



Software-Hardware Interface for Multi-Many-Core (SHIM) Specification V1.00 Final

Document ID: SHIM Specification

Document Version: 1.00

Status: Final

Copyright © 2015 The Multicore Association, Inc.

All rights reserved.

No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission from The Multicore Association, Inc.

All copyright, confidential information, patents, design rights and all other intellectual property rights of whatsoever nature contained herein are and shall remain the sole and exclusive property of Multicore Association. The information furnished herein is believed to be accurate and reliable. However, no responsibility is assumed by The Multicore Association, Inc. for its use, or for any infringements of patents or other rights of third parties resulting from its use.

The Multicore Association, Inc. name and The Multicore Association, Inc. logo are trademarks or registered trademarks of The Multicore Association, Inc. All other trademarks are the property of their respective owners.

The Multicore Association, Inc.
PO Box 4854
El Dorado Hills, CA 95762
530-672-9113
www.multicore-association.org

Table of Contents

Preface	6
Definitions	6
1. Introduction	7
1.1 Overview	7
1.2 Interface	7
1.3 SHIM Editor	9
2. SHIM Concepts	12
2.1 Topology - ComponentSet	12
2.2 Memory - AddressSpaceSet	12
2.3 Inter-core communication – CommunicationSet	13
2.4 Performance	14
2.4.1 General	14
2.4.2 Latency and Pitch	15
2.4.3 Using triplets	15
2.5 Software View – what is in and what is not	16
2.6 XML	17
2.6.1 Data Binding	17
2.6.2 Who Creates SHIM XML	17
2.7 Configuration	18
2.7.1 General	18
2.7.2 Common Configuration File (CCF)	18
2.8 Reference Authoring Tools	20
2.9 Roadmap	20
2.9.1 Componentization of SHIM XML	21
2.9.2 Hardware-Related Software Properties	21
2.9.3 Schema Refinement for Smaller XML	21
3. SHIM Interface	22
3.1 shim.xsd	22
3.2 Conventions	28
3.3 Enumeration	29
3.4 SystemConfiguration	31
3.4.1 ClockFrequency	31
3.5 ComponentSet	32
3.5.1 MasterComponent	32
3.5.2 SlaveComponent	33
3.5.3 Cache	34
3.5.4 AccessTypeSet	34
3.5.5 AccessType	35
3.5.6 CommonInstructionSet	35
3.5.7 Instruction	36
3.5.8 Performance	36
3.5.9 Latency	36
3.5.10 Pitch	37
3.6 AddressSpaceSet	38
3.6.1 AddressSpace	38
3.6.2 SubSpace	39
3.6.3 MemoryConsistencyModel	39
3.6.4 MasterSlaveBindingSet	40
3.6.5 MasterSlaveBinding	40
3.6.6 Accessor	40
3.6.7 PerformanceSet	40

3.7	CommunicationSet.....	41
3.7.1	FIFOCommunication	41
3.7.2	SharedRegisterCommunication	42
3.7.3	InterruptCommunication.....	42
3.7.4	SharedMemoryCommunication	43
3.7.5	EventCommunication	43
3.7.6	ConnectionSet	44
3.7.7	Connection	44
4.	Use Cases.....	45
4.1	Performance Estimation: Auto-Parallelizing Compiler.....	45
4.1.1	Using “CommonInstructionSet”	45
4.1.2	Using “PerformanceSet”	45
4.1.3	Using “Cache”	45
4.1.4	Using “FIFOCommunication”	46
4.2	Tool Configuration - RTOS Configuration Tool.....	46
4.2.1	Using “ClockFrequency”	46
4.2.2	Using “SubSpace”	46
4.3	Hardware Modeling.....	47
5.	SHIM XML Authoring Rules and Guidelines	48
5.1	File Name [Rule].....	48
5.2	Naming of Various Objects [Rule]	49
5.3	Level of Detail and Precision [Guideline].....	49
6.	Common Configuration File (CCF).....	50
6.1	Concept	50
6.1.1	Multiple Hardware Configuration	50
6.1.2	Vendor-Specific Hardware Features Affecting SHIM Objects	50
6.1.3	Configuration Tool User Interface.....	50
6.2	Interface.....	51
6.2.1	XML Schema	51
6.2.2	Semantics	53
6.2.3	FormType.....	53
6.2.4	ConfigurationSet	53
6.2.5	Configuration	54
6.2.6	Item	54
6.2.7	Expression	54
6.2.8	Def.....	55
6.3	Examples.....	55
6.3.1	Generic	55
6.3.2	Nested configuration.....	55
7.	FAQ.....	57
8.	Appendix A: Acknowledgements	58
TABLE 1. SHIM REPRESENTATION OF HARDWARE COMPONENTS		12
TABLE 2. INTER-CORE COMMUNICATION CLASSES		13
TABLE 3. PERFORMANCE PROPERTIES IN SHIM.....		14
TABLE 4. USING TRIPLES.....		16
TABLE 5. PERFORMANCE ESTIMATION USE CASE.....		45
TABLE 6. TOOL CONFIGURATION USE CASE.....		46
TABLE 7. HARDWARE MODELING USE CASE.....		47
FIGURE 1. SHIM PROVIDES THE INTERFACE BETWEEN THE HARDWARE AND THE SOFTWARE TOOLS		7
FIGURE 2. THE SHIM ELEMENTS MAPPED TO A PSEUDO-MULTICORE HARDWARE.....		8

FIGURE 3. CLASS DIAGRAM REPRESENTATION OF THE WHOLE SHIM XML SCHEMA	10
FIGURE 4. SHIM XML FILE EXAMPLE	11
FIGURE 5. SHIM EDITOR MAIN WINDOW	11
FIGURE 6. LATENCY AND PITCH REPRESENT THE PRIMARY PERFORMANCE CHARACTERISTICS.	15
FIGURE 7. CCF EXAMPLE	19
FIGURE 8. GUI GENERATED BY CCF	20
FIGURE 9. COMMON CONFIGURATION FILE (CCF) CLASS DIAGRAM	51

Preface

This document is intended primarily for tool developers and hardware developers who would use SHIM to exchange hardware description for software tools. It also attempts to provide software developers with insights into what hardware information is described in SHIM to foster understanding of the intention and the extent of SHIM.

This document begins with the introduction to SHIM, providing the background, the overall concept, and model. It is followed by a chapter detailing the concept of SHIM, such as the purpose, scope, design, interface, limitation, providing the basic idea why SHIM is as specified in this document, and also trying to explain the basic principles for future extension of the specification. A chapter describing the interface follows, which is a description of SHIM XML schema and APIs that are mostly derived directly from the schema via XML data binding technique. A chapter providing some of the detailed use cases follows, allowing the reader to gain insights into how SHIM can be used in action. Finally, this document ends with various Appendixes providing further detailed information.

Definitions

All new terms are defined at the first appearance, either in the main text body or as a footnote.

1. Introduction

1.1 Overview

Multicore processors have become the norm, and processors with tens, and even more than a hundred cores are emerging. These multicore processors vary not only in the number of cores, but also in inter-connects, cluster organization, and memory systems (including hierarchy and cache coherency), among others. While the trend for an increasing number of cores is both natural and unavoidable from a processor design perspective, this poses tremendous challenges to the software developers to cope with the significant hardware variance, while bearing a burden to re-use the existing and newly created software for different hardware. Moreover, all this must occur while achieving the performance expected from the multicore processors, which requires deep understanding of the specific multicore architecture. Various tools, such as auto-parallelizing compilers, parallelization tools, OS/middleware configurators, and performance analysis tools, aid developers to design, implement, and analyze the software. However, these tools must comprehend the complex multicore processor, transferring the burden to the tool developers. Therefore, it is critical to lower the cost of supporting new multicore hardware by various tools, but there has been a lack of effort in academia or industry to solve these issues, thereby hindering the development of the multicore tool eco-system.

The SHIM, Software-Hardware Interface for Multi-many-core, is a joint industrial and academic effort to standardize the interface between the multicore hardware and the software tools. As a result, we aim to lower the cost of supporting new multicore hardware using the standard interface. This will encourage the development of new innovative multicore tools, resulting in a richer eco-system of multicore technologies, which in turn should benefit system developers, semiconductor vendors, and tool vendors.

1.2 Interface

The SHIM is defined as an XML schema. A multicore hardware implementation is expressed as a SHIM XML file which can be used by various tools (Figure 1).

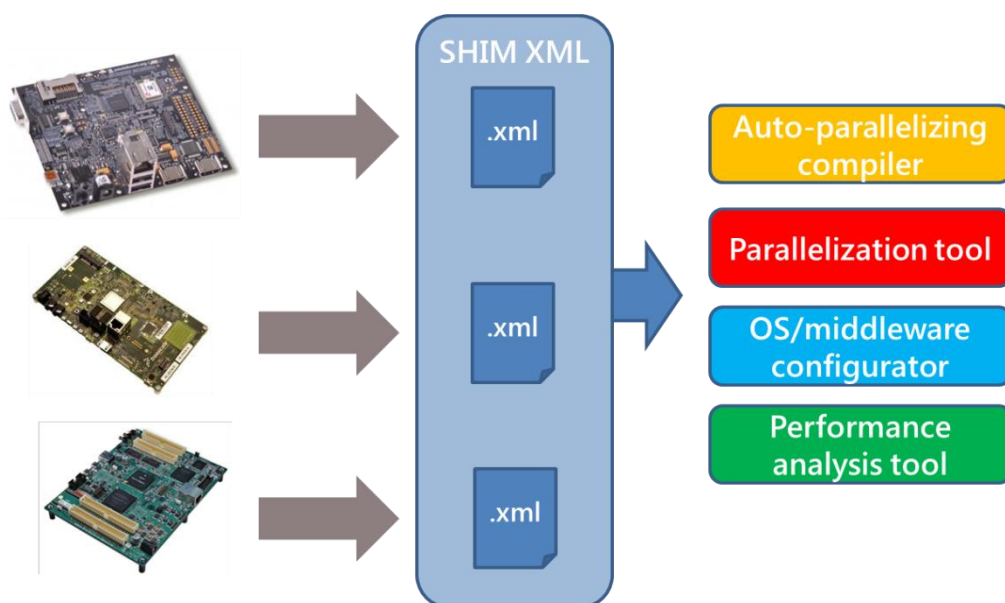
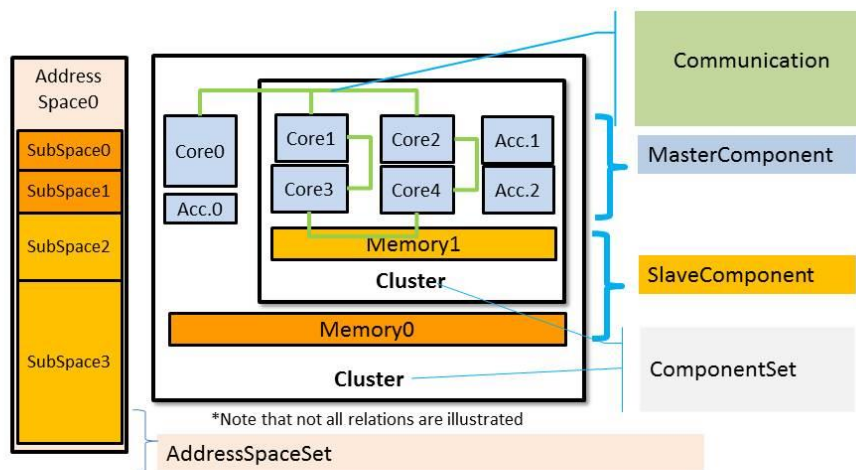


Figure 1. SHIM provides the interface between the hardware and the software tools

The SHIM XML file has a tree structure (Figure 4. SHIM XML file example) with three top level components, namely *ComponentSet*, *AddressSpaceSet*, and *CommunicationSet*, each containing further child elements. The *ComponentSet* contains *MasterComponent* (representing a processor or accelerator) and *SlaveComponent* (representing a memory block or memory subsystem). The *AddressSpaceSet* contains one or more *AddressSpace*, which in turn contains *SubSpace*. Finally the *CommunicationSet* contains any number of *Communication* elements, describing the connection and communication between a pair of *MasterComponents*.



2

Figure 2. The SHIM elements mapped to a pseudo-multicore hardware

A *ComponentSet* can nest itself. For example, it can be used to express a chip that contains multiple hardware clusters, each cluster containing multiple cores with a cluster local memory. It can also be used to describe a board, which in turn may contain one or more multicore chips. A *ComponentSet* can even be used to describe a system with multiple boards, each board connected via PCI Express, for example. As such, the *ComponentSet* tree describes the multicore hardware system topology. This topological architectural information is important for software tools to be able to identify the number of cores, location of the memory devices, and how cores are organized into different clusters.

Since SHIM is for software tools, it is essential to understand from a software perspective, the connection and communication mechanism between the cores (including accelerators), as well as how these cores can access the different memories. The former is described as *CommunicationSet* containing different communication classes. A simple example of defined classes is *InterruptCommunication*, which contains one or more “connection” class, which binds a pair of *MasterComponents*. For memory access, the *SubSpace* contained in the *AddressSpace* includes its start address and size and one or more *MasterSlaveBinding*, containing references to a *MasterComponent* and *SlaveComponent*, describing which core/accelerator can access which memory through the address range.

The hardware architectural information described so far allows tools to understand the hardware topology, and how the cores and memory devices are connected. However, this alone is often insufficient for many tools, since the application software supported by these tools must not just ‘run’, but run with performance qualifiers. To achieve this, the tools must ‘estimate’ the rough performance so that the system designers and software developers know the expected performance from the given application and multicore hardware. Therefore, SHIM, in addition to the hardware topological information, describes the performance properties associated with the processor cycles

consumed to perform the various core-to-core communication (*CommunicationSet*) and also the memory access cycles by different cores and accelerators. The performance is described as *Performance* element, which contains *Latency* and *Pitch*, expressed in processor cycles. The *Performance* element exists for each *CommunicationSet*, for each specific pair of two *MasterComponents*. For memory access performance, for each *MasterSlaveBinding* of each *SubSpace*, and for each *AccessType*, which are defined for each *MasterComponent*, a specific *Performance* element is included. So for each different access type (e.g., read or write, word access or double word access), a different *Performance* element is provided. The cycles can be described in a form of triplet, which are ‘best’, ‘typical’, and ‘worst’, to accommodate the possible performance variance. The tool must be intelligent enough to benefit from these figures, such as analyzing the application code if it is issuing a sequential memory access, which generally falls into use of the ‘best’ cycles. Note that the cycles mentioned here are processor-cycles, and *ClockFrequency* of *MasterComponent* overrides that of *SystemConfiguration* if they are not identical.

1.3 SHIM Editor

Although a SHIM XML schema is relatively simple, as can be seen on the UML Class diagram representation of the whole SHIM XML schema (Figure 3), the resulting SHIM XML file can be quite large, mostly due to all the *Performance* element descriptions for all types of memory accesses. Writing it manually can be tedious and error-prone, so we have developed an editor tool called the SHIM Editor, to foster authoring a SHIM XML file. The generated SHIM XML file is shown in Figure 4, and the SHIM Editor prototype’s main window is shown in Figure 5.

