Predictable Dynamic Embedded Data Processing

Marc Geilen[1], Sander Stuijk[1], Twan Basten[1,2]
[1] Eindhoven University of Technology, Department of Electrical Engineering, Eindhoven, The Netherlands
[2] Embedded Systems Institute, Eindhoven, The Netherlands
{m.c.w.geilen, s.stuijk, a.a.basten}@tue.nl

***Abstract*—Cyber-physical systems interact with their physical environment. In this interaction, non-functional aspects, most notably timing, are essential to correct operation. In modern systems, dynamism is introduced in many different ways. The additional complexity threatens timely development and reliable operation. Applications often have different modes of operation with different resource requirements and different levels of required quality-of-service. Moreover, multiple applications in dynamically changing combinations share a platform and its resources. To preserve efficient development of such systems, dynamism needs to be taken into account as a primary concern, not as a verification or tuning effort after the design is done. This requires a model-driven design approach in which timing of interaction with the physical environment is taken into consideration; formal models capture applications and their platforms in the physical environment. Moreover, platforms with resources and resource arbitration are needed that allow for predictable and reliable behavior to be realized. Run-time management is further required to deal with dynamic use-cases and dynamic trade-offs encountered at run-time. In this paper, we present a model-driven approach that combines model-based design and synthesis with development of platforms that support predictable, repeatable, composable realizations and a run-time management approach to deal with dynamic use-cases at run-time. A formal, compositional model is used to exploit Pareto-optimal trade-offs in the system use. The approach is illustrated with dataflow models with dynamic application scenarios, a predictable platform architecture and run-time resource management that determines optimal trade-offs through an efficient knapsack heuristic.**

I. Introduction

Cyber-physical systems (CPS) are computer systems that are inherently involved in a continuous interaction with their physical environment. Requirements for such systems are to a large degree extra-functional and in particular timing is very important. The integrated CPS is often designed or developed from a model that includes both the computer system to be developed and the physical environment that is to be controlled or interacted with. Such models could be made for instance in Matlab or Simulink. Timing behavior and requirements can be established from the models, but need to be reliably realized in the developed system.

Model-based design approaches intend to realize an automated trajectory from a model in some formalism to an implementation that conforms to the model, in terms of its functional behavior, its interface or its timing behavior. Since systems can be large and consist of separate, interacting components, such strategies would need to be compositional in the sense that individual components can be developed or synthesized separately, according to their own model or definition and the developed components can be combined to form the overall system in such a way that the composition works and the behaviors of the component remain respected. This is challenging, because generally the components may have undesirable or unpredictable interactions through shared resources. Most platforms are designed such that these interactions do not impact functional behavior, but timing behavior is often more susceptible to unpredictable interactions. The nature and impact of the interactions are often hard to predict and only manifest themselves in the final realization or after deployment of the product in the field.

This work was supported in part by Agentschap NL (an agency of the Dutch Ministry of Economical Affairs), as part of the EUREKA/CATRENE/COBRA project under contract CA104.1

Hardware/software platforms can be designed to alleviate these problems, but this is challenging as this may lead to shared resource arbitration schemes with low utilization and an increase in the amount of resources required and hence increasing system cost. It is obviously important to get this trade-off right. Different aspects of timing behavior may be targeted. Some approaches focus on the ability to provide guaranteed worst-case behavior, for instance response times, on platform resources, so as to be able to statically guarantee worst-case system behavior irrespective of how and with what other applications resources may be shared. This is often called *predictability* [1], which may also cover other aspects than the worst-case behavior. Another aspect, often called *composability* [2], intends to realize platforms that guarantee freedom of any interaction whatsoever from other applications that may be sharing the same resources. An important benefit of this approach being that applications can be verified independently from one another and the combination of application on a shared platform is guaranteed not to introduce new unforeseen errors. *Repeatability* [3], [4] refers to platform properties that guarantee that different executions of the same system, provided with the same input, will provide the same output. This also facilitates verification and testing. Another approach to repeatability which involves the platform, although it extends beyond only the platform, enforces a certain degree of *conformance between model time and physical time* [4].

A challenge to getting predictable/repeatable/composable systems is the fact that modern systems have a tendency of becoming more and more dynamic in their behavior. Advanced compression schemes and multi-resolution displays in digital media lead to huge differences in workload among different media streams, but also within a single media stream. Highly optimized wireless transceivers and cognitive radio in software-defined-radio systems switch between different modes depending on conditions of the wireless channel. High performance feedback-control systems operate at different sampling frequencies and switch between different modes of operation. Consumer equipment needs to support downloadable and upgradeable applications. With such dynamism, systems cannot be designed for the worst-case and need to support scalability and trade-offs between quality of operation and resource usage. Run-time resource management is required for sharing of the resources and to exploit the available trade-offs in application configurations to efficiently use the platform.

This paper presents and illustrates a model-driven approach to the design and development of stream-based CPSs. It is based on the use of predictable, repeatable and composable platforms and exploits a run-time management approach to dynamically handle run-time use-cases. Stream-based applications and their platform mappings are formally modeled by a dynamic extension of timed synchronous dataflow, while Pareto-optimal trade-offs within applications, their quality-of-service and their resource usage are captured in a formal, compositional model of trade-offs. The approach is illustrated using a software-defined radio implementation of a WLAN transceiver with dynamic application scenarios on a predictable platform architecture under control of a run-time management system. Run-time optimal configurations are determined using an efficient heuristic for multi-dimensional, multiple-choice knapsack problems.

