RECONFIGURABLE MEDIA CODING: A NEW SPECIFICATION MODEL FOR MULTIMEDIA CODERS

Christophe Lucarz[1], *Marco Mattavelli*[1], *Joseph Thomas-Kerr*[2] *and Jorn Janneck*[3]

[1]Ecole Polytechnique Federale de Lausanne (EPFL), [2]University of Wollongong, [3]Xilinx Inc.

ABSTRACT

Multimedia coding technology, after about 20 years of active research, has delivered a rich variety of different and complex coding algorithms. Selecting an appropriate subset of these algorithms would, in principle, enable a designer to produce the codec supporting any desired functionality as well as any desired trade-off between compression performance and implementation complexity. Currently, interoperability demands that this selection process be *hard-wired* into the normative descriptions of the codec, or at a lower level, into a predefined number of choices, known as *profiles*, codified within each standard specification.

This paper presents an alternative paradigm for codec deployment that is currently under development by MPEG, known as Reconfigurable Media Coding (RMC). Using the RMC framework, arbitrary combinations of fundamental algorithms may be assembled, without predefined standardization, because everything necessary for specifying the decoding process is delivered alongside the content itself. This side-information consists of a description of the bitstream syntax, as well as a description of the decoder configuration. Decoder configuration information is provided as a description of the interconnections between algorithmic blocks. The approach has been validated by development of an RMC format that matches MPEG-4 Video, and then extending the format by adding new chroma-subsampling patterns.

1. INTRODUCTION

Media coding has changed a lot since its infancy in the early nineties. The original MPEG video coding standard was released in 1993, and since then MPEG-2, MPEG-4 and AVC (Advanced Video Coding) have been produced, and SVC (Scalable Video Coding) is well underway. Each successive codec released by MPEG has been substantially more complex than the last, typically yielding twice the compression efficiency of its predecessor. Because of this growing complexity, the textual specification of recent standards (since MPEG-4) has lost its normative role, being replaced by the *reference software* implementation as the true normative specification. However, while this normative specification (typically in C or C++) is very precise, it presents a number of limitations.

A large portion of compression technology (ie. coding tools) are common across all MPEG standards, but there is no direct way to recognize this commonality. Additionally, the sequential C/C++ descriptions do not expose the potential parallelism that is intrinsic to the algorithms constituting the codecs. They have also become excessively large (in terms of lines of code) making it extremely labor intensive, for example, to transform the reference software into a VHDL implementation. In other words, the complex C/C++ specifications no longer constitute a good starting point for implementation of the standard. It would be preferable to develop formalisms that operate at a higher level of abstraction, that simplify *top-down* system development and design.

The large number of coding tools available also leads to difficulty in specifying predefined subsets for different application scenarios (i.e. standard *profiles*). As an example, a low complexity profile is often defined to provide the minimum configuration expected to achieve acceptable results on highly constrained decoding devices. However, specifying such profiles prior to, or soon after, release of the standard would not appear to allow the optimal combinations of tools to be identified. Furthermore, it is often not possible to identify all of the application scenarios in which a codec will be used, at the time of its release. Nor is it feasible to provide a normative profile for every scenario. Ideally, implementers of a standard should be able to select arbitrary combinations of the available tools, in the way that best matches the requirements of each application. The challenge with this approach is ensuring interoperability, and it is with this aim that we present Reconfigurable Media Coding (RMC), a new framework currently under development by MPEG [1].

The following sections consider the objectives (1.1) and requirements (1.2) for a reconfigurable coding framework, as well as related work (2). After that, each of the components of RMC are discussed: the structure of an RMC bitstream (section 3), the CAL language (4), and the framework as a whole (5). Finally, section 6 presents the results of validation experiments on the framework. This paper presents an overview of the framework as a whole; for greater detail on the bitstream structure, see [2].

