THE Multicore ASSOCIATION

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

Document ID: SHIM Specification

Document Version: 1.00

Status: Final

1

The Multicore Association, Inc.

PO Box 4854

El Dorado Hills, CA 95762

530-672-9113

www.multicore-association.org

Table of Contents

133

Preface

SHIM はソフトウェアツールのためのハードウェア記述を目的としている。したがって、この仕様書は、SHIM を使用してハードウェア記述情報を交換するツール開発者とハードウェア開発者を主な対象としている。また、ソフトウェア開発者が SHIM の意図と範囲を理解することを目的とし、SHIM におけるハードウェア情報記述方法を提供することも意図している。

この仕様書は SHIM の導入から始まり、背景、全体のコンセプト、モデルを説明する。そして SHIM のコンセプト(目的、スコープ、デザイン、インターフェース、制限)を詳しく説明する章が続き、そこでは SHIM がこのドキュメント内で仕様化されている基本的な考えを提供し、将来の仕様拡張のための基本的な考え方を説明しようと試みる。インターフェースが記述されている章では、SHIM XML スキーマと、XML データバインディング技術によってスキーマより直接得られる API を説明する。詳細なユースケースを幾つか紹介している章は、読み手に、SHIM がどのように活用されているのかの識見を与える。最後に、この仕様書は更に詳細な情報が記載された様々な付録で締めくられている。

Definitions

全ての新しい用語は最初に現れた個所、本文中もしくは注釈、において定義される。

1. Introduction

1.1 Overview

マルチコアプロセッサは標準となってきており、数 10 コアや 100 コア以上のプロセッサが登場している。それらのマルチコアは、コア数だけでなく内部接続、クラスタ構造、メモリシステム(階層構造やキャッシュのコヒーレンシも含む)などにおいて様々な形態がある。コア数の増加傾向は自然、かつプロセッサ設計の考え方から不可避であるが、ソフトウェア開発者にとって、多種多様なハードウェアに移植することは重大な課題である。異なるハードウェア向けに既存もしくは新しく作られたソフトウェアを再利用することは大きな負担である。このことは、アーキテクチャ仕様の深い理解が必要なマルチコアプロセッサにおいて期待できる性能を達成しようとする間、この問題は発生し続ける。自動並列化コンパイラ、並列化ツール、OS/ミドルウェアのコンフィグレータ、性能解析ツール、設計支援、実装、ソフトウェア解析などのような様々なツールがある。しかしながら、この負担がツール開発者に転嫁されている間、これらのツールは複雑なマルチコアプロセッサを理解しなければいけない。それ故、様々なツールにより、新しいマルチコアハードウェアのサポートコストを下げることが重要であるが、この問題に対する学術界や産業界での解決努力は十分とは言えない。それによって、マルチコア用ツールのエコシステムを妨げている。

SHIM(Software-Hardware Interface for Multi-many-core)は、マルチコアのハードウェアとソフトウェアツール間のインターフェースを標準化するための産業界と学術界の成果の継ぎ目(joint)である。結果として、標準のインターフェースを用いて新しいマルチコアハードウェアへのサポートコストの低下を目標としている。マルチコア技術の豊富なエコシステムができた時に、SHIM はシステム開発者、半導体ベンダー、ツールベンダーに役立つと考えられるため、SHIM を新しい革新的なマルチコアツールの開発に推奨している。