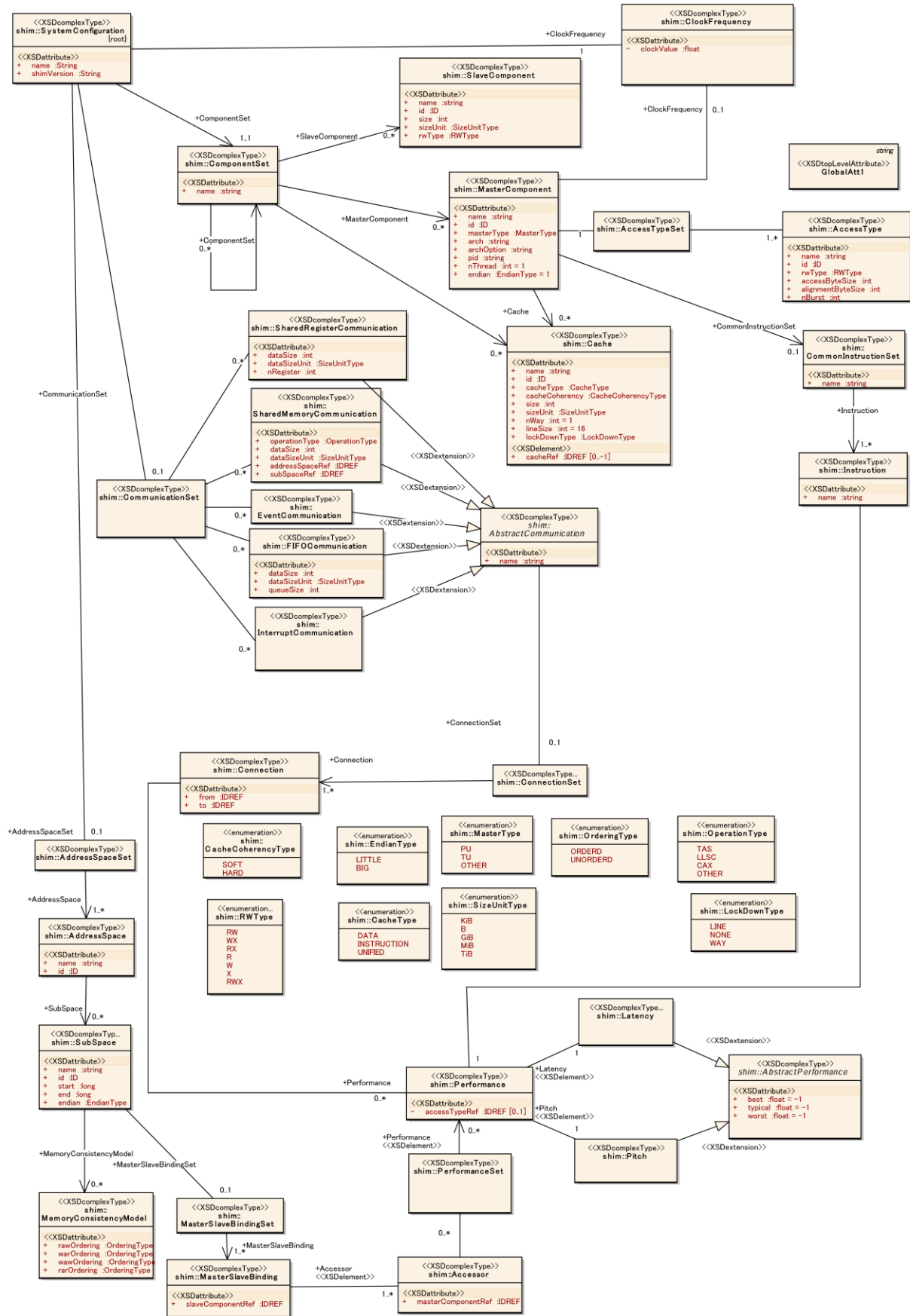


Figure 3. Class diagram representation of the whole SHIM XML schema

1. All root elements

```

1  <?xml:version="1.0" encoding="UTF-8" standalone="yes"?>
2  <SystemConfiguration name="MySystem">
3  <ComponentSet name="Cluster_0">
4  <ComponentSet name="Cluster_0_0">
775 </ComponentSet>
776 <CommunicationSet>
4642 <AddressSpaceSet>
562232 <ClockFrequency clockVa
562233 </SystemConfiguration>

```

2. ComponentSet expanded

```

1  <?xml:version="1.0" encoding="UTF-8" standalone="yes"?>
2  <SystemConfiguration name="MySystem">
3  <ComponentSet name="Cluster_0">
4  <ComponentSet name="Cluster_0_0">
5  <SlaveComponent name="Memory_0_0_0" size="128" sizeUnit="MB" rwT
6  <MasterComponent name="Core_0_0_0" masterType="CPU" arch="Arch"
7  <MasterComponent name="Core_0_0_1" masterType="CPU" arch="Arch"
8  <MasterComponent name="Core_0_0_2" masterType="CPU" arch="Arch"
9  </ComponentSet>
10 </SystemConfiguration>

```

3. Performance under AddressSpaceSet

```

1  <?xml:version="1.0" encoding="UTF-8" standalone="yes"?>
2  <SystemConfiguration name="MySystem">
3  <ComponentSet name="Cluster_0">
4  <ComponentSet name="Cluster_0_0">
775 </ComponentSet>
776 <CommunicationSet>
4642 <AddressSpaceSet>
4643 <AddressSpace name="AS_0_0">
4644 <SubSpace name="SS_0_0_0" start="00000000" end="00040000" endian
4645 <MasterSlaveBindingSet>
4646 <MasterSlaveBinding slaveComponentRef="Memory_0_0_0">
4647 <Accessor masterComponentRef="Core_0_0_0">
4648 </MasterSlaveBinding>
4649 <Performance>
4650 <accessTypeRef>AT_0_0_0_0</accessTypeRef>
4651 <Pitch best="10.0" typical="10.0" worst="10.0"/>
4652 <Latency best="10.0" typical="10.0" worst="10.0"/>
4653 </Performance>
4654 </AddressSpace>

```

Figure 4. SHIM XML file example

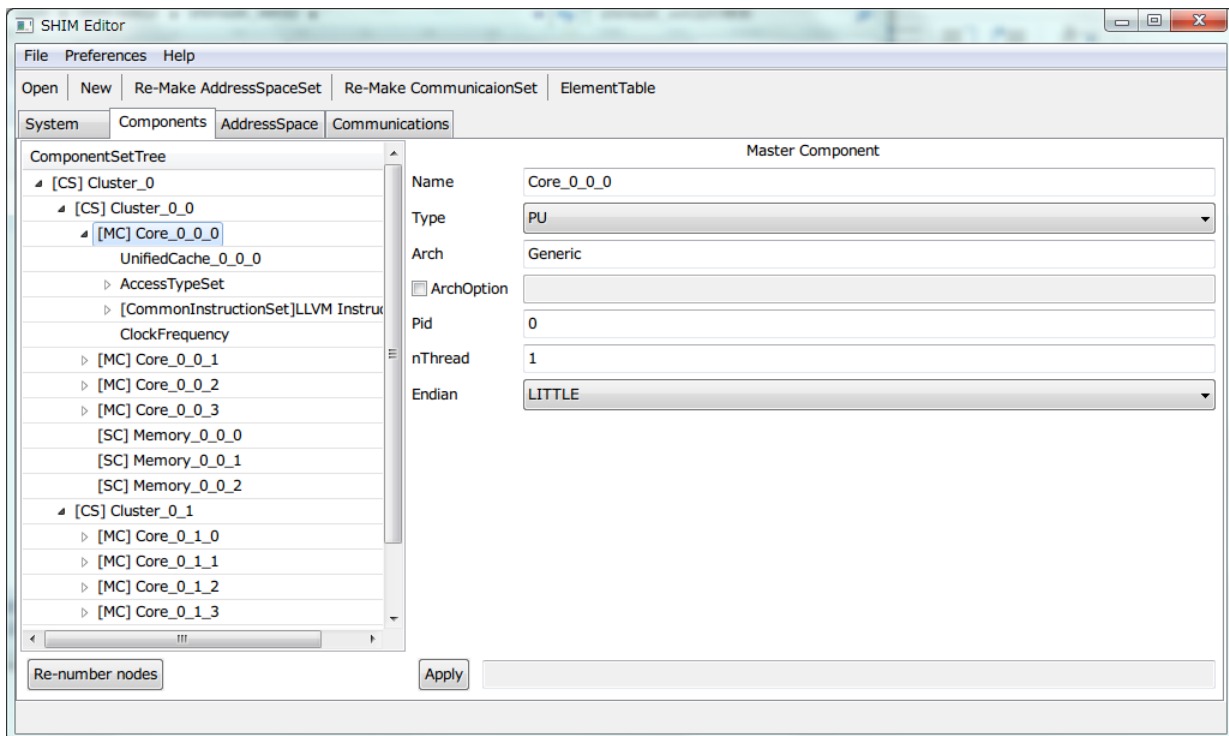


Figure 5. SHIM Editor main window

2. SHIM Concepts

This section describes the major SHIM concepts, providing the basic idea why SHIM is as specified in this document, and also attempts to indicate the principle for future extension of the specification. This chapter should provide a foundation for understanding the [SHIM interface](#), so it is strongly recommended to read this thoroughly before diving into the interface details.

2.1 Topology - ComponentSet

A simple hardware setup may consist of a single processor core and a single memory – however, the multi-many-core hardware has multiple processor cores and memory devices of various types in various configurations. The combination and configuration of processor and memory characterizes the multi-many-core hardware, and it is essential for software tools to comprehend them.

SHIM expresses the particular mix of processors and memory devices as ‘topology’. In the electrical circuits’ terminology, topology “*is the form taken by the network of interconnections of the circuit components. Different specific values or ratings of the components are regarded as being the same topology. Topology is not concerned with the physical layout of components in a circuit, nor with their positions on a circuit diagram. It is only concerned with what connections exist between the components. There may be numerous physical layouts and circuit diagrams that all amount to the same topology*”¹. From the SHIM’s perspective, the topology is extended further. In addition to processor cores and memory devices, which are components in the electrical terminology, we also include ‘clusters’, which is a particular set or grouping of processor cores and memory devices. Usually there are electrical connections between a cluster and other hardware elements, however SHIM does not necessarily deal with actual electrical connections, so the cluster may not form any connection. However, it is critical for software tools to see how processor cores and memory devices are grouped as it is often an indication of a performance difference, therefore SHIM includes *cluster* as a part of its topological expression.

A *cluster* is composed of any combination of another (inner) cluster, processor core, and memory device. SHIM has its own way of classifying and naming these objects (Table 1). A processor core is represented as a *MasterComponent* object. As can be seen from the table, a *MasterComponent* can also be some type of accelerator (e.g., a DMA controller). The objective of *MasterComponent* is to represent those electrical components that play the role of master component in the traditional master-slave bus setup, but only if they are relevant to the software view SHIM defines.

Table 1. SHIM representation of hardware components

SHIM term	Hardware term
ComponentSet	Cluster of any level (a hardware board itself is also a cluster)
MasterComponent	Processor core, accelerator, or other master devices
SlaveComponent	Memory

The *cluster*, or *ComponentSet*, can be used to express not only a processor core cluster, but also a hardware board. It can also be extrapolated to represent a system composed of multiple boards – in this case, the outermost cluster is the system boundary itself.

2.2 Memory - AddressSpaceSet

A software program accesses memory through a logical window called the address space. Processor hardware usually supports multiple address spaces, for different access privileges, for example. An address space is further

¹ [http://en.wikipedia.org/wiki/Topology_\(electronics\)](http://en.wikipedia.org/wiki/Topology_(electronics))

subdivided into multiple subspaces, or address blocks. When a program makes an access somewhere in a memory device, it performs this by issuing a load or store instruction with its source or destination address falling into any one of the subspaces. To accommodate this memory setup, SHIM has a group of objects called *AddressSpaceSet*. An *AddressSpaceSet* can contain multiple *AddressSpace*, and each *AddressSpace* can contain multiple *SubSpace*.

A *SubSpace* is mapped to a physical memory device, or *SlaveComponent*, residing in some cluster, or *ComponentSet*. To describe the binding for which *SlaveComponent* is mapped to a specific *SubSpace*, the SHIM specification uses an object called *MasterSlaveBinding*. The object describes the mapping between a memory device and a memory subspace; it also indicates which *MasterComponent* (e.g., a processor core) has access to the memory. Since it is possible for multiple *MasterComponents* to have access to a memory *SubSpace*, a set object called *MasterSlaveBindingSet* is also defined to group multiple *MasterSlaveBinding* objects.

By exploring the objects under the *AddressSpaceSet*, a tool can discover what memory spaces are available and which processor core or accelerator has what kind of access to those. Multiple *AddressSpace/SubSpace* may share the same *SlaveComponent*. If the sharing occurs for only parts of the physical memory, it can be divided into multiple *SlaveComponents*.

2.3 Inter-core communication – CommunicationSet

For software to run on multiple processor cores and accelerators with some degree of cooperative manner, it often exchanges data, which may be available via a shared memory region. The software must also trigger, synchronize, or perform mutual exclusion in some way. In cases where shared memory is not available, some form of core-to-core or *MasterComponent* to *MasterComponent* communications is required. To accommodate this situation, SHIM defines a class of objects called *CommunicationSet*. All SHIM objects have a child object called *ConnectionSet*, which includes one or more *Connection* that describes the source and destination *MasterComponents* for the communication. The variety of *CommunicationSet* classes have similar communication mechanisms (Table 2).

Table 2. Inter-core communication classes

CommunicationSet classes	Description
SharedRegisterCommunication	Shared register based communication. Often such hardware provides a set of registers that can be accessed by multiple processor cores.
SharedMemoryCommunication	Shared memory based communications. An operation type is specified from TAS (Test and set), LLSC (Load-link/Store conditional), CAX (Compare and exchange), and OTHER (other unspecified operation).
EventCommunication	An event is often a register bitmap based communication – if a processor core raises an event (Boolean), that is sent to another core and can be seen as the mapped event signaled in its event register. It may or may not trigger an interrupt.
FIFOCommunication	A FIFO is sometimes used for inter-core communication and often implemented as FIFO registers, possibly with buffers of varying depth.
InterruptCommunication	This is a typical inter-processor-interrupt. This object only has the <i>ConnectionSet</i> .

Each class has its unique properties or attributes. All classes include connection information describing which pairs of cores are connected by the particular communication object. Since there can be multiple connections, the object contains *ConnectionSet*, which in turn contains any number of *Connection*. Each *Connection* contains references to a pair of *MasterComponents*.

Software tools can use this information to obtain the type of *MasterComponent*-to-*MasterComponent* communication mechanisms are supported by a particular hardware implementation represented by a SHIM XML. Note that the connection can be across multiple *ComponentSet* boundaries, even if it traverses the chip or hardware board boundaries.

2.4 Performance

2.4.1 General

As expected, different processor hardware has different performance characteristics. The performance characteristics can be very complex for a multi-many-core hardware and will have tremendous impact on the software design. Since SHIM's principle is to capture the properties that affect the software at the architectural design level, it is intrinsic to include such performance properties (Table 3).

Table 3. Performance properties in SHIM

Performance Property	Related SHIM Object	Description
Instruction execution	<i>CommonInstructionSet, Instruction</i>	The execution cycles of processor instructions. The instruction set is described as LLVM IR, and the cycles of a particular processor architecture is expressed in terms of these LLVM IR instructions.
Memory access	<i>SubSpace, MasterSlaveBinding, Accessor, AccessType</i>	The processor cycles for accessing a memory. Each processor core can have different cycles for different access types such as read or write and accessing by byte, word, double word accesses, etc.
Inter-core communication	<i>CommunicationSet</i> classes	The time needed for a particular connection of two <i>MasterComponent</i> for a particular <i>Communication</i> class, such as <i>InterruptCommunication</i> , in processor cycles.

There are significant performance variations among different processor, memory, and inter-connect architectures, so all performance properties are expressed as a triplet of best, typical, and worst cycles. Some architectures are highly deterministic and may have little variation in the performance of some operations; this will be depicted with the triplet bearing similar, if not the same, values. The software tool can use this information to determine the hardware dynamism or determinism by examining the deviation in the values. For such hardware, the estimation based on SHIM XML can be highly accurate, well under the 20% error rate that SHIM targets (even possibly nearing single digits of error percentage). Some hardware could have fairly dynamic performance characteristics, performing some operations mostly in two cycles, and possibly in 200 cycles in some cases, for example. This dynamic behavior often is derived from a wide range of speculative and probabilistic algorithms employed by modern hardware; this 'best-effort' approach as opposed to a 'guarantee' approach is quite popular and the trend continues.

SHIM, as said in [Software View - what is in and what is not](#), provides software with a simpler view of the underlying hardware and it avoids descriptions of what speculative algorithm is supported and its detailed spec. The triplet performance representation provides a window to adapt the dynamism by carefully setting up the three values, encapsulating the various hardware mechanism underneath. After all, it is technically infeasible, if not impossible, to achieve 100% accuracy in the hardware performance estimation – the idea is to obtain accurate enough performance estimation for system architectural design – the rest must be optimized in the later phase of system development. This approach is reasonable since the final set of software is unavailable before the system development stage and there are many other factors that influence the development, and thus the design, as the project progresses.

SHIM.xml is created for a specific hardware (and system software if necessary) configuration. If, for example, quality of service (QoS) is to impact the performance characteristics at a level greater than the goal of 20% error rate, multiple SHIM XML files must be authored or a [Common Configuration File \(CCF\)](#) must be used to describe the variation in performance.

2.4.2 Latency and Pitch

The performance object is characterized by a pair of triplets - one associated with ‘latency’, the other for the ‘pitch’ (Figure 6). The latency, or *Latency* in terms of SHIM class, is specifically the processor cycles for performing the particular operation. The *Pitch* is a trickier process – it is the size of stride when executing the operation in consecutive manner, also expressed in processor cycles. As indicated previously, modern hardware has a mechanism to speculate what would be the next software action. When accessing a memory, for example, the hardware has a cache that reads the memory in its line size, even if a smaller size of memory is requested by a particular ‘load’ instruction. In essence, it can read the next memory address ahead of time, hoping the next ‘load’ instruction will follow at the consecutive address (called a speculative fetch). If that fetch proves true, the next read operation can complete by reading from the cache, without actually accessing the slower main memory. The hardware supports other similar mechanisms – all trying to take advantage of repetitive software behavior. This action results in the performance characteristics that, if the similar operation is performed repeatedly in some way, the average execution cycles per operation are less than it would be if it is not. The *Pitch* is specifically meant to describe this performance property.