1.1. Objectives

A recent trend in multimedia devices (Cell phones, music players, PDAs and the like) is convergence in terms of the functions supported on any single device. This means that the

device must support an increasing number of media formats for images (such as JPEG or TIFF), audio (MP3, AAC, Real Audio) and/or video (MPEG-2, MPEG-4, AVC or Quicktime). Since many of these codecs share common or similar coding tools, the traditional codec-level conformance specification and implementation is not the most efficient way of implementing multiple codec support on real devices. However, this redundancy between coding formats is implicit, at best, and device vendors must expend a great deal of effort to identify and exploit this redundancy. The objective of an RMC framework is thus to describe current and future codecs in a way that makes commonality explicit, reducing the implementation burden for device vendors.

The framework has the following objectives:

- ➢ to create a flexible video and audio coding framework for new codec and coding tool development;
- ➢ to simplify the specification and adoption of new coding tools by explicitly reusing the desired elements of previous standards, instead of defining a new monolithic standard;
- ➢ to provide a new interoperable model of codec definition at the level of fundamental algorithmic modules (such as the Discrete Cosine Transform) that gives users the ability to utilize any module required to suit the requirements of the application, content or network; and
- ➢ to simplify the implementation process for new codecs by making component reuse explicit.

1.2. Requirements

The key requirement for the construction of RMC decoders is that their basic architecture allows for a variety of implementations,—e.g. in software on single or multiple processors, in hardware, or in a heterogeneous mix of hardware and software components. Consequently, the description of an RMC decoder should lend itself easily to parallelization, and it should permit the use of various scheduling policies.

Another essential requirement is that components of RMC decoders can be developed independently and be composed easily. Consequently, the interfaces between components must be well-defined, with precisely specified interactions between components.

Both requirements point to the need for a component model that emphasizes strong encapsulation of state and thin communication interfaces. In particular, the requirement for parallelizability, schedule independence and well-defined interactions suggest the absence of shared memory between components—i.e. components need to strictly encapsulate any state information so that no other components can see or modify it.

In the absence of shared memory, components need to interact by sending each other messages containing packets of data we call *tokens*. In order to increase the independence from specific scheduling and execution mechanisms, message sending (or *token passing*) needs to be asynchronous, and it will often be buffered, in order to accommodate jitter in the execution between the sender and receiver of tokens.

Finally, an RMC decoder requires information about the syntax of the media content, so that it may pass the correct input data to each of the subsequent components. This information must include enough detail to parse data into the atomic units expected by each component. It must identify not just the cardinality constraints of syntactical elements, but also the algorithm to determine the actual cardinality of an instance.

2. RELATED WORK

The requirements outlined in section 1.2 are usually met very well by approaches known by names such as *dataflow* or *stream processing*. Early examples of dataflow are Kahn process networks [3] and Dennis Dataflow [4]. Kahn process networks have the interesting property that they guarantee complete determinism irrespective of the schedule used, at the price of significantly constraining the kinds of computation that could be expressed in that formalism. In Dennis' dataflow the components (called *actors*) execute in a sequence of atomic state transitions (*firings*). It was primarily designed for very loosely coupled computational systems allowing significant generality, while limiting the amount of analysis that can be performed on the actors themselves, or on their composition. Other approaches, such as Hewitt's message passing [5] make similar tradeoffs.

Synchronous dataflow (SDF) [6] combines dataflow with firing with an even further restricted form of Kahn process networks. The result is a model which permits sufficient a priori analysis to compute a complete cyclic schedule statically (i.e. off-line), including sophisticated analysis and optimizations of buffer access patterns (e.g. [7]. The downside of this approach is even less expressiveness, limiting it to fixed-rate systems, and making it quite unsuitable for general media coding.

Cyclo-static dataflow (CSDF) [8] provides a slight generalization over SDF while retaining its advantages of static schedulability and analyzability, but it also shares the problem of being essentially limited to systems with fixed data rates.

A family of *synchronous languages* (such as Lustre [9], Signal [10], and Esterel [11]) use dataflow-like constructions (such as tokens and signals) to provide abstractions of time. Yet while their handling of time makes them eminently suitable for real-time applications (a field in which they enjoyed some notable successes), it makes them less attractive for expressing "pure" dataflow-dominated applications such as media coders.