1.2 Interface

SHIM は XML スキーマにより定義される。マルチコアハードウェアの実装は様々なツールに使用可能な SHIM XML ファイルとして表現される（図 1）。

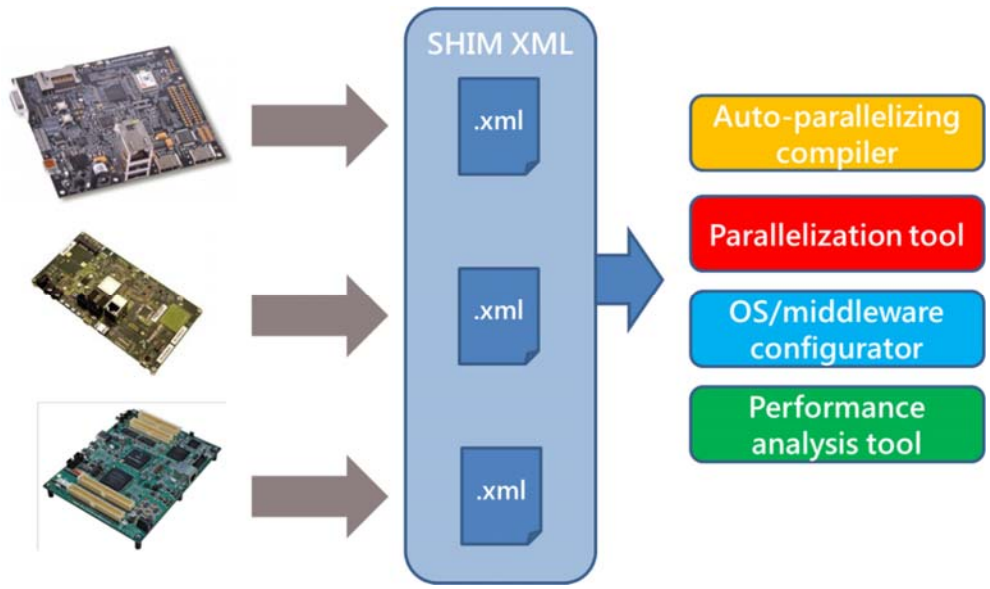
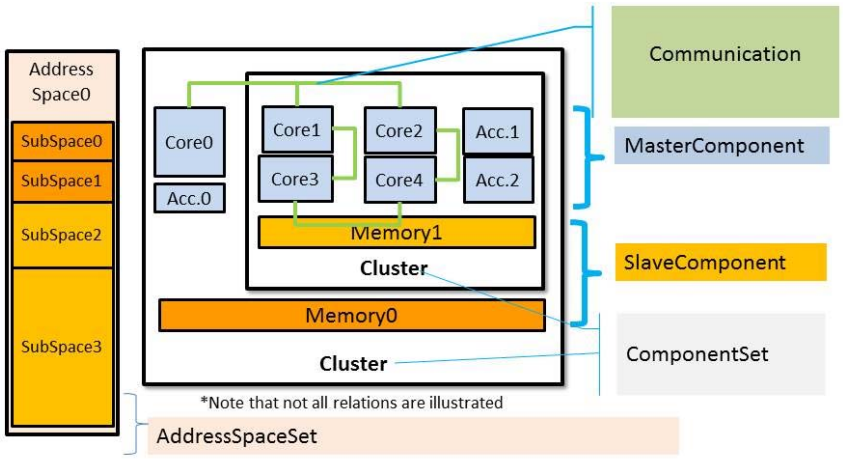


Figure 1. SHIM provides the interface between the hardware and the software tools

SHIM XML ファイルは、「ComponentSet」、「AddressSpaceSet」、「CommunicationSet」と名付けられたそれぞれ子の要素を含む 3 つのトップレベルコンポーネントを用いたツリー構造（図 4 に SHIM XML の例）を持つ。ComponentSet は MasterComponent（プロセッサやアクセラレータなど）と SlaveComponent（メモリブロックやメモリのサブシステムなど）が含まれる。AddressSpaceSet は SubSpace を含んだ 1 つ以上の AddressSpace を含む。最後に、CommunicationSet は MasterComponent のペアをつないだ接続が記述された、幾つかの Communication(通信)の要素を含む。



2

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

ComponentSet は入れ子構造を使用することができる。例えば、複数のハードウェアクラスタ（それぞれのクラスタに複数のコアとクラスタローカルメモリ）を持つチップを表現する時に使用できる。同様に 1 つ以上のマルチコアチップを含むボードにも利用可能である。複数のボードからなり、相互に PCI Express を経由して通信するシステムを記述するときも ComponentSet は使うことができる。この様に ComponentSet の木構造を用いることにより、マルチコアシステムのトポロジを記述できる。このトポロジのアーキテクチャ情報は、コア数、メモリの位置、他のクラスタを構成するコアの特性を確認することができるため、ソフトウェアツールにとって重要である。.

SHIM はソフトウェアツールを対象としているため、コアが異なるメモリにアクセスする方法は勿論、ソフトウェア視点での接続とコア（アクセラレータを含む)間の通信メカニズムの理解が必須である。通信メカニズムは様々な通信のクラスを含む CommunicationSet として記述される。定義されているクラスの簡単な例が InterruptCommunication(割込み通信)であり、それは MasterComponent のペアを結びつけた 1 つ以上の「通信」クラスを含む。メモリアクセスのために、AddressSpace に含まれた SubSpace は開始番地、サイズ、1 つ以上の MasterSlaveBinding(MasterComponent と SlaveComponent の参照関係、どのコア・アクセラレータがアドレスの領域を通じてどのメモリにアクセスできるかの記述)を含む。