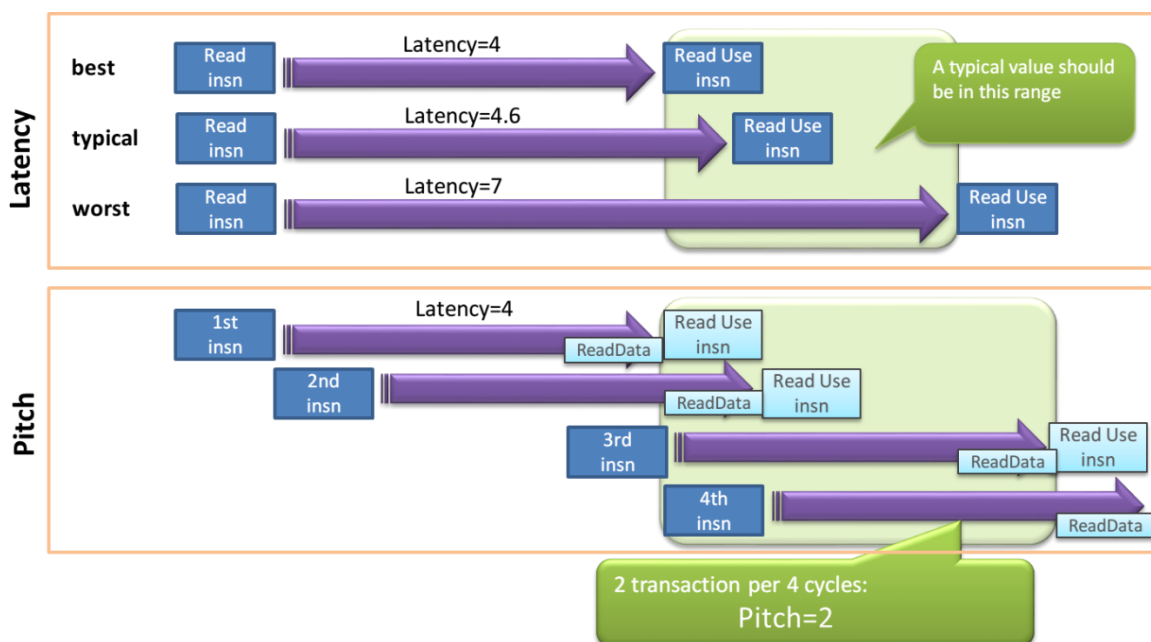


Figure 6. Latency and Pitch represent the primary performance characteristics.

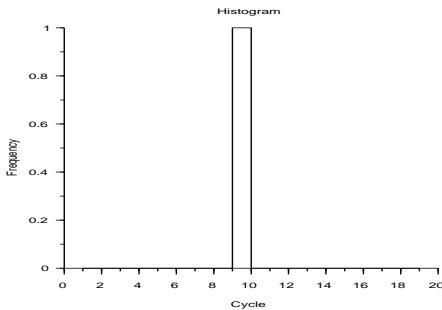
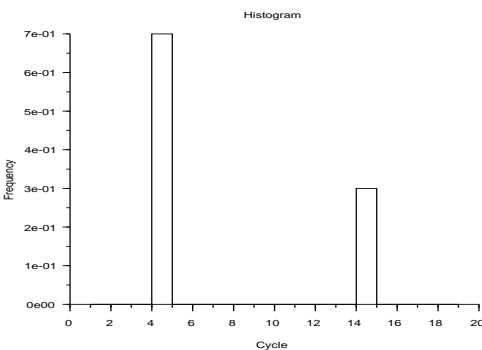
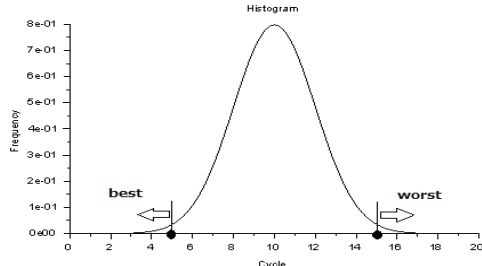
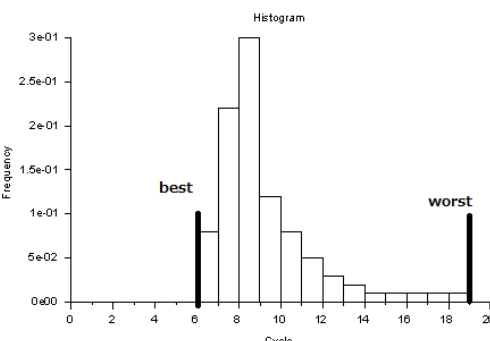
The software tool’s job is to see if a particular operation is repeated, and use the *Latency* and *Pitch* triplets accordingly.

2.4.3 Using triplets

Any factors which influence performance characteristics should be expressed in the triplet of ‘best’, ‘typical’ and ‘worst’ to describe the performance variations. This is described with examples in Table 4. Using Triples).

Please note that, statistically speaking, the ‘typical’ value is not the average but the mode.

Table 4. Using Triples

 <p>A histogram titled 'Histogram' with 'Frequency' on the y-axis (0 to 1) and 'Cycle' on the x-axis (0 to 20). A single sharp peak is centered at cycle 10, reaching a frequency of 1.0.</p>	<p>In the simple case in which a single number can describe the specific performance (the triplet has three equal values), the triplet notation is still useful to explicitly note that it has the three equal values.</p> <p>Example: 10.0, 10.0, 10.0</p>
 <p>A histogram titled 'Histogram' with 'Frequency' on the y-axis (0e00 to 7e-01) and 'Cycle' on the x-axis (0 to 20). There are two distinct peaks: one at cycle 5 with a frequency of approximately 0.7, and another at cycle 15 with a frequency of approximately 0.3.</p>	<p>When the distribution has two numbers, a triplet permits more precise expression. The 'typical' number represents the more frequent conditions.</p> <p>Example: 5.0, 5.0, 15.0</p>
 <p>A histogram titled 'Histogram' with 'Frequency' on the y-axis (0e00 to 8e-01) and 'Cycle' on the x-axis (0 to 20). A smooth Gaussian curve is overlaid on the histogram. Two points on the curve are marked with arrows and labeled 'best' (at cycle ~5.1) and 'worst' (at cycle ~14.9).</p>	<p>Gaussian distribution may be observed in an ideal case, where the 'typical' value equals the mean. The best and worst represent greater than or equal to three standard deviations away from the mean.</p> <p>Example: 5.1, 10.0, 14.9</p>
 <p>A histogram titled 'Histogram' with 'Frequency' on the y-axis (0e00 to 3e-01) and 'Cycle' on the x-axis (0 to 20). The distribution is skewed to the right. Two points are marked with vertical lines and labeled 'best' (at cycle ~6.0) and 'worst' (at cycle ~18.9).</p>	<p>On the real distribution, the measurement of the machine performance limit is sometimes very difficult. In such a case, adopt the 99.9% cumulative point (almost equal to the three standard deviation point on Gaussian distribution) of the distribution as the worst and the 0.1% cumulative point as the best. The typical value represents the mode of the distribution.</p> <p>Example: 6.0, 8.5, 18.9</p>

2.5 Software View – what is in and what is not

As said in [Introduction](#), tools should primarily use SHIM to aid developing software that runs on multi-many-core hardware. Therefore, the key strategy in defining the SHIM specification is to describe the hardware but only for the information that is relevant to such tools. We call this a 'software view' of hardware, as opposed to 'hardware view', where the focus would be the physical/electrical means of inter-connects between processing cores, the

NoC² protocol used to route the memory read request by a particular core, the number of processor pipeline stages, the cache coherency protocol, etc., unless these features matter greatly to some class of tools that aid software development.

It is tempting and relatively easy to include additional hardware properties in SHIM, however this will result in a more complex SHIM XML, requiring more effort to grasp the schema and complicating the effort for tools to use this information. Furthermore, the most critical issue is the challenge to create a SHIM XML in the first place – leading to limited adoption of the SHIM standard.

The basic principle is to capture the properties that affect the software at the architectural design level. This is to say, if a design-aid tool uses SHIM to produce an appropriate software design for a particular hardware described by an SHIM XML, then the design should not require modification at the software architectural level at the later stages of system development.

Although the “software architectural design level” is the baseline, it is sometimes difficult to agree on whether a particular hardware property is important. The rule of thumb is that if we cannot derive an actual (even imaginable) use case, the SHIM specification excludes it.

For various reasons, a number of potential hardware properties have not been included into the current specification. One of the primary reasons is that the excluded types of hardware properties are peripheral to existing properties in the specification. Such hardware properties may be included in a future version of the specification, but we decided to take an evolutionary approach and stabilize the more basic properties first.

The most basic properties selected for inclusion are the following: topology, address space, inter-core communication, and performance and configuration.

2.6 XML

The SHIM interface uses extensible markup language (XML); specifically, the SHIM XML uses the XML Schema (XSD) to define its XML structure. XSD is essentially the same as an UML class diagram. Each SHIM XML file represents a unique hardware, but all must conform to the SHIM XML schema. The UML class diagram representation of SHIM XML schema is shown in Figure 3.

The XML schema allows the definition of the SHIM XML structure, but with the help of a validating parser that reads XML files, the schema also allows validation of SHIM XML. Validating parsers are readily available, both openly and commercially, often bundled with various XML related libraries in many different programming languages.

Therefore, technically speaking, the SHIM XML schema, or the shim.xsd, is the core interface definition of SHIM.

2.6.1 Data Binding

A common technique to read XML files is via SAX or DOM libraries. Using XSD, it is possible to generate class libraries in many choices of programming languages by running a schema compiler against the shim.xsd. The generated library includes all the SHIM XML classes of the chosen programming language, with automatically added methods or functions to get and set the data. This allows tools to access hardware properties expressed in a SHIM XML similar to accessing normal objects in their programming languages.

2.6.2 Who Creates SHIM XML

The hardware provider is expected to create and provide the SHIM XML, which will then be used by the software tools. On the other hand, a hardware provider may not provide the SHIM XML. If SHIM XML can only be authored by the hardware provider, it can be a significant roadblock in the hardware’s adoption. Therefore, [Reference authoring tools](#) are made freely available along with the specification. If a user has access to the basic

² Network on Chip

technical reference manuals, and either simulator or actual hardware (e.g., evaluation board), the [Reference authoring tools](#) allows for the creation of the SHIM XML for most multi-many-core hardware in fewer than 1-2 days.

2.7 Configuration

2.7.1 General

There are two different aspects of configuration in SHIM that are needed by software tools. One aspect is the configuration of software tools based on the basic hardware properties (e.g., cluster organization, number of cores, memory size, processor ISA). These are static hardware properties and tools are able to read the SHIM XML file and configure themselves accordingly. The other aspect is configuring the hardware dynamic properties (e.g., clock frequency, various modes and setting for transfer accelerator) that can be modified according to the system design. For dynamic properties, the tools' user is often required to input the configuration, thus the tools must provide a user interface (either command line or graphical). SHIM provides a mechanism called [Common Configuration File \(CCF\)](#), to serve both for describing the configurable properties and also simultaneously defining the user interface.

Changing the configuration often affects the performance properties. The CCF is designed so that it can also describe how the selection or input value of particular configurable items affects the performance properties.

2.7.2 Common Configuration File (CCF)

The CCF extends SHIM to describe configurable hardware elements and also defines a standard way to generate configuration UI by the tools that support it. The CCF describes the configurable items in a file called CCF XML; this is a separate XML file from the SHIM XML. Software tools using SHIM can utilize this mechanism to provide a [Configuration tool user interface](#) within its tool, or as a separate standalone tool. When the configuration tool is executed, along with the SHIM XML and CCF, it provides a mechanism to modify the specific parts of SHIM XML, according to the inputs made by the tool user, which can also be automated by the tool.

The SHIM XML and CCF are inter-linked via XPath, the XML Path Language (a query language for selecting nodes from an XML document). In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document.

Here is an example of an actual CCF:

```

1 <?xml version="1.0" encoding="UTF-8"?>↓
2 <ConfigurationSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="CCF Sample for SHIM" xsi:noNamespac
eSchemaLocation="ccf-schema.xsd">↓
3   <DefineSet>↓
4     <Def name="@sclock" path="/SystemConfiguration/ClockFrequency/@clockValue" uri="shim_sample_data.xml"/>↓
5     <Def name="@cashSize" path="//Cache[@name='UnifiedCache_0_0_0']/@size" uri="shim_sample_data.xml"/>↓
6   </DefineSet>↓
7   <Configuration formType="select" name="System clockValue-Select" path="/SystemConfiguration/ClockFrequency/@clockV
alue" uri="shim_sample_data.xml">↓
8     <Item key="value" value="20.0"/>↓
9     <Item key="value" value="40.0"/>↓
10    <Item key="value" value="100.0"/>↓
11  </Configuration>↓
12  <Configuration formType="expression" name="Sample Expression" path="//MasterComponent/ClockFrequency/@clockValue
" uri="shim_sample_data.xml">↓
13    <Expression>↓
14      <description>description</description>↓
15      <Exp>@sclock * 2</Exp>↓
16    </Expression>↓
17  </Configuration>↓
18  <Configuration formType="text" name="Arch" path="//MasterComponent/@arch" uri="shim_sample_data.xml"/>↓
19  <Configuration formType="integer" name="nRegister" path="//SharedRegisterCommunication/@nRegister" uri="shim_samp
le_data.xml"/>↓
20  <Configuration formType="float" name="ClockFrequency:clockValue" path="/SystemConfiguration/ClockFrequency/@clockV
alue" uri="shim_sample_data.xml"/>↓
21  <Configuration formType="bool" name="BooleValue Sample"/>↓
22 </ConfigurationSet>↓
23 [EOF]

```

Figure 7. CCF example

This CCF, when opened by a CCF capable tool, will dynamically create a GUI like below (this is a CCF sample application available with source codes from MCA).

The screenshot shows a window titled "CCF Sample Application" with a menu bar containing "OpenCCF" and "Configure All". The main area contains six configuration sections, each with a "FormType" dropdown, a "Configure" button, a "Result" field, and an "enable" checkbox.

- System clockValue-Select**: FormType is "select", Result is "20.0", and "enable" is checked. Below it, a dropdown menu shows "20.0". To the right, the URI is "shim_sample_data.xml" and the XPath is "/SystemConfiguration/ClockFrequency/@clockValue".
- Sample Expression**: FormType is "expression", Result is "80.0", and "enable" is checked. Below it, a text box contains "@clock * 2". To the right, the URI is "shim_sample_data.xml" and the XPath is "//MasterComponent/ClockFrequency/@clockValue".
- Arch**: FormType is "text", Result is "Generic", and "enable" is checked. Below it, a text box contains "Generic". To the right, the URI is "shim_sample_data.xml" and the XPath is "//MasterComponent/@arch".
- nRegister**: FormType is "integer", Result is "32", and "enable" is checked. Below it, a text box contains "32". To the right, the URI is "shim_sample_data.xml" and the XPath is "//SharedRegisterCommunication/@nRegister".
- ClockFrequency:clockValue**: FormType is "float", Result is "40", and "enable" is checked. Below it, a text box contains "40". To the right, the URI is "shim_sample_data.xml" and the XPath is "/SystemConfiguration/ClockFrequency/@clockValue".
- BooleanValue Sample**: FormType is "bool", Result is "false", and "enable" is checked. Below it, a checkbox is unchecked and labeled "false". To the right, the URI is "*file*" and the XPath is "*xpath*".

Figure 8. GUI generated by CCF

Please refer to [Common Configuration File \(CCF\)](#).

2.8 Reference Authoring Tools

In addition to the specification itself, SHIM also provides a free set of reference authoring tools. As a reference, anyone can provide their own version of the SHIM authoring tools. The Multicore Association provides the reference authoring tool, SHIM Editor, for the following reasons:

1. Easy authoring of SHIM XML to enable better adoption
2. Serves as a sample SHIM application with source code

2.9 Roadmap

The first version of SHIM contains the fundamental and critical hardware properties that many tools will find useful. However, as mentioned in [Software view - what is in and what is not](#), some items have not been covered in this first version. SHIM is an open technology and wider adoption will fuel its innovative use; this may require enhancements to the specification and we would like to remain open to such changes.

Properties that are under consideration for future versions of the spec include: debug/trace, power consumption, and basic peripheral components. A few other items are worth mentioning such as componentization of SHIM XML, hardware-related software properties, and schema refinement for smaller XML.

2.9.1 Componentization of SHIM XML

The current version of SHIM must stand alone, meaning that a SHIM XML file should describe an executable hardware platform (e.g., a virtual simulator or an actual hardware board). However, a dedicated multi-many-core chip is rarely designed for each board, therefore, the same chip is often deployed in multiple boards. This means that separate SHIM XML files must exist for each board, though the SHIM XML description of the multi-many-core chip description is redundant for these two files.

Once SHIM's use starts to spread, it is natural to reuse a particular component description in a SHIM XML file among multiple SHIM XML files. This is essentially componentization of SHIM XML, a feature already under consideration for inclusion in the next major version of SHIM. Meanwhile, one can use an existing SHIM XML file resembling your target hardware as a basis of authoring a new SHIM XML file. SHIM Editor indeed supports editing of an existing SHIM XML file.

A challenge of componentizing SHIM XML is that a SHIM XML class, such as *MasterComponent*, contains properties that are static for the hardware board design it is being included with, and also properties that may differ depending on how it is integrated into a particular hardware board. The two must be decoupled in order to reuse the SHIM XML description of the *MasterComponent*. One idea is to use [Common Configuration File \(CCF\)](#), which allows for adjusting the performance value based on some other information, as long as it can be found in the final SHIM XML file or somewhere within the CCF.

Also along with the componentization of SHIM XML itself, we are investigating the possibility to align SHIM with IP-XACT where appropriate, to ease SHIM XML authoring tools ability to support importing IP-XACT XML files to semi-automate authoring a SHIM XML using the relevant information contained in IP-XACT. The objects contained in the *ComponentSet*, at least the topology part and the object names, should be importable, while others are unique to SHIM XML and must be added.