The paper proceeds as follows. The following section gives some pointers to related work. Section III then gives an overview of our approach. Section IV discusses the temporal model we use as a conservative abstraction. We discuss the targeted hardware and software platforms in Section V. Section VI discusses how run-time dynamics and use cases can be handled. Section VII concludes.

II. Related Work

We describe a model-based design approach where the modeling pertains in particular to timing and performance. Some other works share this idea. The Ptides project [4] proposes a design approach and hardware / software architecture in which cyber-physical systems can be realized composably and repeatably by strictly adhering to the timing behavior of the semantic model in the implementation and prohibiting any impact of implementation related timing aspects on the functional behavior. Combined with static analysis, a repeatable and composable system can be made. BIP [5] is a formal component based lapproach in which formal models specify the interface behavior of components. Rigorous verification that checks whether the specified interfaces are respected make the component based design approach composable. Giotto [6] is a specification language that uses the time-trigged [2] perspective to rigorously define the timing behavior of software components. Static execution time and schedulability analysis ensure that specified timing behavior is met.

Real-Time Calculus (RTC) [7] and Timed Dataflow [8] are models that are often used for performance analysis of stream-based applications and platforms. In RTC, application requirements are specified in terms of arrival density of events or workloads and resources are described in terms of the amount of service they deliver to an application in a given time interval. Timed dataflow actors are used both to describe application tasks as well as the temporal behavior of the resources onto which the tasks are mapped.

[9], [10] describe predictable design flows using dataflow models to represent resource allocations. [9] uses constraint solving to find solutions that satisfy requirements, [10] is based on SAT solving techniques. Both approaches determine throughput optimal mappings using static, single-rate models. [11] describes an approach which is similar to ours in the models that are used and platforms that are considered. The models are more static and no run-time reconfiguration is used.

Architectural concepts to support composable and predictable realizations are discussed, among others in [2], [12], [13]. The Time-Triggered Architecture [2] is a scheduling and arbitration architecture that is used for scheduling and arbitration of networks as well as processing resources. CompSOC [12] is a platform that focuses on composable realizations. It combines solutions for composable use of on-chip networking, processor scheduling and external memory access control. The PRET project [13] focuses primarily on building repeatable processor and memory architectures. Repeatability is primarily achieved through the use of composability in processor sharing, scheduling, interrupts, caches, and accesses to shared external memory.

Some works share our use of compositional trade-offs in design and run-time phases. [14] explores system-level trade-offs at design-time. [15] explores trade-offs between time and memory usage for memory architectures for data storage and transfer. For run-time trade-offs, [16], [17], [18] present heuristics for run-time configuration selection for multiple applications with design-time evaluated resource-usage trade-offs on shared resources.

III. Overview of the Approach

In this section, we give an overview of the elements needed in an approach for predictable embedded data processing. The individual elements are discussed in more detail in the following sections.

The central concept for dealing with dynamic behavior is a *scenario* [19]. Dynamic behaviors within an application are divided in a limited number of so-called scenarios. Behaviors within a single scenario are similar in their resource usage, such that a corresponding static resource allocation can be made. To exploit this allocation, it is required that the behavioral scenarios are detectable or predictable and the platform may allocate the appropriate resources. For a wireless radio transceiver scenarios could be the different MIMO configurations used, or, at a smaller time granularity, synchronization, header processing, receiving payload, etcetera.

To describe the applications, we use a dataflow based model, where an application is divided in concurrent actors, which synchronize and communicate with each other. Beside the functional behavior of the actors, actors are annotated with their worst-case execution times (which may depend on the mapping when different mappings are possible). A dynamic, but statically analyzable extension of the synchronous dataflow model has been developed to support dynamism in the form of scenarios: Scenario-Aware Data Flow (SADF) [20], [21]. SADF is introduced in more detail in Section IV.

The annotation with worst-case execution time allows the prediction of lower bounds on the performance of the application for a given mapping and thus supports the mapping

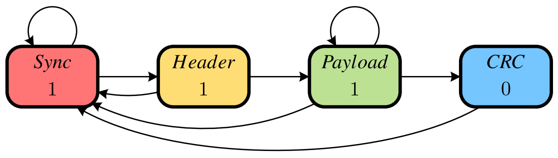

Fig. 1. Scenario FSM of WLAN receiver

process to find one or more mappings per scenario which guarantee that the required performance is met. In order to achieve this, timed dataflow models are used that conservatively represent the timing behavior of the architectural components of the platform; those models are composed with the timed dataflow models of the application itself and used together to evaluate the performance.

To support the above approach to predictability, it is required that the platform hardware/software architecture supports the generation of conservative timed dataflow models *per application*, despite the fact that platform resources will be shared at run-time with other applications. Such predictable platforms are discussed in Section V. We use a tile-based chip multiprocessor system, where the tiles are connected by a network-on-chip. Time-division multiplexing (TDM) based resource arbitration of the various resources makes the platform predictable and composable. External memory may be used as long as it is done through a predictable memory interface which permits a timed-dataflow conservative abstraction.