これまでに説明したハードウェアのアーキテクチャ情報により、ハードウェアのトポロジやコアとメモリの接続方法を、ツールに理解させることが可能である。しかしながら、ツールは、アプリケーションソフトウェアを「ただ動作させる」だけでは十分ではなく、性能情報も示す必要がある。したがって多くのツールにとって、ハードウェアのアーキテクチャ情報のみでは不十分である。ツールは粗い性能の「見積もり」をし、アプリケーションとマルチコアハードウェアから期待できる性能を、システム設計者とソフトウェア開発者に伝える必要がある。それ故に SHIM にはハードウェアのトポロジの情報に加えて、様々なコア間通信(CommunicationSet)のプロセッサ実行サイクルと異なったコアやアクセラレータによるメモリアクセスサイクルから関連付けられる性能特性が記述される。通信性能は、プロセッササイクルで表現されるレイテンシやピッチを含む Performance の要素として記述される。Performance 要素は、MasterComponent のペアごとに存在する CommunicationSet に記述される。メモリアクセス性能に対しては、各 MasterComponent に定義された、各 SubSpace の各 MasterSlaveBinding と、各 AccessType に対し、個別の Performance 要素が定義される。そのため、異なるアクセスタイプ(例：読み書き、word アクセスや double word アクセス)ごとに異なる Performance 要素が定義される。起こり得る性能分散に対応するために、サイクルは「best(最良)」,「worst(最悪)」、「typical(典型)」の三つ組の形で記述される。これらの値を有効に活用するため、例えば連続するメモリアクセスが発行されるならば最良サイクルに落としこむなど、ツールは、アプリケーションコード解析などにより、十分な賢さを持つ必要がある。なお、ここで言うサイクルはプロセッササイクルであり、MasterComponent と SystemConfiguration の ClockFrequency が一致しない場合には、MasterComponent の値が優先される。

1.3 SHIM Editor

SHIM XML のスキーマは比較的単純であるが、SHIM XML のスキーマ (図 3) 全体の UML のクラス図を見てみると分かる通り、多くの場合メモリアクセスの全てのタイプのための全ての Performance の要素記述の影響により、結果として SHIM XML は非常に規模が大きくなることがある。これを手動で書くと長々しくエラーが多くなるため、我々は SHIM XML ファイルの作成を促進する目的で SHIM Editor と呼ばれるエディタツールを開発している。この生成された SHIM XML ファイルを図 4 で示し、SHIM Editor のプロトタイプのメインウィンドウを図 5 で示す。