2.9.2 Hardware-Related Software Properties

In addition to the hardware properties that SHIM describes, some tools have a dependency on the system software properties such as the operating system and even some middleware. For example, for a parallelization design aid tool such as a parallelizing compiler, the performance of OS mutual exclusion primitives is critical in deciding on an appropriate lock mechanism for particular processing. Similarly, the tool may need to know the performance of some message passing mechanism. Currently, this kind of information is not included in SHIM, partly because it will require separate SHIM XML files for different system software implementations, along with the library interface definition; a future version of SHIM may extend its coverage into this kind of information.

2.9.3 Schema Refinement for Smaller XML

The SHIM schema is intended to be simple, while allowing it to support both homogeneous and heterogeneous hardware. This has led to using repetitive sets of lines in the XML for homogenous hardware that have multiple instances of the same component, like a hardware composed of multiple instances of the same cluster configuration. If the clusters are heterogeneous, with each cluster having a different configuration of processing cores, then the number of XML lines does not change but they will have different lines. If SHIM can provide a mechanism to express the redundancy in the schema, the size of SHIM XML file for homogenous hardware can be reduced. We intend to consider this along with [Componentization of SHIM XML](#).

3. SHIM Interface

The major part of the SHIM interface is the SHIM XML schema itself. Therefore, understanding the schema comprises the major part of understanding the interface. The basics are described in the chapter [SHIM Concepts](#) and it assumes the schema is divided into the following groups:

- *Enumeration*
- *SystemConfiguration*
- *ComponentSet*
- *AddressSpaceSet*
- *CommunicationSet*

For each group above, the schema and the description are explained in the following sections. For each object or XML element contained in each group, the description and example XML are provided.

The schema is converted into different programming language bindings, using various schema compilers. The SHIM specification does not specify the programming language as this can vary according to the nature of the tools and intended use cases. However, Java is assumed to be one of the primary languages used and the Java class library interface of SHIM, called the SHIM API library, is also provided. Some utility interfaces are defined along with the reference implementation in Java to further ease the programming using of the SHIM class libraries.

The following sections describe each part, detailing the XML elements and their attributes, along with a pointer to the Java class library interface.

3.1 **shim.xsd**

The SHIM XML schema file. Please refer to the following sections for description of elements.

```

<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ComponentSet" type="ComponentSet"/>
  <xs:complexType name="ComponentSet">
    <xs:sequence>
      <xs:element name="ComponentSet" type="ComponentSet" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="SlaveComponent" type="SlaveComponent" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="MasterComponent" type="MasterComponent" minOccurs="0"
maxOccurs="unbounded"/>
      <xs:element name="Cache" type="Cache" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" use="required" type="xs:string"/>
  </xs:complexType>
  <xs:element name="SlaveComponent" type="SlaveComponent"/>
  <xs:complexType name="SlaveComponent">
    <xs:annotation>
      <xs:documentation>Memory</xs:documentation>
    </xs:annotation>
    <xs:sequence/>
    <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="id" use="required" type="xs:ID"/>
    <xs:attribute name="size" use="required" type="xs:int"/>
    <xs:attribute name="sizeUnit" use="required" type="SizeUnitType"/>
    <xs:attribute name="rwType" use="required" type="RWType"/>
  </xs:complexType>
  <xs:element name="MasterComponent" type="MasterComponent"/>
  <xs:complexType name="MasterComponent">
    <xs:sequence>
      <xs:element name="CommonInstructionSet" type="CommonInstructionSet" minOccurs="0"
maxOccurs="1"/>
      <xs:element name="Cache" type="Cache" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="ClockFrequency" type="ClockFrequency" minOccurs="0"
maxOccurs="1"/>
      <xs:element name="AccessTypeSet" type="AccessTypeSet" minOccurs="1"
maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="id" use="required" type="xs:ID"/>
    <xs:attribute name="masterType" use="required" type="MasterType"/>
    <xs:attribute name="arch" use="required" type="xs:string"/>
    <xs:attribute name="archOption" use="optional" type="xs:string"/>
    <xs:attribute name="pid" use="optional" type="xs:string"/>
    <xs:attribute name="nThread" use="optional" type="xs:int"/>
    <xs:attribute name="endian" use="optional" type="EndianType"/>
  </xs:complexType>
  <xs:simpleType name="RWType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="RW"/>
      <xs:enumeration value="WX"/>
      <xs:enumeration value="RX"/>
      <xs:enumeration value="R"/>
      <xs:enumeration value="W"/>
      <xs:enumeration value="X"/>
      <xs:enumeration value="RWX"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="AddressSpaceSet" type="AddressSpaceSet"/>
  <xs:complexType name="AddressSpaceSet">
    <xs:sequence>
      <xs:element name="AddressSpace" type="AddressSpace" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="AddressSpace" type="AddressSpace"/>
  <xs:complexType name="AddressSpace">
    <xs:sequence>
      <xs:element name="SubSpace" type="SubSpace" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="id" use="required" type="xs:ID"/>
  </xs:complexType>
  <xs:element name="SubSpace" type="SubSpace"/>
  <xs:complexType name="SubSpace">

```

```

    <xs:sequence>
      <xs:element name="MemoryConsistencyModel" type="MemoryConsistencyModel"
minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="MasterSlaveBindingSet" type="MasterSlaveBindingSet"
minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="id" use="required" type="xs:ID"/>
    <xs:attribute name="start" use="required" type="xs:long"/>
    <xs:attribute name="end" use="required" type="xs:long"/>
    <xs:attribute name="endian" use="optional" type="EndianType"/>
  </xs:complexType>
  <xs:simpleType name="MasterType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="PU">
        <xs:annotation>
          <xs:documentation>Processing Unit</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="TU">
        <xs:annotation>
          <xs:documentation>Transfer Unit</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="OTHER"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="Instruction" type="Instruction"/>
  <xs:complexType name="Instruction">
    <xs:sequence>
      <xs:element name="Performance" type="Performance" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" use="required" type="xs:string"/>
  </xs:complexType>
  <xs:element name="InterruptCommunication" type="InterruptCommunication"/>
  <xs:complexType name="InterruptCommunication">
    <xs:complexContent>
      <xs:extension base="AbstractCommunication">
        <xs:sequence/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="Latency" type="Latency"/>
  <xs:complexType name="Latency">
    <xs:complexContent>
      <xs:extension base="AbstractPerformance">
        <xs:sequence/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="AbstractPerformance" type="AbstractPerformance"/>
  <xs:complexType name="AbstractPerformance" abstract="true">
    <xs:sequence/>
    <xs:attribute name="best" use="optional" type="xs:float"/>
    <xs:attribute name="typical" use="required" type="xs:float"/>
    <xs:attribute name="worst" use="optional" type="xs:float"/>
  </xs:complexType>
  <xs:element name="Pitch" type="Pitch"/>
  <xs:complexType name="Pitch">
    <xs:complexContent>
      <xs:extension base="AbstractPerformance">
        <xs:sequence/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="MasterSlaveBinding" type="MasterSlaveBinding"/>
  <xs:complexType name="MasterSlaveBinding">
    <xs:sequence>
      <xs:element name="Accessor" type="Accessor" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="slaveComponentRef" use="required" type="xs:IDREF"/>
  </xs:complexType>
  <xs:element name="CommunicationSet" type="CommunicationSet"/>
  <xs:complexType name="CommunicationSet">
    <xs:sequence>

```



```

        <xs:element name="SharedRegisterCommunication" type="SharedRegisterCommunication"
minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="SharedMemoryCommunication" type="SharedMemoryCommunication"
minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="EventCommunication" type="EventCommunication" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="FIFOCommunication" type="FIFOCommunication" minOccurs="0"
maxOccurs="unbounded"/>
        <xs:element name="InterruptCommunication" type="InterruptCommunication"
minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="AbstractCommunication" type="AbstractCommunication"/>
<xs:complexType name="AbstractCommunication" abstract="true">
    <xs:sequence>
        <xs:element name="ConnectionSet" type="ConnectionSet" minOccurs="0"
maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" use="required" type="xs:string"/>
</xs:complexType>
<xs:element name="Connection" type="Connection"/>
<xs:complexType name="Connection">
    <xs:sequence>
        <xs:element name="Performance" type="Performance" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="from" use="required" type="xs:IDREF">
        <xs:annotation>
            <xs:documentation>Reference to the instance of
MasterComponent</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="to" use="required" type="xs:IDREF">
        <xs:annotation>
            <xs:documentation>Reference to the instance of
MasterComponent</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:complexType>
<xs:element name="PerformanceSet" type="PerformanceSet"/>
<xs:complexType name="PerformanceSet">
    <xs:sequence>
        <xs:element name="Performance" type="Performance" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="FIFOCommunication" type="FIFOCommunication"/>
<xs:complexType name="FIFOCommunication">
    <xs:complexContent>
        <xs:extension base="AbstractCommunication">
            <xs:sequence>
                <xs:attribute name="dataSize" use="required" type="xs:int"/>
                <xs:attribute name="dataSizeUnit" use="optional" type="SizeUnitType"/>
                <xs:attribute name="queueSize" use="required" type="xs:int"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:element name="CommonInstructionSet" type="CommonInstructionSet"/>
<xs:complexType name="CommonInstructionSet">
    <xs:sequence>
        <xs:element name="Instruction" type="Instruction" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" use="required" type="xs:string"/>
</xs:complexType>
<xs:element name="Cache" type="Cache"/>
<xs:complexType name="Cache">
    <xs:sequence>
        <xs:element name="cacheRef" type="xs:IDREF" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="id" use="required" type="xs:ID"/>
    <xs:attribute name="cacheType" use="required" type="CacheType">
        <xs:annotation>
            <xs:documentation>soft / hard</xs:documentation>
        </xs:annotation>
    </xs:attribute>
</xs:complexType>

```

```

        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="cacheCoherency" use="required" type="CacheCoherencyType"/>
    <xs:attribute name="size" use="required" type="xs:int"/>
    <xs:attribute name="sizeUnit" use="required" type="SizeUnitType"/>
    <xs:attribute name="nWay" use="optional" type="xs:int"/>
    <xs:attribute name="lineSize" use="optional" type="xs:int"/>
    <xs:attribute name="lockDownType" use="optional" type="LockDownType"/>
</xs:complexType>
<xs:element name="SystemConfiguration" type="SystemConfiguration"/>
<xs:complexType name="SystemConfiguration">
    <xs:sequence>
        <xs:element name="ComponentSet" type="ComponentSet" minOccurs="1" maxOccurs="1"/>
        <xs:element name="CommunicationSet" type="CommunicationSet" minOccurs="0"
maxOccurs="1"/>
        <xs:element name="AddressSpaceSet" type="AddressSpaceSet" minOccurs="0"
maxOccurs="1"/>
        <xs:element name="ClockFrequency" type="ClockFrequency" minOccurs="1"
maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="shimVersion" use="required" type="xs:string"/>
</xs:complexType>
<xs:element name="ConnectionSet" type="ConnectionSet"/>
<xs:complexType name="ConnectionSet">
    <xs:sequence>
        <xs:element name="Connection" type="Connection" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:simpleType name="CacheCoherencyType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="SOFT"/>
        <xs:enumeration value="HARD"/>
    </xs:restriction>
</xs:simpleType>
<xs:element name="MemoryConsistencyModel" type="MemoryConsistencyModel"/>
<xs:complexType name="MemoryConsistencyModel">
    <xs:sequence/>
    <xs:attribute name="rawOrdering" use="optional" type="OrderingType">
        <xs:annotation>
            <xs:documentation>Read After Write</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="warOrdering" use="optional" type="OrderingType">
        <xs:annotation>
            <xs:documentation>Write After Read</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="wawOrdering" use="optional" type="OrderingType">
        <xs:annotation>
            <xs:documentation>Write After Write</xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="rarOrdering" use="optional" type="OrderingType"/>
</xs:complexType>
<xs:simpleType name="OrderingType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="ORDERD"/>
        <xs:enumeration value="UNORDERD"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="EndianType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="LITTLE"/>
        <xs:enumeration value="BIG"/>
    </xs:restriction>
</xs:simpleType>
<xs:element name="SharedRegisterCommunication" type="SharedRegisterCommunication"/>
<xs:complexType name="SharedRegisterCommunication">
    <xs:complexContent>
        <xs:extension base="AbstractCommunication">
            <xs:sequence/>
            <xs:attribute name="dataSize" use="required" type="xs:int"/>
            <xs:attribute name="dataSizeUnit" use="required" type="SizeUnitType"/>

```

```

        <xs:attribute name="nRegister" use="required" type="xs:int"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="SharedMemoryCommunication" type="SharedMemoryCommunication"/>
  <xs:complexType name="SharedMemoryCommunication">
    <xs:complexContent>
      <xs:extension base="AbstractCommunication">
        <xs:sequence>
          <xs:attribute name="operationType" use="optional" type="OperationType"/>
          <xs:attribute name="dataSize" use="optional" type="xs:int"/>
          <xs:attribute name="dataSizeUnit" use="optional" type="SizeUnitType"/>
          <xs:attribute name="addressSpaceRef" use="optional" type="xs:IDREF"/>
          <xs:attribute name="subSpaceRef" use="optional" type="xs:IDREF"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="EventCommunication" type="EventCommunication"/>
  <xs:complexType name="EventCommunication">
    <xs:complexContent>
      <xs:extension base="AbstractCommunication">
        <xs:sequence>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="ClockFrequency" type="ClockFrequency"/>
  <xs:complexType name="ClockFrequency">
    <xs:sequence minOccurs="1"/>
    <xs:attribute name="clockValue" use="required" type="xs:float"/>
  </xs:complexType>
  <xs:element name="Accessor" type="Accessor"/>
  <xs:complexType name="Accessor">
    <xs:sequence>
      <xs:element name="PerformanceSet" type="PerformanceSet" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="masterComponentRef" use="required" type="xs:IDREF"/>
  </xs:complexType>
  <xs:element name="AccessType" type="AccessType"/>
  <xs:complexType name="AccessType">
    <xs:sequence minOccurs="1"/>
    <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="id" use="required" type="xs:ID"/>
    <xs:attribute name="rwType" use="optional" type="RWType"/>
    <xs:attribute name="accessByteSize" use="optional" type="xs:int"/>
    <xs:attribute name="alignmentByteSize" use="optional" type="xs:int"/>
    <xs:attribute name="nBurst" use="optional" type="xs:int"/>
  </xs:complexType>
  <xs:element name="MasterSlaveBindingSet" type="MasterSlaveBindingSet"/>
  <xs:complexType name="MasterSlaveBindingSet">
    <xs:sequence>
      <xs:element name="MasterSlaveBinding" type="MasterSlaveBinding" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="CacheType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="DATA"/>
      <xs:enumeration value="INSTRUCTION"/>
      <xs:enumeration value="UNIFIED"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="Performance" type="Performance"/>
  <xs:complexType name="Performance">
    <xs:sequence>
      <xs:element name="Pitch" type="Pitch" minOccurs="1" maxOccurs="1"/>
      <xs:element name="Latency" type="Latency" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="accessTypeRef" use="optional" type="xs:IDREF"/>
  </xs:complexType>
  <xs:element name="AccessTypeSet" type="AccessTypeSet"/>
  <xs:complexType name="AccessTypeSet">
    <xs:sequence minOccurs="1" maxOccurs="1">
      <xs:element name="AccessType" type="AccessType" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

```

```

    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="SizeUnitType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="KiB"/>
      <xs:enumeration value="B"/>
      <xs:enumeration value="GiB"/>
      <xs:enumeration value="MiB"/>
      <xs:enumeration value="TiB"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="LockDownType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="LINE"/>
      <xs:enumeration value="NONE"/>
      <xs:enumeration value="WAY"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="OperationType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="TAS">
        <xs:annotation>
          <xs:documentation>Test and Set</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="LLSC">
        <xs:annotation>
          <xs:documentation>Load Link/Store Conditional</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="CAX">
        <xs:annotation>
          <xs:documentation>Compare and Exchange</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="OTHER"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>

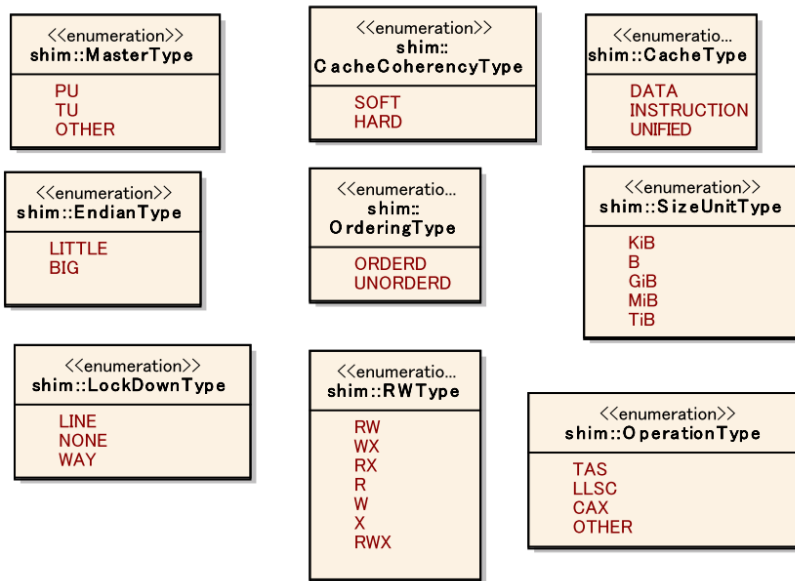
```

3.2 Conventions

- The interface is grouped into [Enumeration](#), [SystemConfiguration](#), [ComponentSet](#), [AddressSpaceSet](#), and [CommunicationSet](#). Each group has **SCHEMA** and **DESCRIPTION**.
- Each group describes its objects in separate subsections. These have **DESCRIPTION** and **EXAMPLE**.
- The objects and attributes use **bold** style, and the types use *italic*.