Parallel programming languages such as Hoare's Communicating Sequential Processes (CSP) [12] also provide channels and the exchange of units of data across them as mechanisms for coordinating concurrent computations. In addition, CSP and the languages built on it (Occam, Mï£¡bius, Handel-C, etc.) conflate the issue of communicating data and of synchronizing computation by building on top of a rendezvous-style interaction, where sending and receiving data is synchronized.

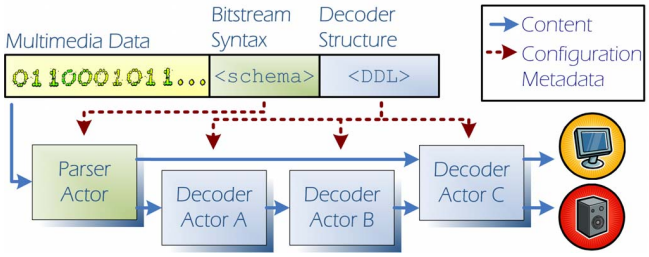

Fig. 1. A general view of an RMC bitstream

Consequently, CSP-style programs can be very sensitive to scheduling and often include a high degree of unchecked nondeterminism.

Instead, RMC builds on the CAL actor language [13] for describing modules of media codecs. This is a language for writing dataflow actors, designed to combine expressiveness with analyzability: it supports the construction of very general actors (more general than Kahn processes, on par with general purpose languages such as CSP), while allowing tools to identify potential sources of nondeterminism, as well as more specialized classes of dataflow such as SDF and CSDF. This information can then be used by tools to decide about implementation options and for off-line scheduling and other optimizations.

Finally, the requirements for content syntax are well met by the Bitstream Syntax Description Language (BSDL), but for a detailed discussion of other alternatives see [2].

3. AN RMC BITSTREAM

The novelty of RMC is that instead of a decoder being rigidly specified, its architecture is transmitted with the encoded data, to enable reconfiguration on-the-fly. In other words, an RMC bitstream is essentially *self-describing*, in that its structure and that of the decoder are both transmitted as part of the bitstream (Figure 1). The decoder structure is written using the Decoder Description Language (DDL), which is a XML dialect, described in section 5.1. The compressed content, on the other hand, is described using a tool from the MPEG-21 standard [14] known as the Bitstream Syntax Description language (BSDL), which is discussed in section 5.2.

In the RMC framework, the receiver device gets the decoder description which fully specifies the architecture of the decoder. In order to instantiate the decoder, the receiver then needs an implementation of the standard library of building blocks specified by RMC. This library is normatively specified using CAL (see section 4), which can be directly synthesized into both hardware (VHDL) and software (C, C++, Java, and so on) by using appropriate tools. Device vendors are, however, free to provide alternative implementations of the standard library that are optimized for their particular platform.

An appropriate level of granularity for blocks within the standard library is important, to enable efficient reuse within the RMC framework. If the library is too coarse, modules will be too large to allow reuse between different codecs. On the other hand, if the granularity is too fine, the number of modules in the library will be too large for an efficient and practical reconfiguration process, and may obscure the desired high-level description of the RMC decoder.

4. THE CAL ACTOR LANGUAGE

One fundamental component of the RMC framework is the standard library of coding tools that are the *high level* building blocks of a codec. For this library a syntax is required to specify each algorithm and its interfaces, in such a way that algorithms may be combined easily, yet correctly. Traditional libraries composed of C functions or C++ classes are inadequate, because they require too much integration overhead to yield a *working* codec model. For these reasons, CAL [13] was chosen over C/C++ for specifying the RMC standard library. This section presents the fundamental characteristics of CAL, and the features that make it suitable for RMC.

4.1. Dataflow oriented processing

Looking for high level descriptions of MPEG codecs leads naturally to a *dataflow* processing paradigm. This is not surprising since the fundamental operation of such codecs is to transform a stream of data from the compressed domain to a stream of decoded audio or video (or vice versa). Furthermore, this transformation is characterized by a sequence of operations that are repeated for each unit in the stream.