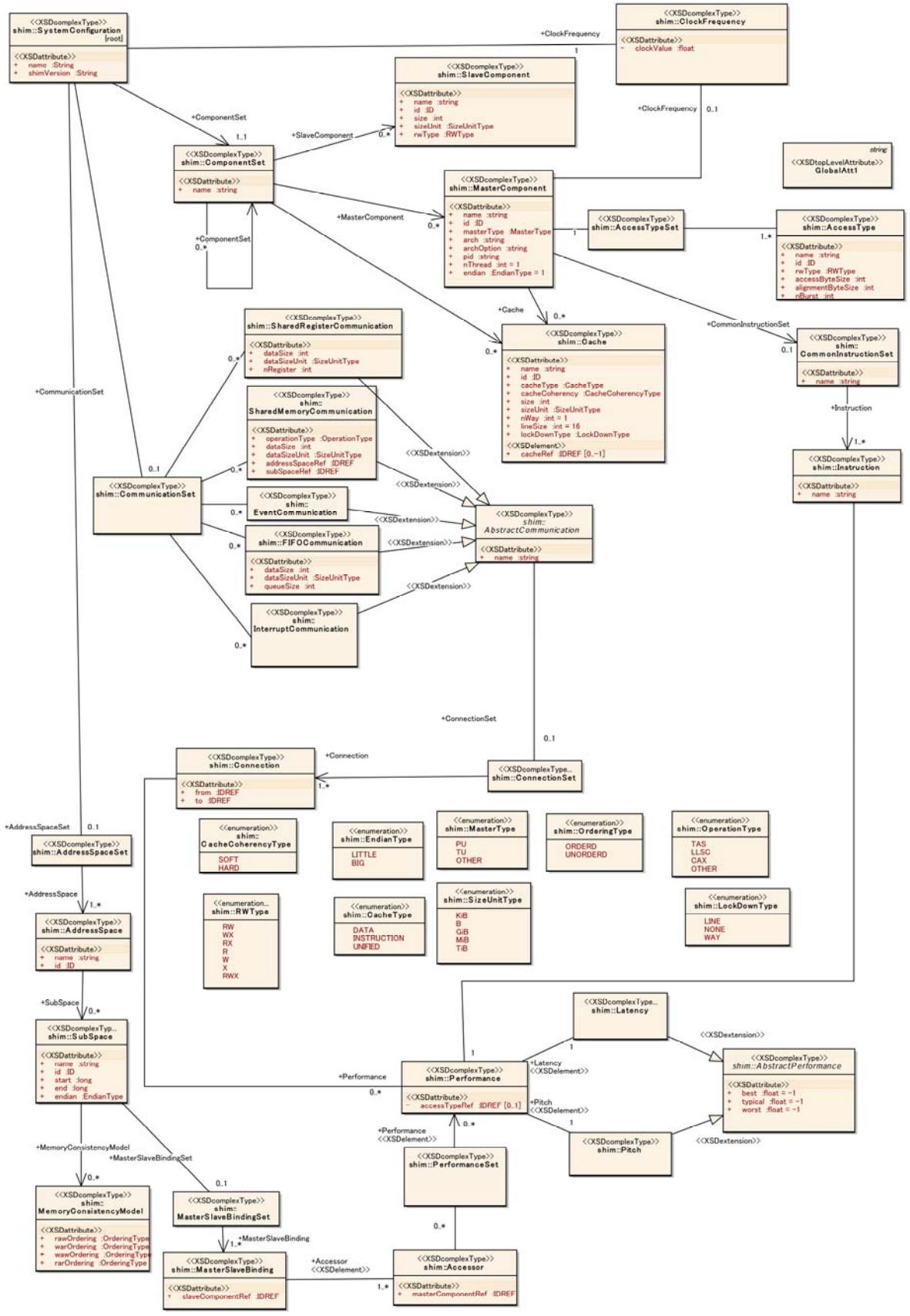
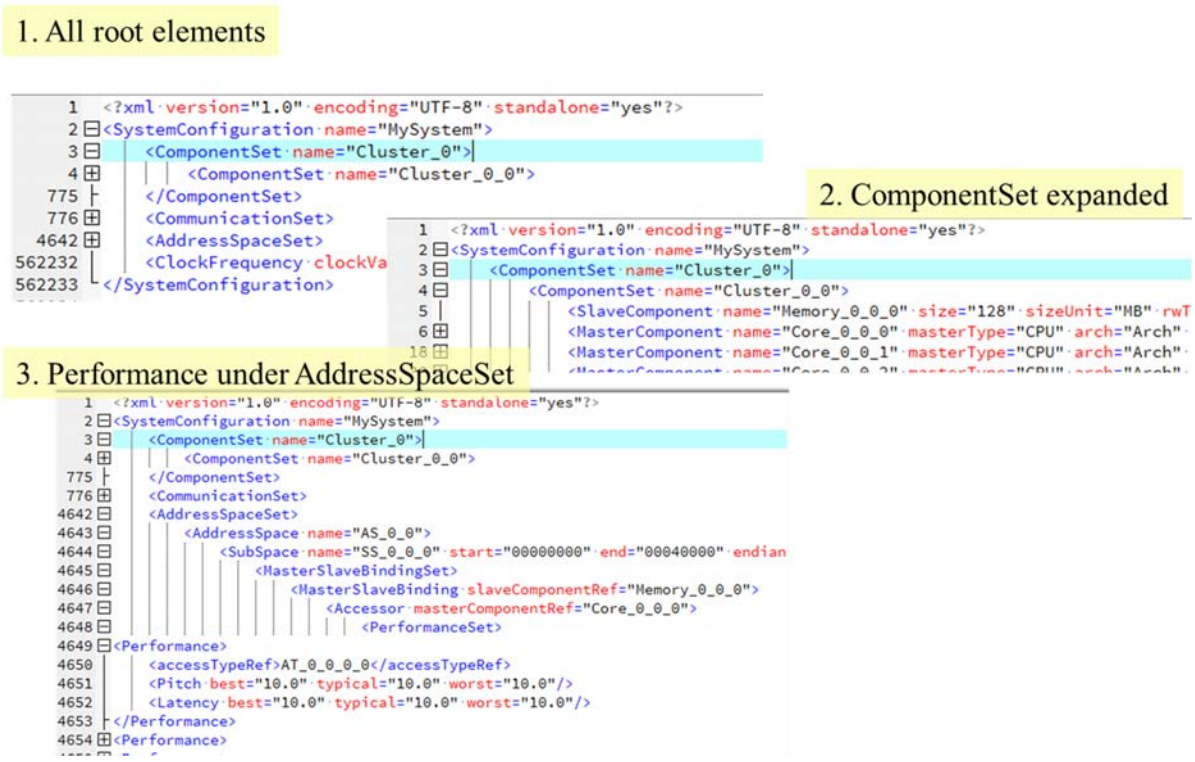