3.3 Enumeration

SCHEMA



DESCRIPTION

Enumeration is a special group which defines various constants used in some of the SHIM object attributes. The objects use the constants as values for selected attributes. When the attributes take one enumeration as its value, its attribute types specify which enumeration type it uses.

The following enumeration types are defined:

- **MasterType** specifies a type of *MasterComponent*. The values can be one of PU (Processor Unit, such as CPU), TU (Transfer Unit, such as DMA), or OTHER.
- **EndianType** specifies the endian, or byte-order.
- **LockDownType** specifies the type of supported cache content lockdown operation. The values can be one of LINE for line-lockdown, WAY for way-lockdown, and NONE if the lockdown is not supported.
- **CacheCoherencyType** specifies the type of cache coherency mechanism supported. It can be either HARD for hardware-based coherency or SOFT for software-based coherency.
- **OrderingType** specifies the memory consistency model. It can be ORDERED for ordered memory consistency or UNORDERED for unordered memory consistency.
- **RWType** specifies memory access types, which can be R for read, W for write, X for execute, RW for both R and W, RWX for all of R, W and X, WX for both W and X, and RX for both R and X.
- **CacheType** specifies the type of cache, which can be DATA for data cache, INSTRUCTION for instruction cache, and UNIFIED for a unified cache.
- **SizeUnitType** specifies the unit for data size, which can be B for byte, KiB for kilo binary byte, MiB for mega binary byte, GiB for giga binary byte, and TiB for tera binary byte.

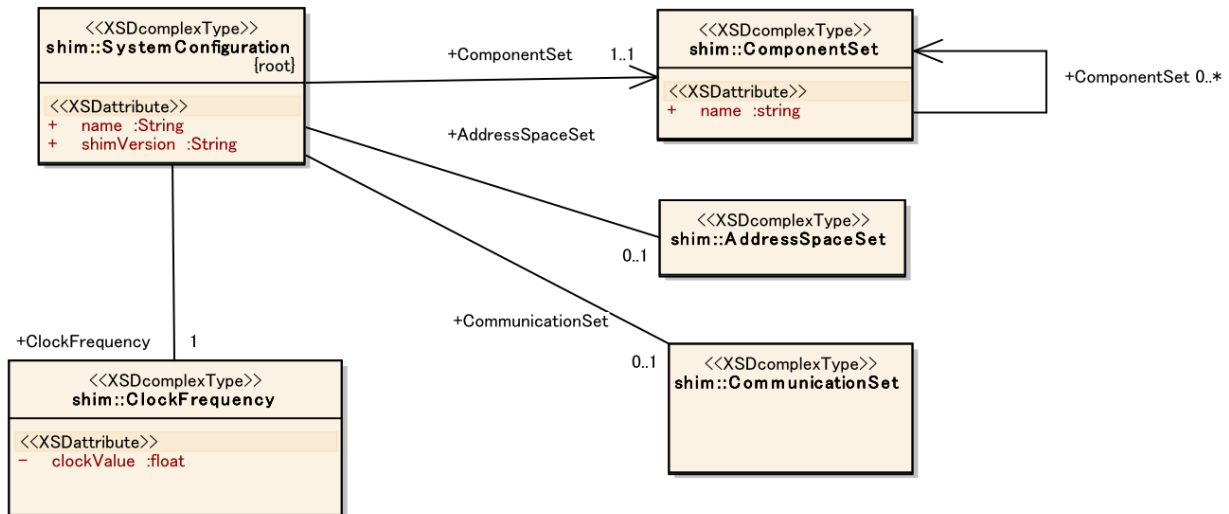
- ***OperationType*** specifies the type of shared memory communication, which can be TAS for Test and Set, LLSC for Load-link/Store Conditional, CAX for Compare and Exchange, and OTHER for other unspecified operation.

EXAMPLE

See examples for objects that use these types in the following sections.

3.4 SystemConfiguration

SCHEMA



DESCRIPTION

The *SystemConfiguration* is a root object. All SHIM XML has this object as its root.

- **SystemConfiguration** (mandatory): the root object; it has name of type string. It has one *ComponentSet*, *ClockFrequency*, and zero or more *AddressSpaceSet* and *CommunicationSet*.
- **name** (mandatory; type: *string*): the name of this SHIM description.
- **shim Version** (mandatory; type: *string*): the version SHIM interface specification. For this version of SHIM interface, it is “1.0”. It may be trailed with minor revision numbers (e.g., “1.0.1”).
- Refer to [ComponentSet](#), [AddressSpaceSet](#) and [CommunicationSet](#).

EXAMPLE

```

<SystemConfiguration name="System" shimVersion="1.0">
  <ComponentSet name="Cluster_0">
    </ComponentSet>
  <CommunicationSet>
    </CommunicationSet>
  <AddressSpaceSet>
    </AddressSpaceSet>
  <ClockFrequency clockValue="1.0E8"/>
</SystemConfiguration>
  
```

3.4.1 ClockFrequency

DESCRIPTION

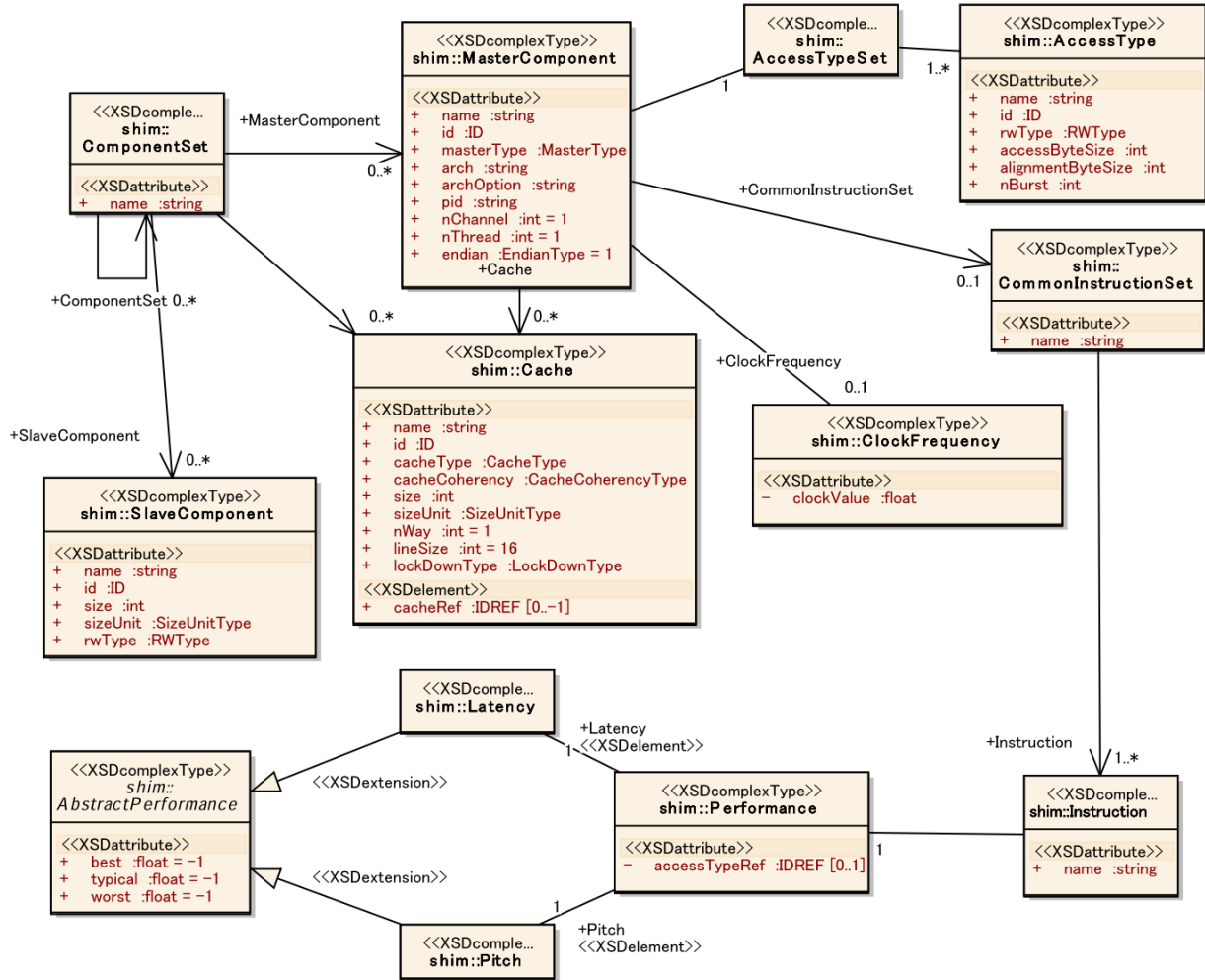
ClockFrequency: the system clock frequency with the following objects and/or attributes.

- **clockValue** (mandatory; type: *float*): the clock frequency value in Hz.
- **EXAMPLE**

See [SystemConfiguration](#).

3.5 ComponentSet

SCHEMA



DESCRIPTION

ComponentSet is the root of the hardware component topology description that SHIM contains. It has a **name** (mandatory) attribute. It may have *MasterComponent*, *Cache*, *SlaveComponent*, and another *ComponentSet*.

3.5.1 MasterComponent

DESCRIPTION

MasterComponent is a processor core, accelerator (including DMA accelerator), or any other type of component that can be a master. It has the following objects and/or attributes.

- AccessTypeSet** (mandatory): refer to [AccessTypeSet](#).
- CommonInstructionSet** (optional): refer to [CommonInstructionSet](#).
- name** (mandatory; type *string*): the name of this object. It should follow the same text used in the hardware reference manual.

- **id** (mandatory; type *ID*): the ID of this object.
- **masterType** (mandatory): the type of master and *MasterType*
- **arch** (mandatory; type *string*): specifies the name of this component's architecture and is intended mostly for describing processor instruction architecture. It is advised to use the official identifier for the ISA generally found in the architecture reference manual or similar.
- **archOption** (optional): specifies additional architecture properties.
- **pid** (optional): the ID of this *MasterComponent*. It is intended to be used for processor core id when the processor has some way of identifying the processor core when there are multiple cores. The scheme for describing the processor ID can be different, and it should follow the semantics used in the architecture reference manual of the processor. Since ID may not be expressed by an integer or single integer, this attribute is of type *string*.
- **nChannel** (optional; type *int*): specifies the number of channels and is intended for describing a number of channels of DMA, when **masterType** is **TU**.
- **nThread** (optional; type *int*): specifies the number of hardware thread, and intended for a processor core that supports hardware-threading.
- **translation** (optional; type *string*): specifies if address translation is supported. It is intended for describing when a processor supports some address translation unit such as MMU.
- **protection** (optional; type *string*): specifies the supported types of protection and is intended for describing the protection type supported by the processor.
- **endian** (optional; type *EndianType*): the endianness of this object.

EXAMPLE

```
<MasterComponent name="Core_0_0_0" id="SHIMEDITOR25331849408820141005130142974" masterType="PU"
  arch="Generic" archOption="" pid="16" nChannel="16" nThread="1" endian="LITTLE">
  <CommonInstructionSet name="LLVM Instructions">
    <Instruction name="ret">
      <Performance>
        <Pitch best="10.0" typical="10.0" worst="10.0"/>
        <Latency best="10.0" typical="10.0" worst="10.0"/>
      </Performance>
    </Instruction>
    ...
  </CommonInstructionSet>
  <Cache name="UnifiedCache_0_0_0" id="SHIMEDITOR8622411901820141005130145548"
    cacheType="UNIFIED" cacheCoherency="SOFT" size="64" sizeUnit="KiB" nWay="16" lineSize="128"
    lockDownType="LINE"/>
  <ClockFrequency clockValue="0.0"/>
  <AccessTypeSet>
    <AccessType name="AT_0_0_0_0" id="SHIMEDITOR27237422393120141005130145551" rwType="R"
      accessByteSize="4" alignmentByteSize="4" nBurst="8"/>
    ...
  </AccessTypeSet>
</MasterComponent>
```

3.5.2 SlaveComponent

DESCRIPTION

SlaveComponent is for describing a slave device such as memory. It has the following objects and/or attributes:

- **name** (mandatory; type *string*): the name of this object.
- **id** (mandatory; type *ID*): the ID of this object.

- **size** (mandatory; type *int*): the size of this memory.
- **sizeUnit** (mandatory; type *SizeUnitType*): the unit of **size**.
- **rwType** (mandatory; type *RWType*): specifies this memory is readable and/or writable.

EXAMPLE

```
<SlaveComponent name="Memory_0_0_0" id="SHIMEDITOR8181774865020141005130142975" size="128"
  sizeUnit="KiB" rwType="RW"/>
```

3.5.3 Cache**DESCRIPTION**

This object describes a cache with the following objects and/or attributes:

- **cacheRef** (optional; type *IDREF*): specifies the **id** of another **Cache** that is one level away from **MasterComponent**.
- **name** (mandatory; type *string*): the name of this object.
- **id** (mandatory; type *ID*): the ID of this object.
- **cacheType** (mandatory; type *CacheType*): specifies this cache type.
- **cacheCoherency** (mandatory; type *CacheCoherencyType*): specifies what cache coherency mechanism is provided.
- **size** (mandatory; type *int*): this cache size.
- **sizeUnit** (mandatory; type *SizeUnitType*): the unit of **size**.
- **nWay** (optional; type *int*): specifies the number of cache ways.
- **lineSize** (optional; type *int*): specifies the cache line size.
- **lockDownType** (optional; type *LockDownType*): specifies the supported cache lock down operation.

EXAMPLE

```
<Cache name="UnifiedCache_0_0_0" id="SHIMEDITOR8622411901820141005130145548"
  cacheType="UNIFIED" cacheCoherency="SOFT" size="64" sizeUnit="KiB" nWay="16" lineSize="128"
  lockDownType="LINE"/>
```

3.5.4 AccessTypeSet**DESCRIPTION**

This object bundles one or more [AccessType](#). It has the following objects and/or attributes:

- **AccessType** (mandatory): refer to [AccessType](#).

EXAMPLE

```

<AccessTypeSet>
  <AccessType name="AT_0_0_0_0" id="SHIMEDITOR27237422393120141005130145551" rwType="R"
    accessByteSize="4" alignmentByteSize="4" nBurst="8"/>
  <AccessType name="AT_0_0_0_1" id="SHIMEDITOR29805129821320141005130145552" rwType="R"
    accessByteSize="8" alignmentByteSize="8" nBurst="8"/>
</AccessTypeSet>

```

3.5.5 AccessType**DESCRIPTION**

This object describes the type of access, mostly intended for, but not limited to, memory access by a processor. It has the following objects and/or attributes.

- **name** (mandatory; type *string*): the name of this object.
- **id** (mandatory; type *ID*): the ID of this object.
- **rwType** (optional; type *RWType*): specifies the type of access.
- **accessByteSize** (optional; type *int*): specifies the data size of access in bytes.
- **alignmentByteSize** (optional; type *int*) specifies the alignment requirement in byte of this access.
- **nBurst** (optional; type *int*): specifies the burst length. The burst size is accessByteSize. It is mostly intended for masterType=TU.

EXAMPLE

See [AccessTypeSet](#).

3.5.6 CommonInstructionSet**DESCRIPTION**

This object contains [Instruction](#), which describes the instruction supported by the [MasterComponent](#). It is not explicitly described by the XML schema, however, it must always have LLVM instruction set³ and this can be extended if necessary. Each instruction of the LLVM instruction set corresponds to an instruction or a sequence of instructions of the [MasterComponent](#). It has the following objects and/or attributes:

- **Instruction** (mandatory): refer to [Instruction](#).
- **name** (mandatory; type *string*): the name of this object.

³ <http://llvm.org/docs/LangRef.html#instruction-reference>

EXAMPLE

```

1. <CommonInstructionSet name="LLVM Instructions">
  <Instruction name="ret">
    <Performance>
      <Pitch best="10.0" typical="10.0" worst="10.0"/>
      <Latency best="10.0" typical="10.0" worst="10.0"/>
    </Performance>
  </Instruction>
  <Instruction name="br">
    <Performance>
      <Pitch best="10.0" typical="10.0" worst="10.0"/>
      <Latency best="10.0" typical="10.0" worst="10.0"/>
    </Performance>
  </Instruction>
  ...
  <Instruction name="landingpad">
    <Performance>
      <Pitch best="10.0" typical="10.0" worst="10.0"/>
      <Latency best="10.0" typical="10.0" worst="10.0"/>
    </Performance>
  </Instruction>
</CommonInstructionSet>

```

3.5.7 Instruction**DESCRIPTION**

This describes the instruction. It has following objects and/or attributes:

- **Performance** (mandatory): refer to [Performance](#).
- **name** (mandatory; type *string*): the name of this object.

EXAMPLE

See [CommonInstructionSet](#).

