I. INTRODUCTION

The rapidly increasing performance requirements of embedded applications, such as HDTV and high-speed networking, motivated the production of customized, network-centric, multi-processor hardware platforms, e.g., Intel IXPXX and Philips Wasabi/Cake, composed of multiple heterogeneous, multi-threaded processors, which communicate through sophisticated interconnects. To exploit the benefits introduced by these application-oriented architectures, there is a need to develop a design, analysis and implementation flow providing integrated, rather than separate, software and hardware modeling and simulation. The main challenge is to provide methodologies and tools capable of supporting accurate, but fast, performance analysis for dimensioning the hardware platform and mapping of software.

To this end, we have developed a platform-based design framework composed of a code-generation tool, JAHUEL [1], and a simulation tool, P-WARE [2].

JAHUEL is based on formal language which provides (1) platform-independent constructs for parallel software programming application using an abstract execution model, and (2) semantic and syntactic support for architecture-dependent code-generation via program transformations. P-WARE relies on a formal component-oriented meta-model for cleanly composing hardware and software sub-systems without interfacing wrappers.

In the context of an academic-industrial cooperation, current work in progress aims to develop a design flow which starts from a high-level modelling of application software and hardware, all the way down to the concrete implementation and mapping (currently manual), while going through the synthesis of an application-specific software scheduler [4] (currently manual), the analysis of software-hardware joint performance simulation, and the generation of executable code.

II. METHODOLOGY

Setting up a system with this framework is operated as follows (figure 1). First of all, JAHUEL is used to define the models of the application and the architecture. The application is the software (e.g., a video encoder) and its environment (e.g., camera and display devices), while the architecture is the underlying execution hardware (e.g., processors, buses, caches, etc.). The application model only takes into account environment execution times and periodic activations, and software and environment interactions due to data and control dependencies. That is, the model abstracts away from hardware-dependent issues, such as conflicts between software and environment communication due to concurrent bus and memory accesses, and, as a consequence of this abstraction, software execution and communication times are treated as unknown parameters.

```
1 <pnode-sum>
2  <dep-list> ... </dep-list>
3 <body>
4  <!-- Writer -->
5 <while>
6   <label>Writer</label>
7   <condition>
8     <b-exp>
9       <boolean default-value="true"/>
10     </b-exp>
11   </condition>
12 <period>
13   <time/>30</time>
14   <time-unit>us</time-unit>
15 </period>
16 <body>
17   ...
18   <source-code>
19     function write();
```

Then, the step “constraint synthesis” derives a scheduler of this application and a constraint on the parameters which must be satisfied at runtime, by any parallel mapping of software which will be defined later. After that, we look for software implementations, i.e., mappings on processors, which must satisfy the latter constraint. This is done by considering several classes of parallel implementations (e.g., data parallel, task parallel and hybrid implementations).

Software components corresponding to each of these implementations are generated, in addition to the hardware ones and their bindings, as input to P-WARE.

If performance objectives are met, JAHUEL synthesizes the executable code of software.

III. A SIMPLE PRODUCER/CONSUMER SYSTEM

The software is composed of two elements: a producer (Writer) and a consumer (User). The snippet code below depicts the FXML model. The producer is an infinite loop writing a variable \( x \) and the consumer is an infinite loop reading it. Data dependencies, time properties and constraints are also specified in the model by the content of dep-list, execution-time and period elements, respectively. Writer and User are concurrent and are thus put inside a pnode-sum element, used to specify task parallelism.
The hardware is composed of two processors, accessing the off-chip memory through two DMAs and one data bus. Hardware components may be connected to one or several target components. The caller can send transactions requests (TR) (eg. read/write) to the recipient through its output ports. These bindings are specified in the snippet code below by elements qoutput and qout, for one-to-one or one-to-n ports bindings, respectively.

```plaintext
1  <Bus><-- CHANNEL definition -->
2  <arch_label>CHANNEL</arch_label>
3  <id>1</id>
4  <qin>RBUS, BF</qin>
5  <qoutput>
6  <qout>DMA</qout>
7  <number>1</number>
8  <qoutput>
9  ...
10  <clock-port> CLK </clock-port>
11  </Bus>
12 ...
```

A. From JAHUEL to P-WARE

To produce P-WARE components code for the Producer/Consumer example above, JAHUEL was configured to operate the following sequence of transformations:

The componentization step encapsulates the FXML nodes into software component objects. Current possible componentizations consist of mapping each leaf to a P-WARE software component, where a leaf is a node which has no pnode-sequence or pnode-sum as a descendant in its syntax tree. Componentization may also be derived using components names given by designer in a mapping attribute attached to the software nodes.

The synchronization step implements the FXML nodes dependencies and timing constraints. It consists of adding the definition of the schedule block for each component in the FXML model. This is done by indicating for each node its role in a dependency (senders or recipients), the dependency pattern (eg. the recipient must wait all senders), the number of senders and the number of recipients and timing conditions.

Finally, components code is generated for P-WARE by applying a set of code generation templates (stylesheets), where a template is a pair composed of a node name in the software syntax tree, and a rule to apply when this node is matched.

Figures below depict bindings of the generated Writer/User system, and as an example, the CHANNEL component model, and the MEMORY component model, respectively.

B. Performance monitoring

Given a set of hardware and software components, automated behavior monitoring is achieved by P-WARE meta-model integrated performance observers.

As an example, figure below shows the disparity of tasks synchronisation times provided by the tool, for a task-parallel implementation of a part II MPEG-4 implementation using the 16 range based hierarchical motion estimation algorithm, H16, on the Philips Cake architecture including up to 8 processors.

IV. Conclusion

JAHUEL/P-WARE is a flexible and extensible tool-suite which provides a platform-based design flow for multiprocessor embedded systems. Our framework supports modelling, code-generation and joint software/hardware performance simulation which enables fast and precise exploration of different system configurations, both at software and hardware levels, in order to find one that meets QoS requirements. In this paper, we gave an overview of JAHUEL modelling and exploration environment on the top of P-WARE simulation tool.

Acknowledgment

This work is partially supported by projects MEDEA+ NEVA and Minalogic SCEPTRE. The first author has been funded by a joint CNRS/STMicroelectronics research grant.

References