CAL is a language used to define the behavior of dataflow components called *actors*, which is a modular component that encapsulates its own state. That is, an actor can neither read nor modify the state of any other actor. The only interaction between actors is via messages (known in CAL as *tokens*) which are passed from an output of one actor to an input of another. The behavior of an actor is defined in terms of a set of *actions*, at most one of which is active at any point in time. The operations an action can perform are to consume (read) input tokens, modify internal state, produce output tokens, and interact with the underlying platform on which the actor is running. Examples of such interaction include reading the incoming RMC bitstream or rendering decoded output.

After an action completes, the next action to be executed (*fired*) depends on

- ➢ the availability of token(s) at the requisite input(s);
- ➢ the value of input tokens;
- ➢ the state of the actor; and
- ➢ the priority of each action.

An actor may contain any number of actions. Its execution follows a cycle:

(a) determine, for each action, whether it is enabled, by testing all the conditions specified in that action;

(b) execute one enabled action (if any); go to (a).

The selection order and the firing conditions for actions form the core of the design of an actor. CAL provides several constructs for describing action selection, including:

- action *guards*: conditions on the values of input tokens and/or the values of actor state variables, that need to be true for an action to be enabled;
- a *finite state machine*, expressed as a set of transitions from one state to another. The condition for each transition is specified by the guards of an action, and when a transition is made, the associated action is fired; and
- action *priorities*: actions may be related to each other by a partial priority order, such that an action will only execute if no higher-priority action can execute. In this way, the process of action selection is specified in a declarative manner by the designer. As a result the actor becomes more compact and easier to understand.

4.2. Hierarchical modular design

With CAL, a RMC decoder is composed of a network of independent actors, which interact with each other only via token passing. This approach facilitates modularity, where the internal implementation of any actor can be modified without impacting other actors. The behavioral description of an actor, and the architecture of the system are thus completely separate. In contrast, the reference implementations of existing MPEG codecs (written in C or C++) make extensive use of shared memory and are difficult or impossible to componentize.

4.3. Communication protocols

Interaction between actors is solely via FIFO channels connecting output ports to input ports. The atomic unit of data sent across these channels is called a *token*, which may be a simple value (such as an integer), an arbitrarily complex data structure, or even a function or procedure (borrowing from the functional programming paradigm).

When a token is produced at an output port, it is delivered to the queue at each input port to which it is connected. The token remains in the queue(s) until it is consumed by the actor that owns that queue.

4.4. Nondeterministic scheduling and explicit parallelism

Notwithstanding the firing conditions and schedules discussed above (in 4.1), the order of execution for actions is nondeterministic. This provides the designer of an RMC decoder great flexibility to schedule action execution according to the particular requirements and constraints of the hardware/software. In terms of the former, this allows better optimization of area, throughput, power consumption, latency, and so on.

Moreover, a codec specified as a network of CAL actors explicitly exposes parallelism by virtue of the independence of different actors. This parallelism can be exploited if desired, by specific implementations. This is not possible with monolithic C/C++ specifications, where the identification of parallelism is a significant and resource-intensive task.