3.5.8 Performance**DESCRIPTION**

This object describes performance. It has the following objects and/or attributes:

- **Latency** (mandatory): refer to [Latency](#).
- **Pitch** (mandatory): refer to [Pitch](#).
- **accessTypeRef** (optional; type *IDREF*) a reference to [AccessType](#) **id**. This is intended to be used when describing the performance of memory access.

EXAMPLE

See [CommonInstructionSet](#).

3.5.9 Latency**DESCRIPTION**

It has the following objects and/or attributes. Refer to [Latency and Pitch](#).

- **best** (optional; type *float*): the number of processor cycles for the best-case latency.

- **typical** (mandatory; type *float*): the number of processor cycles for the typical latency.
- **worst** (optional; type *float*): the number of processor cycles for the worst-case latency.

EXAMPLE

See [CommonInstructionSet](#).

3.5.10 Pitch**DESCRIPTION**

It has following objects and/or attributes:

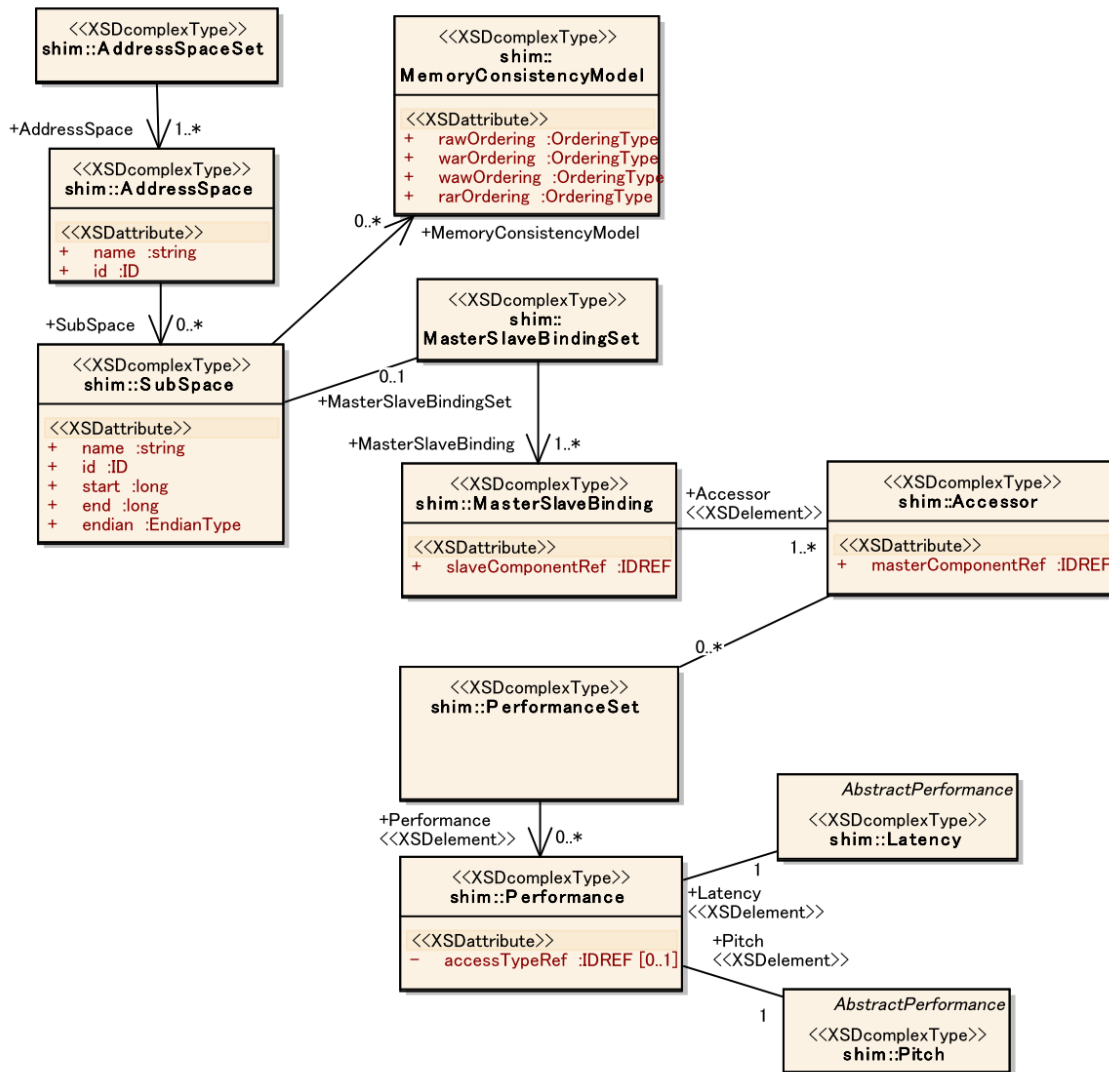
- **best** (optional; type *float*): the number of processor cycles for the best-case pitch.
- **typical** (mandatory; type *float*): the number of processor cycles for the typical pitch.
- **worst** (optional; type *float*): the number of processor cycles for the worst-case pitch.

EXAMPLE

See [CommonInstructionSet](#).

3.6 AddressSpaceSet

SCHEMA



DESCRIPTION

AddressSpaceSet describes how the memory address spaces are organized and which *MasterComponent* is bound to which *SlaveComponent*.

3.6.1 AddressSpace

DESCRIPTION

It has following objects and/or attributes:

- **SubSpace** (optional): refer to [SubSpace](#).
- **name** (mandatory; type *string*): the name of this object.
- **id** (mandatory; type *ID*): the ID of this object.

EXAMPLE

```

<AddressSpaceSet>
  <AddressSpace name="AS_0_0" id="SHIMEDITOR22375265206920141005130143354">
    <SubSpace name="SS_0_0_0" id="SHIMEDITOR9140938132320141005130143355" start="0" end="128"
      endian="LITTLE">
      <MemoryConsistencyModel rawOrdering="ORDERD" warOrdering="ORDERD" wawOrdering="ORDERD"
        rarOrdering="ORDERD"/>
      <MasterSlaveBindingSet>
        <MasterSlaveBinding slaveComponentRef="SHIMEDITOR8181774865020141005130142975">
          <Accessor masterComponentRef="SHIMEDITOR25331849408820141005130142974">
            <PerformanceSet>
              <Performance accessTypeRef="SHIMEDITOR27237422393120141005130145551">
                <Pitch best="10.0" typical="10.0" worst="10.0"/>
                <Latency best="10.0" typical="10.0" worst="10.0"/>
              </Performance>
              <Performance accessTypeRef="SHIMEDITOR29805129821320141005130145552">
                <Pitch best="10.0" typical="10.0" worst="10.0"/>
                <Latency best="10.0" typical="10.0" worst="10.0"/>
              </Performance>
            </Accessor>
            ...
          </MasterSlaveBinding>
          ...
        </MasterSlaveBindingSet>
      </SubSpace>
      ...
    </AddressSpace>
    ...
  </AddressSpaceSet>

```

3.6.2 SubSpace**DESCRIPTION**

This object describes a segment in *AddressSpace*. It has the following objects and/or attributes:

- **MasterSlaveBindingSet** (mandatory): refer to [MasterSlaveBindingSet](#).
- **MemoryConsistencyModel** (optional): refer to [MemoryConsistencyModel](#).
- **name** (mandatory; type *string*): the name of this object.
- **id** (mandatory; type *ID*): the ID of this object.
- **start** (mandatory; type *long*): the start address.
- **end** (mandatory; type *long*): the end address.
- **endian** (optional; type *EndianType*): the endianness of this object.

EXAMPLE

See [AddressSpace](#).

3.6.3 MemoryConsistencyModel**DESCRIPTION**

It has the following objects and/or attributes:

- **rawOrdering** (optional; type *OrderingType*): specifies the memory ordering of read-after-write access.
- **warOrdering** (optional; type *OrderingType*): specifies the memory ordering of write-after-read access.

- **wawOrdering** (optional; type *OrderingType*) specifies the memory ordering of write-after-write access.
- **rarOrdering** (optional; type *OrderingType*): specifies the memory ordering of read-after-read access.

EXAMPLE

See [AddressSpace](#).

3.6.4 MasterSlaveBindingSet**DESCRIPTION**

It has the following objects and/or attributes:

- **MasterSlaveBinding** (mandatory): refer to [MasterSlaveBinding](#).

EXAMPLE

See [AddressSpace](#).

3.6.5 MasterSlaveBinding**DESCRIPTION**

This object binds a *MasterComponent* to a *SlaveComponent*. It has the following objects and/or attributes:

- **Accessor** (mandatory): refer to [Accessor](#).
- **slaveComponentRef** (mandatory; type *IDREF*): specifies the **id** of *SlaveComponent*.

EXAMPLE

See [AddressSpace](#).

3.6.6 Accessor**DESCRIPTION**

It has the following objects and/or attributes:

- **PerformanceSet** (optional): refer to [PerformanceSet](#).
- **masterComponentRef** (mandatory; type *IDREF*) specifies the **id** of *MasterComponent*.

EXAMPLE

See [AddressSpace](#).

3.6.7 PerformanceSet**DESCRIPTION**

This groups one or more [Performance](#). It has the following objects and/or attributes:

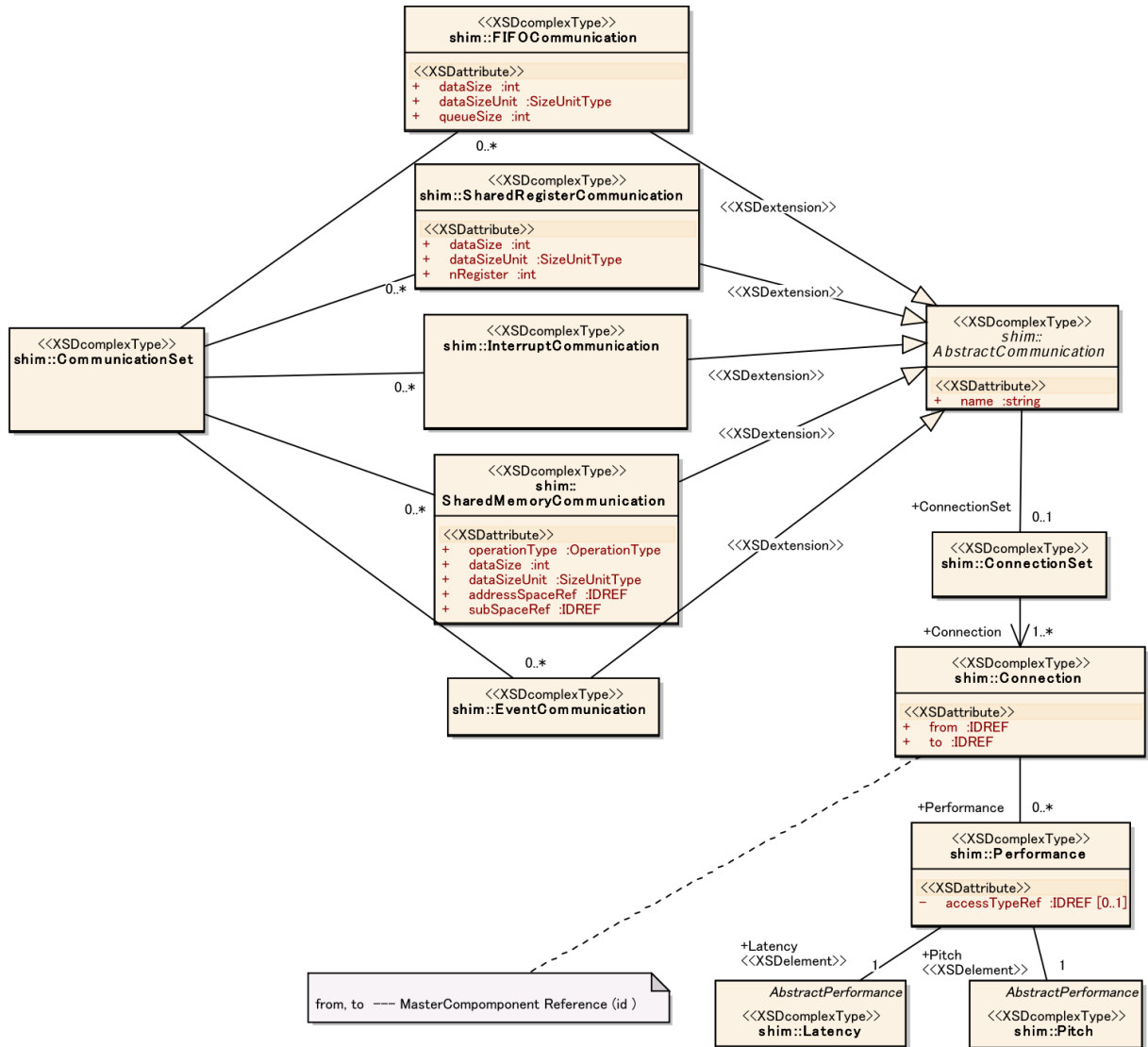
- **Performance** (optional): refer to [Performance](#).

EXAMPLE

See [AddressSpace](#).

3.7 CommunicationSet

SCHEMA



DESCRIPTION

CommunicationSet describes the available *MasterComponent*-to-*MasterComponent* communication. There are six objects that describe different types of communication.

3.7.1 FIFOCommunication

DESCRIPTION

This object describes FIFO-based communication. It has the following objects and/or attributes:

- **ConnectionSet** (mandatory): refer to [ConnectionSet](#).
- **name** (mandatory; type *string*): the name of this object.
- **dataSize** (mandatory; type *int*): the data size of this FIFO.

- **dataSizeUnit** (mandatory; type *SizeUnitType*): the unit of **dataSize** of this FIFO.
- **queueSize** (mandatory; type *int*): the queue size (multiples of **dataSize** in **dataSizeUnit**, so that the total capacity is a product of **dataSize** * **dataSizeUnit** * **queueSize**) of this FIFO.

EXAMPLE

```
<FIFOCommunication dataSize="64" dataSizeUnit="KiB" queueSize="32" name="fifo_00">
  <ConnectionSet>
    <Connection from="SHIMEDITOR25331849408820141005130142974"
      to="SHIMEDITOR31113490118120141005130142974">
      <Performance>
        <Pitch best="10.0" typical="10.0" worst="10.0"/>
        <Latency best="10.0" typical="10.0" worst="10.0"/>
      </Performance>
    </Connection>
    ...
  </ConnectionSet>
</FIFOCommunication>
```

3.7.2 SharedRegisterCommunication**DESCRIPTION**

This object describes a shared-register based communication. It has the following objects and/or attributes:

- **ConnectionSet** (mandatory): refer to [ConnectionSet](#).
- **name** (mandatory; type *string*): the name of this object.
- **dataSize** (mandatory; type *int*): the data size of one shared register.
- **dataSizeUnit** (mandatory; type *SizeUnitType*): the unit of **dataSize** of this shared register.
- **nRegister** (mandatory; type *int*): the number of shared registers.

EXAMPLE

```
<SharedRegisterCommunication dataSize="32" dataSizeUnit="KiB" nRegister="32" name="sreg_00">
  <ConnectionSet>
    <Connection from="SHIMEDITOR25331849408820141005130142974"
      to="SHIMEDITOR31113490118120141005130142974">
      <Performance>
        <Pitch best="10.0" typical="10.0" worst="10.0"/>
        <Latency best="10.0" typical="10.0" worst="10.0"/>
      </Performance>
    </Connection>
    ...
  </ConnectionSet>
</SharedRegisterCommunication>
```

3.7.3 InterruptCommunication**DESCRIPTION**

It has the following objects and/or attributes:

- **ConnectionSet** (mandatory): refer to [ConnectionSet](#).
- **name** (mandatory; type *string*): the name of this object.

EXAMPLE

```

<InterruptCommunication name="Interrupt_00">
  <ConnectionSet>
    <Connection from="SHIMEDITOR25331849408820141005130142974"
      to="SHIMEDITOR31113490118120141005130142974">
      <Performance>
        <Pitch best="10.0" typical="10.0" worst="10.0"/>
        <Latency best="10.0" typical="10.0" worst="10.0"/>
      </Performance>
    </Connection>
  </ConnectionSet>
</InterruptCommunication>

```

3.7.4 SharedMemoryCommunication**DESCRIPTION**

It has the following objects and/or attributes:

- **ConnectionSet** (mandatory): refer to [ConnectionSet](#).
- **name** (mandatory; type *string*): the name of this object.
- **operationType** (optional; type *OperationType*): the type of this shared memory communication.
- **dataSize** (optional; type *int*): the data size of this *SharedMemoryCommunication*.
- **dataSizeUnit** (mandatory; type *SizeUnitType*): the unit of **dataSize** of this shared memory.
- **addressSpaceRef** (optional; type *IDREF*): specifies the **id** of [AddressSpace](#) that the shared memory this object uses. If this attribute is not declared and the subsequent *subSpaceRef* is not declared, then it means the *SharedMemoryCommunication* mechanism is valid for all *AddressSpace* and *SubSpace*.
- **subSpaceRef** (optional; type *IDREF*) specifies the **id** of [SubSpace](#) that this *SharedMemoryCommunication* supports. When this attribute is declared, the corresponding *addressSpaceRef* must also be declared as above. When this attribute is omitted and *addressSpaceRef* is declared, it means the *SharedMemoryCommunication* mechanism is valid for all *SubSpace* for the declared *AddressSpace*. When *addressSpaceRef* is omitted but *subSpaceRef* is declared, the interpretation is undefined and must not be used.