An automated design-flow produces different configurations in which the different scenarios of the application can be mapped so as to provide flexibility at run-time for combining multiple applications on the platform. Moreover, in case the application allows it, configurations may provide trade-offs between quality-of-service levels of the application versus resources required to provide the service. A run-time quality and resource management can use the predefined configurations and their quality and resource trade-offs to determine optimal configurations for the dynamic use cases arising at run-time. A formal, compositional model for computing trade-offs, called Pareto Algebra [22], is used for this. This run-time management is the subject of Section VI.

IV. Modeling Applications with Scenario-Aware Dataflow

Applications consist of concurrent tasks that are mapped onto different processing resources of the platform, general purpose processors, DSPs, vector processors and so on. For some tasks, different mapping options may be feasible. The application behavior is partitioned into a set of scenarios and the individual scenarios are characterized in terms of their worst-case execution times and resource requirements. The result of this process is an application model called Scenario-Aware Data Flow (SADF) [20].

An SADF model consists of a set of scenarios of behavior and specifies in what order scenarios may occur. This order is specified by means of a finite state machine. The states of the state machine are labeled with scenarios (there can be different states labeled with the same scenario) and any (infinite) path through the state machine represents a possible sequence of scenarios. Figure 1 shows such a finite state machine (FSM) for the WLAN receiver model from [23] (in fact it over-approximates the possible scenarios, as explained below, but this model is conservative and tight enough). The reception of a data packet consists of four phases: synchronization, header processing, payload processing and a CRC check and transmission of acknowledgement. These phases are traversed in the given order, where multiple synchronization attempts may be needed to detect the start of a packet and the payload may consist of an arbitrary number of OFDM symbols (in the model, which over-approximates the real situation, in which it is limited to 256). The packet ends with CRC and acknowledgement. Synchronization may be lost at any time causing a return to the synchronization phase.

The throughput constraint for the graph is that the source produces 1 OFDM symbol every $4.0\mu s$, which must be timely handled by the application. The *Sync*, *Header* and *Payload* scenarios each process 1 symbol, but the *CRC* scenario does not process any new input. To capture this in the SADF model, the scenarios are labeled with the amount of progress made, called the scenario *reward* [24]. Hence the reward is 1 (symbol) for all scenarios except *CRC*, which is extra work to be done at the end of the packet, not related to any new input, for which the reward is 0. The throughput of the graph is then defined as the worst-case amount of progress or reward per time unit. Besides throughput constraints, there are latency constraints on the transmission of the acknowledgement in the *CRC* scenario, but we ignore that aspect for this paper.

For worst-case analysis to guarantee predictable behavior, we need to analyze all possible scenario sequences. For average performance analysis, the transitions of the scenario state machine can be annotated with probabilities, effectively turning it into a Markov Chain. In that case, scenario sequences are also provided with a likelihood of occurrence.

The behavior inside individual scenarios may be complex and exhibit complex dependencies on other scenarios. It is specified by static dataflow models, such as an SDF or CSDF model. An example is shown in Figure 2, which shows the dataflow graph of the *Payload* scenario. (For the moment we ignore the dashed edges.) The *Src* actor models the RF source which periodically produces the data for an OFDM symbol. The token *src* captures the dependency on the previous production from the source. The *Shift* actor aligns the data received from *Src* with the actual start of an OFDM symbol. The *shift* token captures the dependency on the update of the shift value after the previous symbol, which determines the proper shift to align the OFDM symbols. Token *dem_pars* represents the availability of the demodulation parameters to be used by the payload demodulation, modeled by the *Pdem* actor. Demodulation is followed by decoding by the *Pdec* actor. Actor *DOut* processes the data contained in the payload. The *Payld* actor is an artificial actor whose firing signals the end of the decoding of the payload, which is a dependency to following scenarios; *DOut* may be executed concurrently. This dependency is recorded in the *payload* token, which is used in the *CRC* scenario model to represent the dependency of the CRC computation on the decoding of all payload data.

It is important that the tasks that are modeled by actors in the SADF are annotated with worst-case execution times.

They may have alternative mappings, in which case they have a timing annotation for each possible mapping option. In this way, the SADF model provides an abstract timed dataflow model that is guaranteed to be a conservative abstraction, formally in the sense of [25], of the application. This means that given the same (or earlier) timing of the input provided to the application, the application will produce its outputs no later than specified by the abstract model. Compared to traditional static timed dataflow models such as timed SDF or timed CSDF, the use of scenarios allows one to make tighter worst-case timing models and still allows for static analysis of throughput and latency properties.