4.5. Summary

To summarize, CAL is a language that

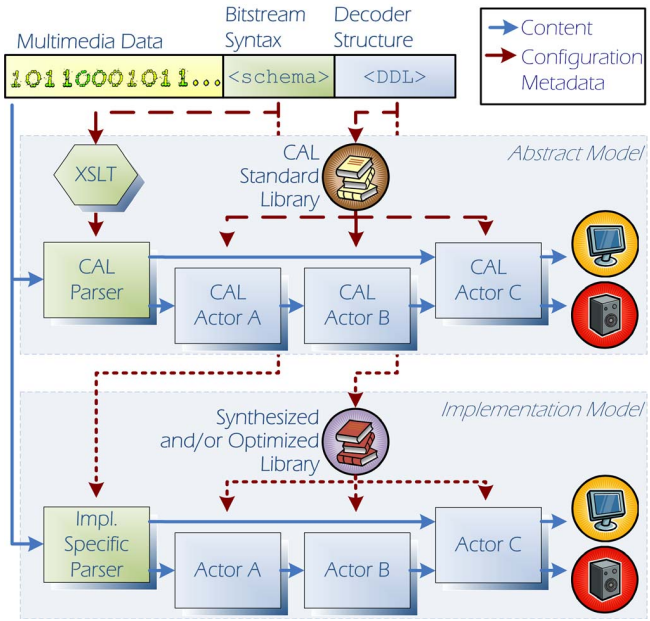

Fig. 2. Reconfigurable Media Coding framework

- is based on dataflow processing primitives;
- facilitates top-down (block diagram) design;
- encapsulates processing tasks in units called *actors*;
- facilitates parallelization both in terms of development (ie actors may be written in parallel by different authors), and operation (actors may be executed on independent processors or cores); and
- hides details of execution scheduling that are unnecessary for dataflow modeling, but can specify scheduling and flow control when necessary.

5. THE RECONFIGURABLE MEDIA CODING FRAMEWORK

Like previous MPEG coding tools, RMC specifies the operation of the decoder and the bitstream syntax, leaving the particulars of the encoder open to proprietary competitive advantage. However, unlike previous tools, RMC does not itself define a new codec. Instead, RMC provides a framework to allow content providers to define a multitude of different codecs, by combining together blocks (actors) from the standard library.

There are two slightly different models for an RMC decoder (Figure 2). In the abstract model used for the reference software, decoder actors are instantiated directly from the reference CAL library. The bitstream schema is transformed into a parser actor (see [2]), and the actors run on an interpreter.

On the other hand, device vendors implementing RMC have considerable latitude to optimize the decoder execution environment. Instead of instantiating CAL blocks, the standard library is implemented in a format native to the environment. The library may be synthesized from the reference library (for example, a CAL to VHDL compiler is available [15]), and/or further optimized as part of the decoder implementation. The

```
<xsd:complexType name="VideoObjectLayerType"cmc:port="vol">
 <xsd:sequence>
  <xsd:choice>                                              <!– a –>
   <xsd:group ref="longHeader" bs2:ifNext="00000120"/>
   <xsd:group ref="shortHeader"/>
  </xsd:choice>
  <xsd:element name="VOP" type="VideoObjectPlaneType"
               bs2:ifNext="000001B6" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:group name="longHeader">
 <xsd:sequence>
  <xsd:element name="VOLStartCode" type="SCType"/>
   <!– ...and so on... –>
   <bs2:variable name="mbCount" value="(($volWidth+15)
                 idiv 16)*(($volHeight+15) idiv 16)"/><!– b –>
 </xsd:sequence>
</xsd:group>
```

Fig. 3. Part of the BSDL Schema for MPEG-4 Video

interface of each actor in the standard library (in terms of inputs, outputs and behavior) is normatively defined, but the implementation is not. Equally, the bitstream schema bitstream metalanguage and semantics are normative, but the parsing process is fully implementation dependent.

5.1. Decoder Description Language

The second fundamental component of the RMC framework is the language used for the description of the decoder as a network of coding tools (i.e. actors). This Decoder Description Language (DDL) specified in RMC is an XML dialect that describes an interconnected network of standard library components, which together represent a complete decoder. A DDL description of the intended decoder configuration is transmitted as part of an RMC bitstream, and is used by the decoder to instantiate and interconnect the appropriate modules from the standard library. DDL can also be used recursively; that is, an Actor may be defined as a composition of other actors, with the interconnections specified by DDL. In this case, the DDL itself declares input and output ports.

DDL provides a facility for declaring parameters, and passing parameters to actors in the network. This is useful for declaring values that are constant for a particular instantiation of an actor, but may vary between different instantiations. For example, a vendor may have a number of different RMC-enabled devices, with varying screen resolution or audio depth. In this case the vendor may implement certain actors in the standard library only once, but with parameters to fix the varying quantities.Parameter values are denoted by expressions, which may depend on the values of other parameters and global or local variables.