EXAMPLE

```

<SharedMemoryCommunication dataSize="128" dataSizeUnit="KiB"
  addressSpaceRef="SHIMEDITOR22375265206920141005130143354"
  subSpaceRef="SHIMEDITOR9140938132320141005130143355" name="shmem_00">
  <ConnectionSet>
    <Connection from="SHIMEDITOR25331849408820141005130142974"
      to="SHIMEDITOR31113490118120141005130142974">
      <Performance>
        <Pitch best="10.0" typical="10.0" worst="10.0"/>
        <Latency best="10.0" typical="10.0" worst="10.0"/>
      </Performance>
    </Connection>
    ...
  </ConnectionSet>
</SharedMemoryCommunication>

```

3.7.5 EventCommunication**DESCRIPTION**

This object describes an event-based communication. It has the following objects and/or attributes:

- **ConnectionSet** (mandatory;): refer to [ConnectionSet](#).

- **name** (mandatory; type *string*): the name of this object.

EXAMPLE

```
<EventCommunication name="Event_00">
  <ConnectionSet>
    <Connection from="SHIMEDITOR25331849408820141005130142974"
      to="SHIMEDITOR31113490118120141005130142974">
      <Performance>
        <Pitch best="10.0" typical="10.0" worst="10.0"/>
        <Latency best="10.0" typical="10.0" worst="10.0"/>
      </Performance>
    </Connection>
    ...
  </ConnectionSet>
</EventCommunication>
```

3.7.6 ConnectionSet**DESCRIPTION**

It has the following objects and/or attributes:

- **Connection** (mandatory): refer to [Connection](#).

EXAMPLE

See examples in various communication objects.

3.7.7 Connection**DESCRIPTION**

It has the following objects and/or attributes:

- **Performance** (optional): refer to [Performance](#).
- **from** (mandatory; type *IDREF*): the **id** of *MasterComponent* that forms the initiator of connection.
- **to** (mandatory; type *IDREF*): the **id** of *MasterComponent* that forms the terminal of connection.

EXAMPLE

See examples in various communication objects.

4. Use Cases

These use cases are provided as examples to see how to use the information that SHIM XML contains. The categories of tools mentioned are to exemplify and to help the user to understand the concepts. This may also apply to other types of tools.

4.1 Performance Estimation: Auto-Parallelizing Compiler

Table 5. Performance Estimation Use case

Illustrated tool (ID)	Auto-parallelizing compiler (APC1)
Applicability	Any tool that can benefit from knowing the performance characteristics of multi-many-core hardware
SHIM elements illustrated	<i>ClockFrequency</i> , <i>MasterComponent</i> , <i>CommonInstructionSet</i> , Latency, Pitch, Cache, FIFOCommunication
Tool processing overview	The compiler takes C sequential code and outputs parallelized C source code at the thread level. The input code is analyzed first to determine what instructions it would consist of, the collection of memory size it uses, the access type (rwx and access width), the control flow and overall data flow. Based on the flow analysis, the code is split into multiple threads to match the number of cores available, so that each thread consumes approximately the same amount of cycles. The data placement is optimized based on the available cache size per core, the latency of memory access, and the inter-core communication latency.

4.1.1 Using “CommonInstructionSet”

- Each “*MasterComponent*” (such as a processor core) has a “*ClockFrequency*” and “*CommonInstructionSet*”
- “*CommonInstructionSet*” is defined as LLVM IR instructions
- “*CommonInstructionSet*” has performance (processor cycles) of each instruction, expressed in best, typical, and worst cycles
- The clock value and the cycle information can be used to estimate the execution time of specific instructions on the hardware

4.1.2 Using “PerformanceSet”

- For memory operation for data read/write, or load/store instructions, performance values are calculated using Latency and Pitch of a particular *SubSpace* where the data resides
- Latency/Pitch has best/typical/worst cycles. The typical value is normally used, however, if the tool is capable of understanding repetitive memory access, the best value is used
- Based on the memory performance characteristics and the data usage, the compiler selects the best memory to locate the data

4.1.3 Using “Cache”

- First, the tool reads *Cache::cacheCoherency* to determine whether hardware cache coherency is supported. If not supported, a software-based coherency operation is inserted where necessary, while mapping data/threads to cores so that such operations are minimized

- `Cache::blockSize` is the cache line size – this information is used by the tool to optimize the data placement
- `Cache::size` is the cache size used by the tool to judge the optimal work data unit size

4.1.4 Using “FIFOCommunication”

- All *CommunicationSet* elements, including this *FIFOCommunication*, have *ConnectionSet* containing *Connection(s)* describing which pair of *MasterComponents* are connected via this communication feature
- *FIFOCommunication* has *dataSize* and *queueSize* which are used by the tool to determine the unit of data transferred
- All *XXXCommunication* have *Performance*, which contains *Latency* and *Pitch* expressed in cycles. This can be used by the tool to determine the execution cost of transferring data via this communication channel

4.2 Tool Configuration - RTOS Configuration Tool

Table 6. Tool Configuration Use case

Illustrated tool (ID)	A configuration tool for a runtime software such as RTOS or middleware (RTS1)
Applicability	This model can be utilized to generate a runtime software specific configuration file. It is also applicable to other host tools that require configuration.
SHIM elements illustrated	<i>ClockFrequency</i> , <i>SlaveComponent</i> , <i>SubSpace</i> , <i>MasterSlaveBinding</i> , Common Configuration File (CCF)
Tool processing overview	An RTOS has a configurator that generates RTOS configuration C source code, which is later compiled and linked with the RTOS libraries. The configurator has a GUI, which allows a user to select/specify the PU clock to set by RTOS boot code, the memory address and size for the RTOS memory pool.

4.2.1 Using “ClockFrequency”

- Each “*MasterComponent*” (such as a processor core) has a “*ClockFrequency*”
- The value attribute can contain an XPath expression, which points to a separate, Common Configuration File (CCF), another XML file used to express configuration parameters.
- In this scenario, the selectable values are listed in the part of the CCF, and it has “formType” called “select”.
- The configurator reads the CCF and when it reads off the formType, it dynamically displays a combo-box GUI control object with the selectable values listed.
- The text label of the GUI can be obtained from the “name” attribute of the *ClockFrequency*, and the parent text label can be in the *MasterComponent* name it is tied to.
- These names can be used as the base of C #define symbol name in the generated C source code.
- Therefore the tool does not need to know it is configuring the clock frequency, but still serves the purpose.

Note: Other configurable elements can be handled in the same way, or the tool can deliberately look for a specific element and use the value.

4.2.2 Using “SubSpace”

- The configurator allows setting of RTOS memory pool base address and size

- It needs to know what memory is available, its address and size
- The tool first checks all “*SlaveComponents*” in the “*ComponentSet*” and checks the attribute “RWType” being “rw” and record the name of the *SlaveComponent*
- Then it digs in “*SubSpace*” under “*AddressSpace*” and checks “*MasterSlaveBinding*” tied to the *SubSpace*
- *MasterSlaveBinding* contains “*SlaveComponentRef*” attribute and the tool must determine if it matches the name recorded
- Once the matching *SubSpace* is found, the name, start, end (addresses) should display to user the available memory area

4.3 Hardware Modeling

This use case uses the same SHIM classes described in [Performance estimation - auto-parallelizing compiler](#), so the actual steps of accessing the SHIM objects are omitted. Table 7 provides the basic idea.

Table 7. Hardware Modeling Use Case

Illustrated tool (ID)	Quick and simple hardware modeling tool (HM1)
Applicability	This model can be applied to any tool that provides virtual hardware functionality. If such tools can import a SHIM XML, the modeling functionality itself may evolve to offer more sophisticated features
SHIM elements illustrated	<i>ComponentSet</i> , Latency/Pitch, and other components mentioned in Performance estimation - auto-parallelizing compiler
Tool processing overview	The tool can take a starting SHIM XML describing a multicore hardware. A performance analysis tool can take a SHIM XML and a set of software. Using the similar processing described in APC1, it can make a rough ‘static’ performance analysis of the software on the given SHIM XML. The hardware modeling tool can manipulate “ <i>ComponentSet</i> ”, such as adding a processor core with a specific performance or change memory “latency/pitch”. The modified SHIM XML can be served as the input again to the static performance analysis tool to see the change of performance for the new hardware model.

5. SHIM XML Authoring Rules and Guidelines

This section defines rules and guidelines for authoring (creating) a new SHIM XML file. The rules are the ones that are mandated to follow, while guidelines are recommendations. The following sections state either [Rules] or [Guidelines].

The software tools that consume SHIM XML will expect the SHIM XML files to follow the rules (and hopefully the guidelines). One may choose to not to follow the guidelines, but it is the responsibility of the SHIM XML provider to ensure that the expected use case and tools consuming the SHIM XML file do not face any issues by not following particular items in the guidelines.

5.1 File Name [Rule]

The SHIM Editor does not create the SHIM XML file name automatically for you – you must specify the appropriate name. The file name should describe what the specific SHIM XML is about. In most cases, a SHIM XML should describe some hardware board, which may or may not contain multiple chips and memory. The file name should describe the outermost hardware entity – so if it is a hardware board, it should describe the name of the board, or a unique name that characterizes the hardware board. If the SHIM XML is not an actual board, but instead a virtual hardware platform, use the name of the particular virtual platform instance.

In addition to the name of a hardware entity (or virtual platform), it needs to have a version information in the file name. Note that even if you use some RCS, when you export the SHIM XML file out to someone that does not have access to the RCS repository, then add a version information to avoid confusion. The version information can be any alphanumerical strings, as long as it is unique over different versions of that particular SHIM XML file. Examples would be the revision of the file from the RCS, modification date in yyyy-mm-dd format, or your own versioning scheme.

Another element worth mentioning is the compiler used to measure the performance, especially if there is a multiple choice of compilers supported for the processor architectures implemented in the hardware. The file name shall also carry the version and revision information of the compiler. If the hardware described by SHIM contains multiple ISA and if multiple compilers are used, the compilerName and compilerVersion shall denote those of SDK/tool-chain that integrates or packages these multiple tools.

The file name must be unique and the best way to do this is to use the internet domain name, in the manner Java uses to name its packages⁴. Combined with what is described above, the template for the SHIM XML file name is as below:

domainName.hardwarePlatformName.platformVersion.compilerName.compilerVersion.xml

For example, if the ABC evaluation board version 1.0.0 manufactured by XYZ Ltd, whose internet domain name is www.xyz.com and the compiler used to measure the performance is gcc 4.9.0, then the SHIM XML file name should be

`com.xyz.abcEvalBoard.1_0_0.gcc.4_9_0.xml`

If the name of the hardware platform is too vague, it is advisable to extend the platform name part with some other sub-name, like the name of the multi-many-core processor integrated, to make the name more distinguishable.

Also, the hardware platform name shall be extended to contain any other platform-specific information that further sub-categorizes the particular SHIM XML file. This is sometimes favorable in a case where the hardware platform

⁴ <http://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html>

has multiple operation modes, and if you choose to create multiple SHIM XML files to describe this. Another option is to use CCF to use a single SHIM XML file to be configurable, but that will require one to write a CCF and a CCF-compatible SHIM XML. There are pros and cons of doing either, so it is at your discretion which model you would employ. The rule of thumb would be to use CCF if you have a stable SHIM XML file and you are about to create similar but another SHIM XML file, and you know that there will be more, and these SHIM XML files are expected to be reused with multiple minor modification, then it should be more efficient to adapt CCF model.

5.2 Naming of Various Objects [Rule]

All the SHIM objects will have names that must be unique when expressed as an absolute XML path. It is just as in the file systems – different directories can have files with the same names, as long as the directory names are different. This means you can have *MasterComponent* with the same names, as long as the *ComponentSet* names are different.

Consistency is another thing to consider regarding naming of objects. This is nothing special to SHIM, nevertheless it is critical to maintain consistent naming within a SHIM XML. If you are authoring multiple SHIM XML, these should also be consistent. The SHIM specification does not require object names to serve any purpose other than distinguishing one from another. However, in some situations the names can be effective in conveying important information that the SHIM standard itself does not define (this could be included in a future version of SHIM). In the meantime, the consistent naming may serve the gap. Also, consistency is critical when a SHIM XML, or a part of, is reused. When SHIM supports [Componentization of SHIM XML](#), then the consistency should greatly ease adopting the new specification.

5.3 Level of Detail and Precision [Guideline]

In principle, all the hardware properties that can be expressed in SHIM should be described. It is also advisable to match the names of components to the names given in the hardware manual, if such already exists.

Note: omitting a description of any hardware properties does not necessarily lead to the software tools being non-functional. The tools treat a SHIM XML file as-is, and are unable to determine if the description has been omitted, as long as the SHIM XML describes a functional hardware. So it is at the discretion of the SHIM XML author for what to expose or not, for example.

6. Common Configuration File (CCF)

This chapter describes the Common Configuration File (CCF) which provides a powerful and flexible mechanism to describe configurable hardware (and software) elements. CCF allows reuse of the same SHIM XML for different hardware configurations and also provides consistent configuration interface. It can also be used to describe vendor-specific features, such as providing some special operation mode not supported by the SHIM XML schema that, when enabled, changes the performance which is described in the SHIM XML.

Though it is strongly recommended to support CCF, it is optional and a software tool can still use SHIM without supporting CCF. If CCF is not included, its basic capability is fixed to the default configuration written in the SHIM XML. Note a SHIM XML will not reference the CCF in anyway – only vice versa (CCF XML references a SHIM XML file).

6.1 Concept

6.1.1 Multiple Hardware Configuration

A hardware platform often has multiple configurations (e.g., the system or processor clock frequency, a configurable cache size, the size of FIFO, some operation mode). SHIM tries to generalize the hardware model where possible so that we have a single interface for different hardware. However, there are still some generic items that are often configurable, such as clock frequency. If SHIM does not have the capability to express this configurable clock frequency, then one must create separate SHIM XML files differing only in the *ClockFrequency*.

The CCF describes the configurable items in a file called CCF XML (a separate XML file from the SHIM XML). Software tools using SHIM can utilize this mechanism to provide a [Configuration tool user interface](#) within its tool or as a separate standalone tool. When the configuration tool is executed, along with the SHIM XML and CCF, it provides a mechanism to modify the specific parts of SHIM XML, according to the inputs made by the tool user, which can also be automated by the tool. Altogether, this will relieve the SHIM XML authors from writing similar SHIM XML files, differing only in the values of configurable items, while also helping software tool developers to develop configuration user interface.

6.1.2 Vendor-Specific Hardware Features Affecting SHIM Objects

It is not possible to include in a SHIM XML any hardware mechanisms that are not defined in the schema. The use of such features often results in different hardware performance. Since SHIM describes the performance properties in terms of processor, memory, and communication, this inability to describe such mechanisms can lead to inaccurate performance estimations beyond SHIM's targeted 20% error rate. CCF can be used to describe such vendor-specific hardware features and provide software tools the configuration interface for those features. The SHIM XML author can describe in a CCF a way to modify SHIM XML according to the configuration tool user input. The configuration interface is dynamically created from the CCF, so the software tools need not be aware of the vendor-specific features, while allowing the hardware vendors to describe such features. The SHIM XML is modified according to the CCF and the tools then use the resulting SHIM XML.

6.1.3 Configuration Tool User Interface

Often software tools must provide a user interface (UI), whether graphical or not; however, there is usually support for both interfaces. Commercial tools must support a wide variety of platforms so that it can achieve a critical mass of users required to fuel the continuous evolution of their technology and business. This is especially true in the embedded systems market, which has an incredibly wide range of hardware, and also a wide range of COTS software components. Therefore, it is critical to derive ways to effectively and economically support these configuration requirements by the tools. CCF is intended to provide a standard way to achieve this.

The nature of the problem of providing a user interface for all such variations is that the actual configuration items are specific to whatever entity that it configures. There are some common items, but often they differ in the subset and sometimes they are interleaved. If a tool takes an approach of coding the configuration interface for each

specific variable entity, it can be quite costly (e.g., the configuration management cost, distribution of the software, quality maintenance issues).

The key to mitigating such issues is to bind the UI design description with the specific configuration item description, and use the same algorithm and also the code that interprets the combined description, and create the UI dynamically. This approach has already been quite popular and well-proven. The problem is that there is no standard for describing this, and even if two tools use the same configurable entities, each must create similar, but different, descriptions since it is not standardized. CCF is meant to remedy the situation.

CCF defines six types of configuration input interface objects: *select*, *bool*, *text*, *integer*, *hex_integer*, and *expression* (described later). The tool developer must determine what kind of UI controls it maps the configuration input interface objects to, but the *select* is intended as a combo box, *text* is a text field, *integer* is an integer field, and so on. These controls can be grouped in any combination and are also capable of switching a sub-set of configuration items, depending upon an input on the particular configuration item (often residing at a higher level of configuration items). Most of these control objects, or *Form-Type*, as named in CCF, are simple to use, and, when used in combination, can describe most configuration items needed. Sometimes, a configuration item may depend on the values of other multiple configuration items, so it is necessary to express the relationship in some arithmetic or logical way. Finally, *expression* is a special object that is a pseudo-interface object which serves as a bucket to contain various XPath expression string objects. Since XPath allows the tool to describe basic arithmetic operations, it can be used to calculate a value that is dependent on the values of other configuration items. Along with the Form-type supported, including the capability to describe arithmetic operations with XPath, and the capability to group configuration items and describe them hierarchically, CCF provides a simple yet powerful way to cover most or all of the configuration interface and description needs.