Dataflow performance analysis techniques assume an abundance of resources. Processing resources are assumed to be available whenever an actor can execute and edges have an infinite storage capacity and they do not cause any delay. Scheduling decisions or communication delays can be modeled explicitly into the graph. For example, some of the actors in the WLAN receiver application need to share a processing resource. We assume the firings of these actors are ordered using a static-order schedule. In the *Payload* scenario, the *Shift* and *Pdem* actors need the same embedded vector processor (EVP). The application has a data dependency from the *Shift* actor to the *Pdem* actor. The schedule needs to respect the dependency and therefore starts with the *Shift* actor. The *Pdem* actor is fired next on the EVP. This static-order schedule is modeled in the application graph to ensure that the performance analysis takes the resource sharing into account. In this particular example, the static-order schedule can be modeled by adding two edges to the graph (one from *Shift* to *Pdem* and one in the opposite direction). The resulting dataflow graph is shown in Figure 2, where the dashed edges are due to encoding of the schedules. The initial position of the *evp* token dictates that the static-order schedule of the EVP starts with the *Shift* actor. The token further embodies the dependency between the use of this processor in the *Payload* scenario and the following scenarios in the application; it ensures that any actor firings from different scenarios cannot be overlapping on the EVP. Figure 2 shows also the resource mapping of the *Pdec* and *DOut* actors. Both actors are mapped to a separate processing element (respectively a software codec (SWC) and a control processor (ARM)). This is modeled with the self-edges with the *swc* and *arm* tokens. The actors *Src*, *Payld*, and *Pars* model the input/output behavior and the dependency on availability of demodulation parameters. These actors merely model synchronization constraints due to data dependencies and do not need to be mapped to any processing element. So far we have illustrated how processor dependencies and static-order schedules can be modeled into the dataflow graph. Other implementation aspects such as storage space or communication delays can be modeled in a similar manner in the dataflow graph. These methods are beyond the scope of this paper; interested readers are referred to for example [9], [26], [27].

Figure 3 illustrates an example behavior of the SADF model of the full WLAN application mapped onto a reconfigurable platform, as discussed in detail below. The behavior consisting of three packets with one payload symbol each. In-between

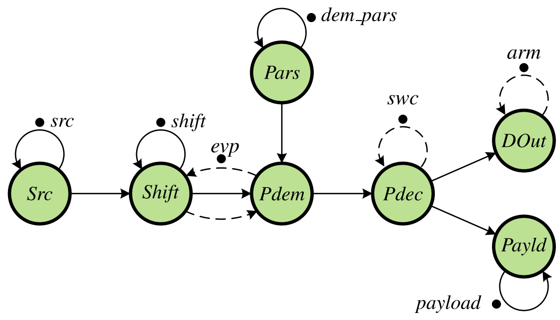

Fig. 2. Dataflow graph of the payload scenario with static-order schedule

packets, reconfigurations are performed, so that packets 1 and 3 are executed in configuration c_1 and packet 2 in configuration c_2. In the semantics of the SADF model, each scenario behavior determines time stamps of the tokens in the model using the dataflow graph of the scenario. The top part of the graph shows in horizontal rows, the tokens in the model. *src* to *dem_pars* represent the availability of the data dependencies and the last three, *evp*, *swc* and *arm* represent the tokens that were added to model the resource mapping. Their time stamps indicate when the resource becomes available to commence with the next scenario. The tokens that together represent the completion of a particular scenario are linked (and shown in the color of that scenario using the scenario color coding of Figure 5 that gives the scenario FSM of the WLAN receiver including reconfigurations). The bottom part of the figure shows the behavior of the SADF model in the form of a Gantt chart. The first row shows the OFDM symbols being received from the radio, each 4μs long and immediately after each other. After the symbol has been received the processing starts and the next three rows show the computation activity of the three computational resources, the EVP, the SWC and the ARM respectively. The purple activities in the EVP represent the time needed for adjusting the operating frequency of the EVP for reconfiguration. From the chart we can observe that during reconfiguration, the processing of packets lags behind. If reconfigurations are not too frequent, the application can catch up and still make the throughput constraint. Note in particular that the subsequent scenarios may overlap in time, and that despite that, they can be nicely individually modeled in the SADF model.

V. Execution Platforms

Applications in cyber-physical systems often have timing requirements. An implementation satisfying the requirements can be derived using a design flow that uses a model-based design approach. Such a design flow takes as input a dataflow model of the application and it determines a resource allocation which ensures that the timing requirements of the application are met. A flow can only provide these timing guarantees when the target hardware platform provides certain features. In order to provide a predictable timing behavior, the platform should provide timing guarantees on the response times of all hardware components, schedulers and arbiters. It should also guarantee some bound on the time needed to execute a

