworks in the domain of Grid Computing and eScience (e.g., [16]).

6 Related Work

Systematic and automated application-to-architecture mapping has been widely studied in the research community. The closest to our work is the Koski MP-SoC design flow [17]. Koski also provides a single infrastructure for modeling of applications, automatic architectural design space exploration, and automatic system-level synthesis, programming, and prototyping of selected MP-SoCs. But unlike Daedalus, Koski does not allow for parallelization of applications, nor design space exploration at application level. Koski requires applications to be specified by hand in UML. The Abhainn design framework [18] has similar objectives as Daedalus, but appears to lack automation for several design steps, such as automated parallelization of applications (applications are modeled using multidimensional arrayed synchronous dataflow specifications), automated design space exploration, and full-fledged MP-SoC synthesis. Other examples of related work can be found in [19,20,21,22]. However, these efforts are limited to processor-coprocesor architectures [19], only provide a limited degree of automation [20,21], or do not provide an automated step towards the register transfer level [22].

Companies such as Xilinx and Altera provide design tool chains attempting to generate efficient implementations starting from descriptions higher than (but still related to) the register transfer level of abstraction. The required input specifications are still so detailed that designing a single processor system is still error-prone and time consuming, let alone designing alternative multiprocessor systems. In contrast, Daedalus raises the design to an even higher level of abstraction allowing the exploration, design and programming of multiprocessor systems in a short amount of time.

With respect to our CDFI ideas, there are a number of related efforts. OCP-IP [14] is an industrial/academic initiative dedicated to proliferating a common standard for intellectual property (IP) core interfaces, or sockets, that facilitate ”plug and play” System-on-Chip (SoC) design. Similarly, the SPIRIT consortium [15] aims at ”Enabling Innovative IP Re-use and Design Automation”. It has defined an XML schema (called IP-XACT) for meta-data that documents the characteristics of IP required for the automation of the configuration and integration of IP blocks as well as APIs to make this

meta-data directly accessible to automation tools. Both the OCP-IP and SPIRIT initiatives focus on standardization with respect to IP blocks, while CDFI goes beyond that by targeting integration and standardization of not only IP blocks but also design tools and tool-flows that cover all aspects of the system design automation. The MoBIES initiative [23] studies model-driven approaches (or model-integrated approaches) to design flows. The goal is to develop new methods and tools that will increase the productivity of the designers. In a sense, the goal is the same as in the CDFI, but the means are different. The CDFI aims at increasing productivity by taking away the bottleneck caused by bad tool interoperability, whereas MoBIES tries to find new ways (i.e. methods) to build embedded systems. The CDFI approach is method neutral, it should increase the productivity of any method.

7 Conclusions

In this paper, we presented our Daedalus system-level design framework for multimedia MP-SoCs from a software perspective, describing how its tools interoperate and discussing the supporting tool infrastructure that improves the actual deployment of the design flow. We also discussed the lack of support for achieving tool interoperability that we have encountered during the development of Daedalus, and presented several initial ideas of future research directions to address this issue. More specifically, we argued that a so-called Common Design Flow Infrastructure (CDFI) for system-level design flows is needed, which surpasses ongoing efforts in this direction, in order to improve and stimulate research and development in the area of system-level design methodology. Such a CDFI would, among other things, heavily reduce the software engineering overheads in system-level design flow projects, enable the comparison of design methodologies/techniques between researchers, and enhance the knowledge transfer from research to industry.

Acknowledgements

We wish to acknowledge that the presented CDFI ideas were developed in cooperation with Timo D. Hämäläinen from Tampere University of Technology and Johan Lilius from Åbo Akademi University.

References

1. Martin, G.: Overview of the MPSoC Design Challenge. In: Proc. Design Automation Conference (DAC 2006), San Francisco, USA (2006)
2. Mihal, A., Keutzer, K.: Mapping concurrent applications onto architectural platforms. In: Networks on Chips, pp. 39–59. Kluwer Academic Publishers, Dordrecht (2003)
3. Thompson, M., Stefanov, T., Nikolov, H., Pimentel, A.D., Erbas, C., Polstra, S., Deprettere, E.F.: A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs. In: Proc. of the Int. Conference on Hardware-Software Codesign and System Synthesis (CODES+ISSS 2007), pp. 9–14 (2007)

4. Nikolov, H., Thompson, M., Stefanov, T., Pimentel, A.D., Polstra, S., Bose, R., Zissulescu, C., Deprettere, E.F.: Daedalus: Toward composable multimedia MP-SoC design. In: Proc. of the Design Automation Conference (DAC 2008) (2008)
5. Daedalus system-level design, http://daedalus.liacs.nl/
6. Verdoolaege, S., Nikolov, H., Stefanov, T.: PN: a tool for improved derivation of process networks. EURASIP Journal on Embedded Systems (2007) doi:10.1155/2007/75947
7. Kahn, G.: The semantics of a simple language for parallel programming. In: Proc. of the IFIP Congress, vol. 74 (1974)
8. Stefanov, T., Kienhuis, B., Deprettere, E.F.: Algorithmic transformation techniques for efficient exploration of alternative application instances. In: Proc. of the Int. Symposium on Hardware/Software Codesign (CODES), pp. 7–12 (2002)
9. Pimentel, A.D., Erbas, C., Polstra, S.: A systematic approach to exploring embedded system architectures at multiple abstraction levels. IEEE Trans. on Computers 55, 99–112 (2006)
10. Erbas, C., Pimentel, A.D., Thompson, M., Polstra, S.: A framework for system-level modeling and simulation of embedded systems architectures. EURASIP Journal on Embedded Systems (2007) doi:10.1155/2007/82123
11. Nikolov, H., Stefanov, T., Deprettere, E.F.: Multi-processor system design with ESPAM. In: Proc. of the Int. Conf. on HW/SW Codesign and System Synthesis (CODES+ISSS 2006), pp. 211–216 (2006)
12. Nikolov, H., Stefanov, T., Deprettere, E.F.: Systematic and automated multi-processor system design, programming, and implementation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 27, 542–555 (2008)
13. Pimentel, A.D., Thompson, M., Polstra, S., Erbas, C.: Calibration of abstract performance models for system-level design space exploration. Journal of Signal Processing Systems for Signal, Image, and Video Technology 50 (2008)
14. OCP-IP, http://www.ocpip.org/
15. SPIRIT, http://www.spiritconsortium.org/
16. Ludäscher, B., et al.: Scientific workflow management and the kepler system. Concurrency and Computation: Practice & Experience 18 (2006)
17. Kangas, T., et al.: UML-based multi-processor SoC design framework. ACM Trans. on Embedded Computing Systems 5, 281–320 (2006)
18. McAllister, J., Woods, R., Fischaber, S., Malins, E.: Rapid implementation and optimisation of DSP systems on FPGA-centric heterogeneous platforms. Journal of Systems Architecture 53, 511–523 (2007)
19. Stefanov, T., et al.: System design using Kahn process networks: The Compaan/Laura approach. In: Proc. of the Int. Conference on Design, Automation and Test in Europe (DATE), pp. 340–345 (2004)
20. Rutten, M.J., et al.: A Heterogeneous Multiprocessor Architecture for Flexible Media Processing. IEEE Design & Test of Computers 19 (2002)
21. Lyonnard, D., et al.: Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip. In: Proc. of the Design Automation Conference (DAC 2001) (2001)
22. Gerstlauer, A., Gajski, D.: System-level abstraction semantics. In: Proc. 15th Int. Symposium on System Synthesis (ISSS 2002), pp. 231–236 (2002)
23. MoBIES, http://ptolemy.eecs.berkeley.edu/projects/mobies/