6.2 Interface

6.2.1 XML Schema

The CCF class diagram is shown in Figure 9. Common Configuration File (CCF) class diagram.

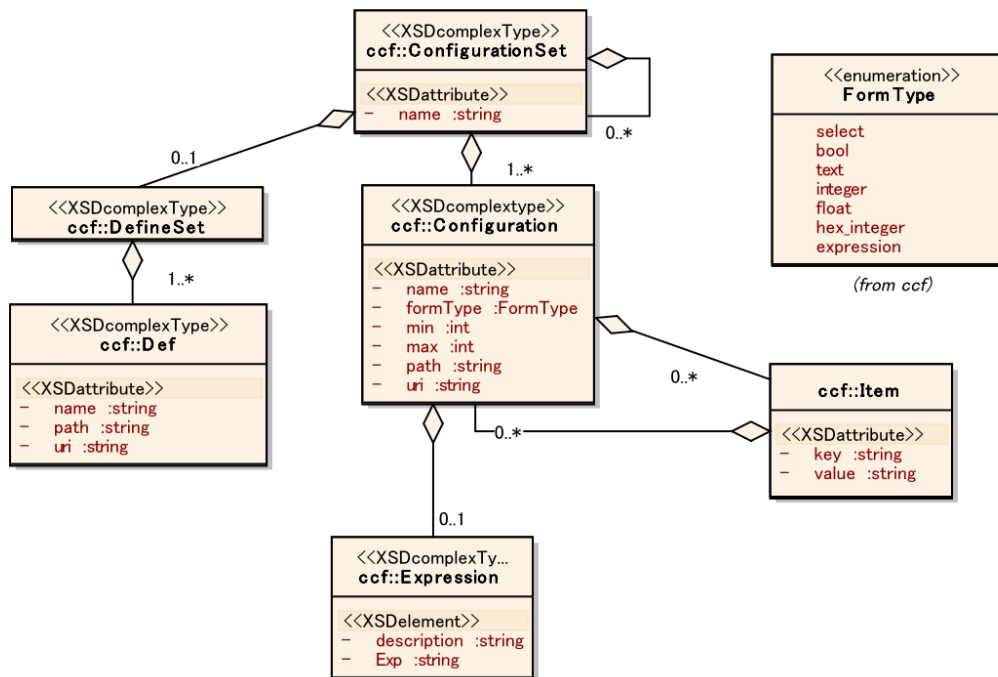


Figure 9. Common Configuration File (CCF) class diagram

```

<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Configuration" type="Configuration"/>
  <xs:complexType name="Configuration">
    <xs:sequence>
      <xs:element name="Item" type="Item" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="Expression" type="Expression" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" use="optional" type="xs:string"/>
    <xs:attribute name="formType" use="required" type="FormType"/>
    <xs:attribute name="min" use="optional" type="xs:int"/>
    <xs:attribute name="max" use="optional" type="xs:int"/>
    <xs:attribute name="path" use="optional" type="xs:string"/>
    <xs:attribute name="uri" use="optional" type="xs:string"/>
  </xs:complexType>
  <xs:element name="Item" type="Item"/>
  <xs:complexType name="Item">
    <xs:sequence>
      <xs:element name="Configuration" type="Configuration" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="key" use="optional" type="xs:string"/>
    <xs:attribute name="value" use="required" type="xs:string"/>
  </xs:complexType>
  <xs:simpleType name="FormType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="select"/>
      <xs:enumeration value="bool"/>
      <xs:enumeration value="text"/>
      <xs:enumeration value="integer"/>
      <xs:enumeration value="float"/>
      <xs:enumeration value="hex_integer"/>
      <xs:enumeration value="expression"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="ConfigurationSet" type="ConfigurationSet"/>
  <xs:complexType name="ConfigurationSet">
    <xs:sequence>
      <xs:element name="Configuration" type="Configuration" minOccurs="1"
maxOccurs="unbounded"/>
      <xs:element name="DefineSet" type="DefineSet" minOccurs="0" maxOccurs="1"/>
      <xs:element name="ConfigurationSet" type="ConfigurationSet" minOccurs="1"
maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="name" use="required" type="xs:string"/>
  </xs:complexType>
  <xs:element name="Expression" type="Expression"/>
  <xs:complexType name="Expression">
    <xs:sequence>
      <xs:element name="description" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="Exp" type="xs:string" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="DefineSet" type="DefineSet"/>
  <xs:complexType name="DefineSet">
    <xs:sequence>
      <xs:element name="Def" type="Def" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Def" type="Def"/>
  <xs:complexType name="Def">
    <xs:sequence/>
    <xs:attribute name="name" use="required" type="xs:string"/>
    <xs:attribute name="path" use="required" type="xs:string"/>
    <xs:attribute name="uri" use="required" type="xs:string"/>
  </xs:complexType>
</xs:schema>

```

6.2.2 Semantics

ConfigurationSet is the topmost object, which includes at least one *Configuration* object that indicates which *FormType* it uses. For *select* *FormType*, multiple *Item* objects are listed that comprise the entries in the combo box control. The key attribute is used as the text to display for the entry, where the value is the actual configuration value. The value itself is often self-explanatory, and the key and value are the same. The name attribute of the parent *Configuration* object can be used as the label for the control. If a *FormType* of *Configuration* object is an *integer*, then the min and max attributes define the minimum and maximum values that users can input, respectively. The *ConfigurationSet* object can nest itself, forming a hierarchical configuration item tree. Also, an *Item* object can have another *Configuration* object underneath, which is useful when *FormType* is *select* and if you need a particular set of configuration only when the user selects a specific *Item*. This hierarchical model can be used to group a particular set of configuration items that tools can use to group configuration UI controls accordingly.

If *FormType* of *Configuration* object is *expression*, then an *Expression* object is defined that has *Exp* attribute, which is literally the XPath expression to use. The XPath allows the CCF to perform basic calculations, taking some values of another XML as parameters. The *ConfigurationSet* object can contain another object called *DefineSet* (this is similar to *#define* in C language). In the XPath expression, one often references the value of the particular configuration item. *Def* object, which hangs onto the *DefineSet*, can be used in the XPath expression in a short text string. The shorter string can be used in the *Exp* attributes and also path attributes of *Configuration* objects that share the same parent *ConfigurationSet* object.

All *Configuration* objects have “path” and “uri” attributes that specify where the result of each *formType* is targeted. The path is an XPath expression and uri is the location of the target XML file. It is the CCF author’s responsibility to match the type of *formType* and the target XPath expression. The configuration tool also uses the path and uri to obtain the default configuration values by reading the current values from the target XML. Therefore, when the configuration tool starts up, the input fields are initialized with values read from the target XML, specified by path and uri.

The expression *formType* usually takes one or more values from some XML file (usually SHIM XML). These values, however, may also be modified by other *Configuration* objects in the same CCF. Therefore, it is important how *Configuration* objects are processed. The CCF must be processed top-down, and CCF must be authored assuming this order of processing.

6.2.3 FormType

DESCRIPTION

This is enumeration (constants) of CCF form type.

- **select** is of the combo box form type.
- **bool** is of checkbox form type.
- **text** is of text field form type.
- **integer** is of integer (decimal) form type.
- **float** is of float (floating decimal) form type.
- **hex_integer** is of integer (hex) form type.
- **expression** is of [Expression](#) form type. See [Expression](#).

6.2.4 ConfigurationSet

DESCRIPTION

It has the following objects and/or attributes:

- **ConfigurationSet** (optional).
- **Configuration** (mandatory): refer to [Configuration](#).
- **name** (mandatory; type *string*): the name of this object.

6.2.5 Configuration

DESCRIPTION

It has the following objects and/or attributes:

- **Item** (optional): refer to [Item](#).
- **Expression** (optional): refer to [Expression](#).
- **name** (mandatory; type *string*): the name of this object.
- **formType** (mandatory; type *FormType*) specifies the type of form this configuration object use.
- **min** (optional; type *int*): the minimum value of this configuration, when **formType** is integer or hex_integer.
- **max** (optional; type *int*): the maximum value of this configuration, when **formType** is integer or hex_integer.
- **path** (optional; type *string*): the XPath expression describing the destination of resulting configuration according to **formType**.
- **uri** (optional; type *string*): the XML file that **path** is applied.

6.2.6 Item

DESCRIPTION

It has the following objects and/or attributes:

- **Configuration** (optional): refer to [Configuration](#).
- **key** (mandatory; type *string*): the name of this configuration item.
- **value** (mandatory; type *string*): the value of this configuration item.

6.2.7 Expression

DESCRIPTION

It has the following objects and/or attributes:

- **description** (mandatory; type *string*): the description of this expression.
- **Exp** (mandatory): refer to [Exp](#).

DESCRIPTION

It has the following objects and/or attributes:

- **Def** (mandatory;): refer to [Def](#).

6.2.8 Def

DESCRIPTION

It has the following objects and/or attributes:

- **name** (mandatory; type *string*): the name of this object.
- **path** (mandatory; type *string*): the XPath expression that maps to **name**.
- **uri** (optional; type *string*): the XML file that **path** is applied.

6.3 Examples

6.3.1 Generic

```
<?xml version="1.0" encoding="UTF-8"?>
<ConfigurationSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="CCF Sample for SHIM" xsi:noNamespaceSchemaLocation="ccf-schema.xsd">
  <DefineSet>
    <Def name="@sclock" path="//SystemConfiguration/ClockFrequency/@clockValue"
      uri="shim_sample_data.xml"/>
    <Def name="@cashSize" path="//Cache[@name='UnifiedCache_0_0_0']/@size"
      uri="shim_sample_data.xml"/>
  </DefineSet>
  <Configuration formType="select" name="System clockValue-Select"
    path="//SystemConfiguration/ClockFrequency/@clockValue" uri="shim_sample_data.xml">
    <Item key="value" value="20.0"/>
    <Item key="value" value="40.0"/>
    <Item key="value" value="100.0"/>
  </Configuration>
  <Configuration formType="expression" name="Sample Expression"
    path="//MasterComponent/ClockFrequency/@clockValue" uri="shim_sample_data.xml">
    <Expression>
      <description>description</description>
      <Exp>@sclock * 2</Exp>
    </Expression>
  </Configuration>
  <Configuration formType="text" name="Arch" path="//MasterComponent/@arch"
    uri="shim_sample_data.xml"/>
  <Configuration formType="integer" name="nRegister"
    path="//SharedRegisterCommunication/@nRegister" uri="shim_sample_data.xml"/>
  <Configuration formType="float" name="ClockFrequency:clockValue"
    path="//SystemConfiguration/ClockFrequency/@clockValue" uri="shim_sample_data.xml"/>
  <Configuration formType="bool" name="BooleValue Sample"/>
</ConfigurationSet>
```

6.3.2 Nested configuration

This example shows a nested configuration. Based on the selection of the system clock frequency, different choices for processor clock frequency (*MasterComponent*) are displayed and configured.

```

<?xml version="1.0" encoding="UTF-8"?>
<ConfigurationSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="CCF Sample for
  SHIM" xsi:noNamespaceSchemaLocation="ccf-schema.xsd">
  <DefineSet>
    <Def name="@sclock" path="/SystemConfiguration/ClockFrequency/@clockValue"
      uri="datas/shim_sample_data.xml"/>
    <Def name="@cashSize" path="//Cache[@name='UnifiedCache_0_0_0']/@size"
      uri="datas/shim_sample_data.xml"/>
  </DefineSet>
  <Configuration formType="select" name="System clockValue-Select"
    path="/SystemConfiguration/ClockFrequency/@clockValue" uri="datas/shim_sample_data.xml">
    <Item key="value" value="20.0">
      <Configuration formType="select" name="Processor clockValue-Select"
        path="//MasterComponent/ClockFrequency/@clockValue" uri="datas/shim_sample_data.xml">
        <Item key="value" value="20.0"/>
        <Item key="value" value="40.0"/>
        <Item key="value" value="60.0"/>
      </Configuration>
    </Item>
    <Item key="value" value="40.0">
      <Configuration formType="select" name="Processor clockValue-Select"
        path="//MasterComponent/ClockFrequency/@clockValue" uri="datas/shim_sample_data.xml">
        <Item key="value" value="40.0"/>
        <Item key="value" value="80.0"/>
        <Item key="value" value="100.0"/>
      </Configuration>
    </Item>
    <Item key="value" value="100.0">
      <Configuration formType="select" name="Processor clockValue-Select"
        path="//MasterComponent/ClockFrequency/@clockValue" uri="datas/shim_sample_data.xml">
        <Item key="value" value="100.0"/>
        <Item key="value" value="200.0"/>
        <Item key="value" value="300.0"/>
      </Configuration>
    </Item>
  </Configuration>
</ConfigurationSet>

```


7. FAQ

Q: Why is the SHIM working group using an XML schema to describe the multicore and many-core architectures and devices?

A: We have selected to use an XML schema because you can use the technology called XML data binding. It allows you to generate a class library for handling the SHIM XML data as data objects, not as XML elements and attributes. For example, you can create a C++ or Java object called *MasterComponent* from a SHIM XML and access the attributes of the *MasterComponent* element just like you would reference/retrieve a member variable of the C++/Java object. There are many popular open source implementations of XML data binding tools. Without the data binding technology, you can still access the SHIM XML via legacy XML libraries of SAX/DOM. Essentially, you read the XML as a file and iterate over each XML element and attribute, however, this is quite tedious programming and your code becomes dependent on the given XML structure and will not be portable should it change. With the XML data binding, when we update the SHIM spec, there is a high probability that the legacy tools code will still operate as is. *Also refer to [SHIM Concepts](#), [XML](#).*

Q: What is the difference between SHIM and IP-XACT?

A: IP-XACT is basically a 'design' language, primarily focusing on a description of how hardware IP components are electronically tied together. On the other hand, SHIM is a 'descriptive' language, primarily focusing on only the hardware property descriptions that matter to the software development tools. Hence, SHIM does not describe the type of interconnect or bus in any direct way. However, it does describe the master/slave IP components and slave components in a hierarchical manner, but there are no specifics regarding how these are connected together (e.g., whether it is a traditional bus, a cross-bar, or NoC). In SHIM, the IP components are listed mostly for describing memory access properties such as latency, any master-to-master communication like FIFO register, and also for basic processor properties such as clock, instruction set (ABI), cache size and type - which all matters to software tools to estimate the configuration. *Also refer to possible alignment with IP-XACT in [Componentization of SHIM XML](#).*

Q: How does the OpenMPI HWloc compare to SHIM?

A: HWloc⁵ is a little similar to SHIM where it deals with the static chip IP organization. However, there are some major differences. One of the major differences seems to be that HWloc depends on information provided by the OS through its interfaces at runtime, and providing that information through the standard API defined by hwloc. SHIM is intended to be used primarily without running the system - its information is used to construct the OS configuration, by which itself is used to create the information hwloc obtains through the OS interfaces. So it does not focus on the standard description of hardware from a software perspective, but standardizing the run-time API for retrieving the hw topology. Unlike SHIM, HWloc doesn't appear to handle hardware performance metrics information. The hwloc seems to focus on the hw topology so that the application using the hwloc library can use the provided information to bind a thread/process to a particular core, for example. This is indeed one possible use case of SHIM.xml but instead we are focusing on tool use cases, such as performance estimation, tool configuration, and hardware modeling. The hwloc seems to have the ability to describe a virtual hardware by using commands or texts, but the capability seems limited. *Having said this, SHIM specification is defined by its XML schema, and through a schema compiler, it can generate C/C++ libraries also. With the help of a host of tools, it should not be difficult to provide a compact implementation of such library without requiring the XML parser and file system to store the SHIM XML file, enabling use of SHIM from the target runtime system. Aligning with hwloc, without mentioning how, is certainly a possibility, too.*

⁵ <http://www.open-mpi.org/projects/hwloc/>

8. Appendix A: Acknowledgements

The SHIM working group would like to acknowledge the significant contributions of the following people in the creation of this specification:

Working Group

Fumio Arakawa, Nagoya University

Sven Brehmer, PolyCore Software

Masato Edahiro, Nagoya University

Hiroshi Fujimoto, Nagoya University

Masaki Gondo, eSOL (chair)

Masamichi Izumda, TOPS Systems

Hiroyuki Kondo, Renesas Electronics

Markus Levy, Multicore Association (president)

Yukoh Matsumoto, TOPS Systems

Hitoshi Suzuki, Renesas Electronics

The SHIM working group also would like to thank the external reviewers who provided input and helped us to improve the specification. Below is a partial list of the external reviewers (others preferred to remain anonymous).

External Reviewers

Sunita Chandrasekaran, University of Houston

Paul Chen, Wind River

Dr. Satyadhyam Chickerur, B V Bhoomaraddi College of Engineering and Technology

Dr. Michael Deubzer, Timing-Architects

Badrinath Dorairajan, Microchip Technology

Jos van Eijndhoven, Vector Fabrics

Erik Fischer, Augment!IT

Dr.-Ing. Jens Gladigau, Robert Bosch GmbH

Christian Helm, Timing-Architects

Mark Honman, Sundance Multiprocessor Technology

Razvan Ionescu, Freescale

Andrei Kovalev, Freescale

Francois Legal, Assystem

Kenn Luecke, Boeing

Maxine Pelcat, INSA - Département EII