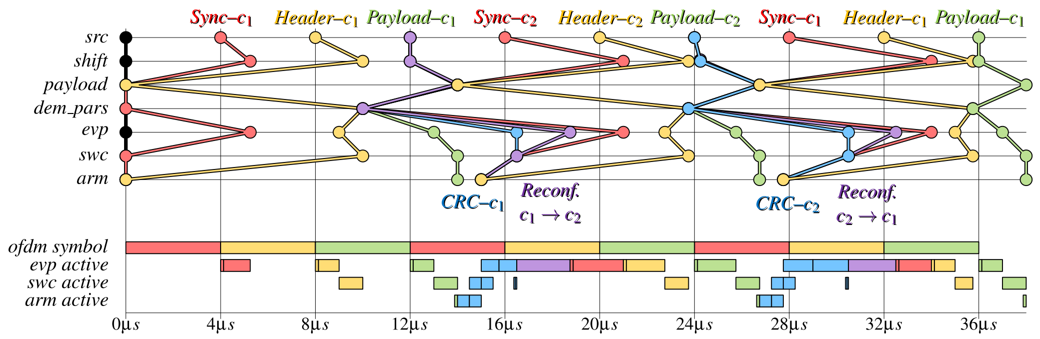

Fig. 3. Execution trace of then WLAN receiver with reconfigurations

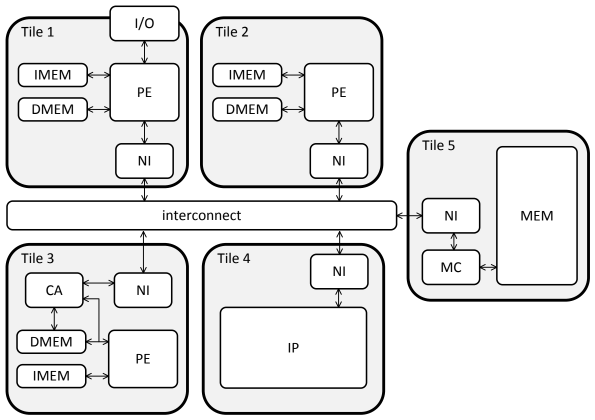

Fig. 4. Tile-based multiprocessor platform

code segment (an actor of the dataflow graph) on a processor even when this processor is shared with actors from different applications. Several such predictable platforms have been proposed in recent years (e.g., MAMPS [28], CompSOC [12], PRET [13]). The precision-timed (PRET) platform focuses on the design of a processing element and memory controller that provide a predictable timing behavior. The MAMPS and CompSOC platform integrate many processing elements and memories into a predictable multiprocessor platform that can be used to implement streaming applications. These last two platforms follow the tile-based multiprocessor platform template described in [29]. In this template multiple tiles are connected through a network interface (NI) to an interconnection network that provides point-to-point connections between tiles. These connections can, for example, be implemented through a network-on-chip or using direct hardwired FIFO connections. The MAMPS and CompSOC platform support several different types of tile. Figure 4 shows a platform instance that contains several of the tile types supported by these platforms. Tile 1 and 2 show simple tile architectures with a processing element (PE) connected to a network interface (NI), local data and instruction memories (DMEM and IMEM), and some optional peripherals (e.g., I/O or timers). Tile 3 shows a similar tile which has been extended with a communication assist (CA) to handle the sending and receiving of data inside the tile. The CA decouples the communication in a tile from the computation taking place on its processing element. This reduces the worst-case response time of the actors on the PE. Tile 4 shows a tile in which a hardware IP block (an accelerator) is directly connected to the interconnect using only a network interface. Tile 5 contains a shared (possibly external) memory (MEM) that can be accessed through a memory controller (MC), which should ensure that memory accesses are handled predictably. It should provide bounds on the time to perform individual load/store operations. Such memory controllers are described for instance in [30], [31].

Multiple applications may be running simultaneously on a cyber-physical system. A system could for example be running two software defined radios concurrently. All applications that share resources in the platform can be given real-time guarantees if the platform provides a predictable timing behavior. The maximal interference from one application on another could is then analyzed and taken into account. However, when an application requires a repeatable timing behavior, independent of other applications, the platform should provide complete temporal isolation between applications. The CompSOC platform addresses this by providing a composable (and predictable) platform [12]. The platform preempts tasks after a fixed period of time. In addition, it delays the completion of a task on a resource until the end of a time slice to prevent tasks that finish early from affecting when the following task is scheduled. Finally, it uses a composable scheduler, such as time-division multiplexing (TDM), where the presence or absence of scheduling requests from one application cannot affect when other applications are scheduled. The components needed to make the platform composable should by themselves be predictable and composable. A composable platform requires that shared resources support preemption. Adding support for preemption on stream processors such as VLIWs, SIMDs, or other processors with a distributed internal state may be very costly in terms of hardware. Some platforms require such processors for computational and energy efficiency; these resources can then not be shared between applications.

The model-based design flow presented in [32] can be used to implement an application modeled with an SADF graph, onto the MAMPS and CompSOC platforms. The design flow analyzes the resource requirements of an application (e.g., buffer sizes needed to meet the timing constraints) and it binds and schedules the actors and edges of the graph onto

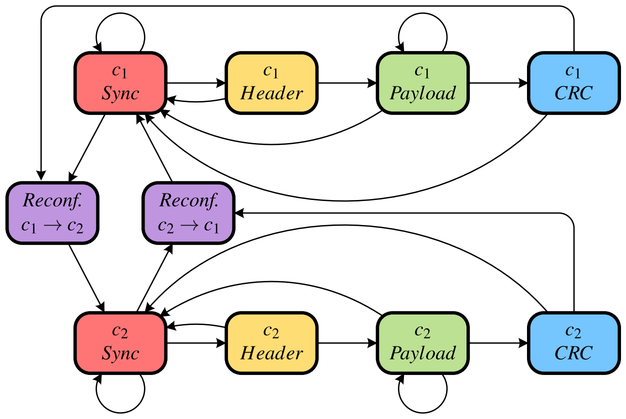

Fig. 5. Scenario FSM of WLAN configurations

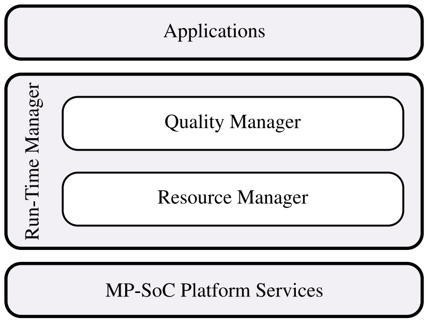

Fig. 6. Run-time manager

the resources in the platform. Whenever the design flow takes a design decision, it is modeled into the dataflow graph, as described in the previous section. By analyzing the timing behavior of the resulting resource-aware graph, using dataflow timing analysis techniques, the design flow is able to validate whether the timing constraints of the application are met under the (partial) resource allocation. The design flow generates a set of resource allocations, called configurations, that provide a trade-off in their resource requirements.