5.2. Bitstream Syntax Description

The other part of a RMC bitstream is a description of the syntax used for the content data. This information allows a RMC decoder to parse the bitstream into fields, as well as group individual fields into semantic units. Of the numerous

Table 1. Comparison of the MPEG-4 Video Decoders

Source Lines of Code	CAL	C++	C	Propr.
Parsing	1064	4646	5750	1620
Texture Decoding	993	5245	12844	1728
Block Splitting	73			
Block Expansion	60			
DC Splitting	17			
DC Reconstruction	524			
Inverse Scanning	77			
Inverse AC Prediction	140			
Inverse Quantization	39			
Inverse DCT	63			
Motion Compensation	1105	4966	6773	540
Motion Vector Splitting	189			
M. Vector Sequencing	124			
M. Vector Reconstruction	314			
Addressing	303			
Frame Buffering	14			
Interpolation	94			
Adding	67			
Other	21	15467	19597	1512
Grand Total	3183	30324	44964	5400

syntax description languages available, BSDL [14] was found to be the most suitable, because

- it is stable and defined by an international standard [14];
- its XML-based syntax integrates well with DDL; and
- a parser may be easily derived by transforming the BSDL using standard tools such as XSLT [16].

BSDL provides a way to create schemata for bitstreams. For example, Figure 3 presents part of the BSDL Schema for any MPEG-4 Video stream. Informally, this excerpt states that a Video Object Layer is made up of either a long header or a short header, as well as many Video Object Plane structures (VOP is MPEG-4 terminology for a video frame). The choice between a long or short header is made on the basis of whether the subsequent bits in the bitstream are equal to the hexadecimal value 00000120 (this is in fact the start code that is subsequently stored as the first field of a long header). The variable (mbCount) is computed on the basis of prior fields in the long header, and is used when parsing VOPs to determine the number of MacroBlocks (MBs) to parse.

For further information on BSDL in RMC, see [2].

6. RESULTS

In order to validate the RMC approach, we have developed an RMC bitstream and decoder that correspond to the MPEG-4 Video Simple Profile (Figure 4). The CAL-based decoder is substantially more concise than either the C and C++ reference software (published by MPEG as the normative specification for MPEG-4), or an optimized, proprietary decoder implemen-