At design time, different mapping options can be explored per scenario behavior. Different options can exploit trade-offs between the resources that are being used and/or the quality-of-service of the application. These different options can be selected at design time, but can also be used at run-time to allow for more flexibility in run-time use cases and for dynamic optimization of resource usage, quality or power consumption. The options are evaluated in terms of different criteria and Pareto optimal configurations are kept.

Switching between different configurations at run-time may require additional operations that may incur a certain overhead. This overhead needs to be taken into account at design time in order to guarantee that performance requirements can still be met under reconfiguration. Figure 5 shows how this can be modeled with SADF for the WLAN case. First of all we see that the application scenarios have been refined with the system scenarios of being able to run the application in two different configurations, called c_1 and c_2. The configurations represent different clock speeds at which the EVP may be run. In c_1, the it runs at 300MHz, in c_2 at 175MHz. Reconfiguration of the EVP clock frequency takes a fixed amount of time. This gives in the model essentially two copies of the scenario state machine, one for each configuration. Switches between configurations are possible, but can only be initiated in-between packets. Moreover, these switches incur overhead which needs to be modeled explicitly; hence there are two separate scenarios that model the switch from one configuration to the next.

VI. Run-time Management

Using the kinds of platforms and conservative application modeling techniques of the previous sections, we can explore different application mapping opportunities and find Pareto optimal mappings. For static systems this would allow one to select a single optimal configuration to be implemented. Many modern systems however show dynamicity at run-time, use cases of simultaneously running applications may change, modes of applications may change, the available resources may change. Such dynamic systems need to be able to adapt at run-time to changes and select, instantiate and enforce optimal configurations.

We discuss an approach for a run-time management system that preserves predictable behavior and combines run-time analysis and decision making with design-time mapping and profiling. Figure 6 illustrates the run-time management infrastructure. The use of the platform services is controlled by a resource manager that allocates resources to running applications. The resource arbiters, implemented on the platform resources, ensure that the actual resource sharing is predictable and that allocated budgets cannot be violated. The resource manager sets resource allocations according to at design-time pre-determined configurations per application. It also manages reconfigurations when an application changes its current configuration.

On top of the resource manager sits a component called Quality Manager in Figure 6. The quality manager is where any ‘intelligence’ in the selection of configurations takes place. It responds to changes such as the user starting a new application and takes decisions on the configurations that applications will be running, based on information about available resources, resource requirements of the application configurations, quality scalability and trade-offs applications may have. An application may allow for trade-offs between the use of different resources, such as different processors, or trade-offs between processing and memory usage or through DVFS between energy consumption and resource utilization. Applications may also be able to trade off quality for resource usage, for instance, a video decoder may trade-off occasional deadline misses for a lower energy consumption.

The configurations determined at design-time are optimal trade-offs between the various criteria of resource usage, energy usage and quality. We use Pareto optimality as a criterion for selection interesting configurations. The configurations are determined on a per-application basis and need to be combined at run-time to investigate system-wide trade-offs. In order to support compositional reasoning about trade-offs

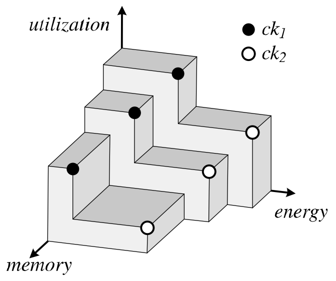

Fig. 7. Pareto-optimal configurations

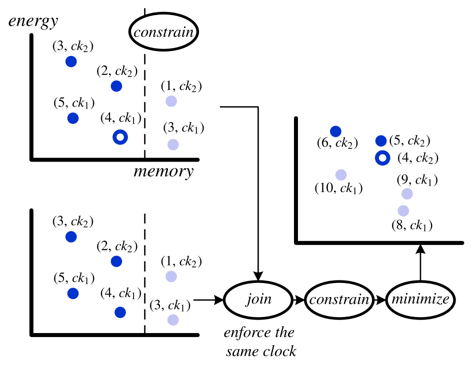

Fig. 8. Pareto-Algebra-based computation of configurations

we developed a calculus of trade-offs. After introducing this calculus, we illustrate how it can be used by the quality manager to determine optimal system configurations by using a fast Knapsack algorithm.

In Section V we saw how the platform may support two different configurations for the receiver and how the reconfiguration process itself can be conservatively modeled, leading to the extended SADF model of Figure 5. The configurations represent different trade-offs between clock speeds/energy and utilization of the EVP processor. A higher clock speed is necessary if the WLAN receiver must be activated simultaneously with other receivers. The graph is guaranteed to conservatively capture both execution and reconfiguration of the receiver. As illustrated with Figure 3, temporal analysis of the graph shows that throughput requirements are met within both configurations, but not when the receiver switches configuration after every frame. This knowledge can be encoded in the quality manager to prevent switching too frequently.