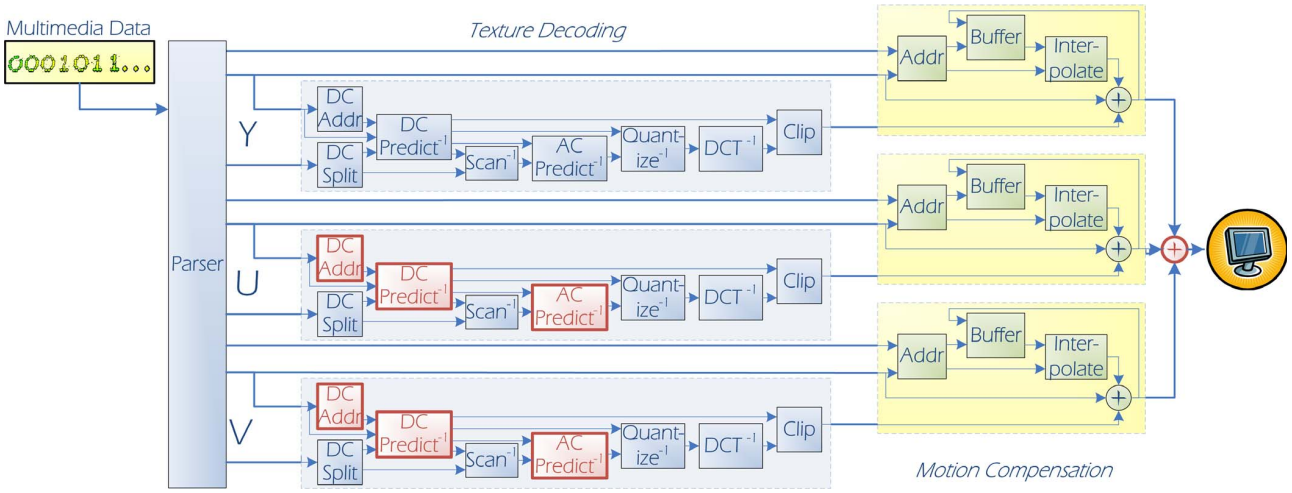

Fig. 4. MPEG-4 Video Simple Profile decoder & extensions

tation (as shown, Table 1). Additionally, the CAL implementation is considerably more modular than any of the others, and its dataflow paradigm greatly simplifies parallelization.

Furthermore, we have extended this decoder with new chroma-subsampling patterns; features which are not available in MPEG-4 Simple Profile. The changes to the bitstream and its schema to effect these extensions consist of extra chroma blocks in each Macroblock, as well as changes to the chroma block pattern header field. The DDL is changed to instantiate DC Addressing and DC & AC inverse prediction blocks with a greater resolution (Figure 4; altered blocks in bold).

7. CONCLUSION

This paper describes the objectives and the essential components of a new framework under standardization at MPEG for Reconfigurable Media Coding. These components are: a standard library of coding tools (actors) described in CAL, a language (DDL) for the specification of networks of actors that provides the decoder description, and a language (BSDL) for the specification of the bitstream syntax. Using these tools it is possible to reconfigure codecs as desired, and this new mode of specification results in decoders that are substantially more compact, modular, and expressive in terms of potential parallelism and task scheduling, in comparison to previous C/C++ specifications. RMC also allows the user to combine coding tools from different standards, and to achieve trade-offs not allowed by current monolithic predefined profiles.

8. REFERENCES

[1] ISO/IEC, "Working draft 3 of ISO/IEC 23001-4: Codec configuration representation," 2007.

[2] J. Thomas-Kerr et al., "Reconfigurable media coding: Self-describing multimedia bitstreams," in *Signal Processing Systems, IEEE Workshop on*, 2007.

[3] G. Kahn, "The semantics of a simple language for parallel programming," in *Proceedings of the IFIP Congress*. 1974, North-Holland Publishing Co.

[4] J.B. Dennis, "First version data flow procedure language," Tech. Memo MAC TM 61, MIT Lab. Comp. Sci., 1975.

[5] C. Hewitt, "Viewing control structures as patterns of passing messages," *Journal of Artifical Intelligence*, vol. 8, no. 3, pp. 323–363, 1977.

[6] E.A. Lee and D.G. Messerschmitt, "Synchronous data flow," *IEEE Proceedings*, 1987.

[7] E.A. Lee and D.G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, 1987.

[8] M. Engels et al., "Cyclo-static dataflow: Model and implementation," in *28th Ann. Asilomar Conference on Signals, Systems, and Computers*, 1994, pp. 503–507.

[9] P. Caspi et al., "Lustre: A declarative language for programming synchronous systems," in *ACM Symposium on Principles of Programming Languages*, Munich, 1987.

[10] P. Le Guernic et al., "Programming real-time applications with SIGNAL," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1321–1336, 1991.

[11] G. Berry and G. Gonthier, "The Esterel synchronous programming language: Design, semantics, implementation," *Science of Computer Programming*, vol. 19, no. 2, pp. 87–152, 1992.

[12] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985.

[13] J. Eker and J.W. Janneck, "CAL Language Report," Tech. Memo UCB/ERL M03/48, UC Berkeley, 2003.

[14] C. Timmerer et al., "Digital item adaptation - coding format independence," in *The MPEG-21 Book*, I. Burnett et al., Eds. Wiley, Chichester, UK., 2006.

[15] J. Janneck, "The CAL actor language: Synthesizing models to FPGA," `http://chess.eecs.berkeley.edu/pubs/181.html`, 2007.

[16] J. Clark, "XSL transformations (XSLT)," `www.w3.org/TR/xslt`, 1999.