At design time multiple optimal configurations are determined for applications and their individual scenarios. *Pareto optimality* is used to determine which configuration may potentially be useful to exploit at run-time. A configuration is Pareto optimal among a given set of possible configurations if there is no other configuration that bests it in all of the optimization criteria considered. Figure 7 shows a fictitious illustrative example. The circles represent six different configurations which form trade-offs between processor utilization, energy usage and memory. For each of these optimization criteria smaller is better and all six are Pareto optimal. It assumes that the process can be run at two different clock speeds, where the black circles are configurations at a lower clock speed ck_1, resulting in a higher utilization of the processor, and the white circles are configurations at a higher clock speed ck_2. Energy/utilization can be traded for memory.

We assume that the quality manager has for each application scenario a table with the possible configurations and their properties. At run-time, the manager needs to find a compatible selection of the configurations from each of the applications. Resource utilization cannot exceed the total resource availability for each of the resources being managed. Other constraints may apply, for instance that if voltage and frequency of a processor may be scaled, then all applications that have tasks mapped to the processor need to agree on the clock frequency being used. From the individual application trade-offs, the quality manager needs to compose the remaining overall system trade-offs and make a decision about which configurations to instantiate. To compute system trade-offs from configurations we use a generic framework for compositional calculation of system-level Pareto optimal configurations from the Pareto points of components supporting various kinds of composition. The framework is called *Pareto Algebra* [22].

Figure 8 illustrates how trade-offs can be compositionally computed by the quality manager. On the left side, it shows two applications with their configurations as in Figure 7, but here only two dimensions are shown; the third dimension, the number of processors used in the configuration is shown as a number next to the configuration, together with the clock frequency used. For the sake of the example, both applications are chosen to have identical configurations. At run-time, additional constraints may be enforced on individual applications, for instance on the available memory to be used and on the available processors. The dashed line illustrates a memory constraint which makes all configurations to the right of the line infeasible in the particular run-time situation. If this were the only application and the goal is to minimize energy, then the open circle configuration would be selected. Now, for both applications, the two sets of configurations need to be combined. The steps can be easily expressed in Pareto Algebra operations, *join*, *constrain* and *minimize*. We take as a starting point the join of the configurations of both applications on the clock frequency. This operation is similar to a join in relational databases and it considers all combinations of configurations from both applications that agree on the clock frequency used. Since the applications will be sharing the processors, we assume they cannot operate at different frequencies. Subsequently, we annotate the new combined configurations to determine how many processors they use, how much memory and how much energy. On the aggregated data we apply the constraints of memory and available processors to prune any combination that exceeds the available memory or processors. This discards the light blue combined configurations, which use too many processors. We now have a collection of feasible configurations, but not all of them need to be Pareto optimal. Therefore, we apply minimization to determine which combined configurations are optimal. Ultimately, from the remaining Pareto points, one configuration is chosen. The most energy-efficient one, for instance, is the open circle configuration.

The approach is compositional and incremental. In a use case where at some time we have running applications and a new application needs to be added, we have al the optimal configurations of the aggregated running applications and we only need to perform the additional step of adding the new application. Nevertheless, combining configurations of many applications can potentially lead to an exponential growth of the number of available options. In fact, the problem is very much akin to the well-known Multi-dimensional Multiple-choice Knapsack Problem (MMKP) [33], which is known to be NP Complete. Work has been done to realize fast heuristics for run-time configuration selection [17], [16], [18]. [18] describes a heuristic based on the Pareto Algebraic model. The run-time composition of system-level Pareto-optimal configurations from components trade-offs using Pareto-Algebra is a very generic approach. It fits in a design pattern that applies to very different domains [34].

VII. Conclusions

We have discussed our approach to the model-driven, predictable development of cyber-physical systems. It combines an automated design flow based on a performance-conservative timed dataflow model with run-time resource and quality management to handle different use-case scenarios. In order to handle dynamic behavior of modern applications, the timed dataflow model exploits scenarios in the Scenario-Aware Data Flow model. We use predictable hardware/software platforms to be able to make realizations with guaranteed performance. This places constraints on platform components and arbitration schemes that can be used and creates new challenges to improve efficiency while maintaining predictability. The approach is based on flexible configurations that allow available trade-offs to be effectively and efficiently exploited by run-time quality and resource management when dealing with dynamic use-cases.

References

[1] J. Stankovic and K. Ramamritham, "What is predictability for real-time systems?" *Real-Time Syst.*, vol. 2, p. 247254, 1990.

[2] H. Kopetz and R. Obermaisser, "Temporal composability," *Computing & Control Engineering Journal*, vol. 13, pp. 156–162, August 2002.

[3] S. A. Edwards *et al.*, "A disruptive computer design idea: Architectures with repeatable timing," in *Proceedings of International Conference on Computer Design (ICCD), IEEE, Lake Tahoe, CA, 4-7 October, 2009*, 2009.

[4] J. Eidson *et al.*, "Distributed real-time software for cyber-physical systems," *Proceedings of the IEEE (special issue on CPS)*, vol. 100, no. 1, pp. 45 – 59, January 2012.

[5] A. Basu *et al.*, "Rigorous component-based system design using the BIP framework," *IEEE Software*, vol. 28, no. 3, pp. 41–48, 2011.

[6] T. Henzinger, B. Horowitz, and C. Kirsch, "Giotto: A time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 91, p. 8499, 2003.

[7] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *The 27th Annual International Symposium on Computer Architecture (ISCA)*, vol. 4, 2000, pp. 101 – 104 vol.4.

[8] S. Sriram and S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization, Second Edition*. CRC Press, 2009.

[9] A. Bonfietti *et al.*, "An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms," in *Int. Conf. on Design, Automation and Test in Europe, DATE 10, Proc.*, 2010.

[10] W. Liu *et al.*, "Efficient SAT-based mapping and scheduling of homogeneous synchronous dataflow graphs for throughput optimization," in *Real-Time Systems Symp., RTSS 08, Proc.* IEEE, 2008, pp. 492–504.

[11] O. Moreira, F. Valente, and M. Bekooij, "Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor," in *Int. Conf. on Embedded software, EMSOFT '07, Proc.* ACM, 2007, pp. 57–66.

[12] B. Akesson *et al.*, "Virtual platforms for mixed time-criticality applications: The CoMPSoC architecture and SDF3 design flow," in *Proceedings of workshop on Quo Vadis, Virtual Platforms? Challenges and Solutions for Today and Tomorrow*, 2012.

[13] B. Lickly *et al.*, "Predictable programming on a precision timed architecture," in *Int. Conf. on Compilers, architectures and Synthesis for embedded systems, CASES 08, Proc.* ACM, 2008, pp. 137–146.

[14] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for Pareto-optimal configurations in parameterized system-on-a-chip," *IEEE Trans. VLSI Syst.,*, vol. 10, no. 4, pp. 416–422, August 2002.

[15] R. Szymanek, F. Catthoor, and K. Kuchcinski, "Time-energy design space exploration for multi-layer memory architectures," in *Proc. of DATE 2004*. IEEE, 2004, pp. 318–323.

[16] P. Yang and F. Catthoor, "Pareto-optimization-based run-time task scheduling for embedded systems," in *Proc. of CODES+ISSS 2003*, 2003, pp. 120–125.

[17] C. Ykman-Couvreur *et al.*, "Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management," in *Int. Symp. on SoC, Proc.* IEEE, 2006, pp. 1–4.

[18] H. Shojaei *et al.*, "A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for cmp run-time management," in *Design Automation Conf., DAC 09, Proc.* ACM, 2009, pp. 917–922.

[19] S. Gheorghita *et al.*, "System-scenario-based design of dynamic embedded systems," *ACM Trans. on Design Automation of Electronic Systems*, vol. 14, no. 1, pp. 1–45, 2009.

[20] B. Theelen *et al.*, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *Int. Conf. on Formal Methods and Models for Co-Design, MEMOCODE, Proc.* IEEE, 2006, pp. 185–194.

[21] S. Stuijk *et al.*, "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications." in *ICSAMOS*, L. Carro and A. D. Pimentel, Eds. IEEE, 2011, pp. 404–411.

[22] M. Geilen *et al.*, "An algebra of Pareto points," *Fundamenta Informaticae*, vol. 78, no. 1, pp. 35–74, 2007.

[23] O. Moreira, "Temporal analysis and scheduling of hard real-time radios running on a multi-processor," Ph.D. dissertation, Eindhoven University of Technology, 2012.

[24] M. Geilen *et al.*, "Performance analysis of weakly-consistent scenario-aware dataflow graphs," Electronic Systems Group, Eindhoven University of Technology, Tech. Rep. ESR-2011-03, December 2011.

[25] M. Geilen, S. Tripakis, and M. Wiggers, "The earlier the better: A theory of timed actor interfaces," in *Proceedings of the 14th international conference on Hybrid systems: computation and control (HSCC 11)*. ACM, April 2011, pp. 23–32.

[26] S. Stuijk *et al.*, "Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs," in *Design Automation Conf., Proc.* ACM, 2007, pp. 777–782.

[27] M. Damavandpeyma *et al.*, "Modeling static-order schedules in synchronous dataflow graphs," in *Design, Automation and Test in Europe, DATE 2012, Proceedings*, 2012.

[28] A. Kumar *et al.*, "Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA," *ACM Transactions on Design Automation of Electronic Systems*, vol. 13, no. 3, pp. 1–27, 2008.

[29] D. Culler, J. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1999.

[30] J. Reineke *et al.*, "PRET DRAM controller: bank privatization for predictability and temporal isolation," in *Int. Conf. on Hardware/software codesign and system synthesis, CODES+ISSS '11, Proc.* ACM, 2011, pp. 99–108.

[31] B. Akesson *et al.*, "Memory controllers for high-performance and real-time mpsocs: requirements, architectures, and future trends," in *Int. Conf. on Hardware/software codesign and system synthesis, CODES+ISSS '11, Proc.* ACM, 2011, pp. 3–12.

[32] S. Stuijk, M. Geilen, and T. Basten, "A predictable multiprocessor design flow for streaming applications with dynamic behaviour," in *Conf. on Digital System Design, DSD 10, Proc.* IEEE, 2010, pp. 548–555.

[33] M. Moser, D. Jokanovic, and N. Shiratori, "An algorithm for the multidimensional multiple-choice knapsack problem," *IEICE Trans. Fundamentals of Electronics*, vol. E80-A(3), pp. 582–589, 1997.

[34] T. Basten, "Reliable run-time adaptation in resource-constrained embedded systems," ECE Seminar, CMU, Pittsburgh, Sptember 2009. [Online]. Available: http://www.es.ele.tue.nl/~tbasten/presentations/cmu20090917.pdf