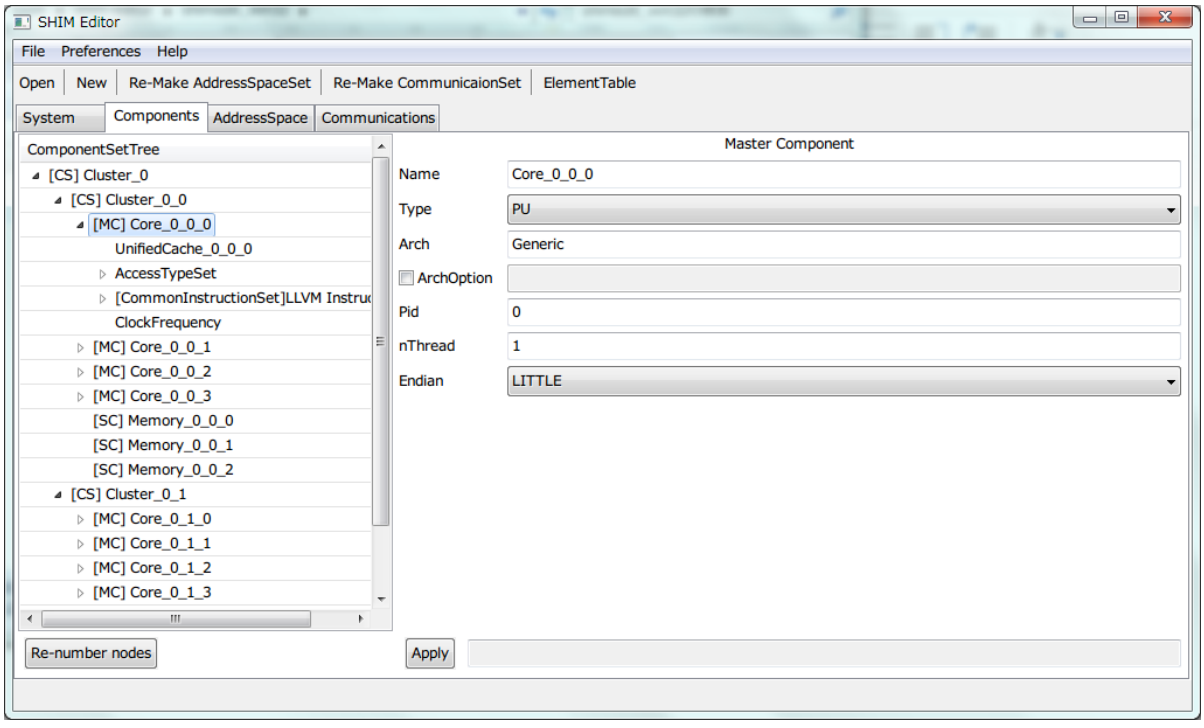
Figure 3. Class diagram representation of the whole SHIM XML schema



215

216 **Figure 4. SHIM XML file example**

217



218

219 **Figure 5. SHIM Editor main window**

2. SHIM Concepts

この章では SHIM の主たる概念を述べる。この文書に説明する方法で SHIM を仕様化する理由を示すため、基本的な考え方を提供する、また、将来における定義の拡張方法を表すことを試みる。この章は SHIM のインタフェースを理解するための基礎を提供するため、インタフェースの詳細に入る前に、この章を徹底的に読むことを強く推奨する。

2.1 Topology - ComponentSet

単純なハードウェアの構成は、単一のプロセッサコアと単一のメモリから成るだろう。しかしながら、マルチコア/メニーコアのハードウェアは、複数のプロセッサコアと多種のメモリデバイスを、様々な配置で持っている。プロセッサやメモリの組み合わせや配置は、マルチコア/メニーコアのハードウェアの特性となり、それはソフトウェアツールがマルチコア/メニーコアのハードウェアを理解するために必要不可欠なものである。

SHIM はプロセッサとメモリデバイスの特定の集まりを「トポロジ」として表現する。電気回路の専門用語では、トポロジとは次のことを指す。「回路要素の相互接続の回路網として形をなすものである。要素が異なる具体的な抵抗値や定格値を持っていたとしても、同じ接続は同じトポロジとして扱われる。トポロジは回路での物理的な配置や、回路図における位置は考慮に入れない。要素間に何の接続があるのかということのみを考える。同じトポロジに相当する物理的な配置や回路図が大量に存在することもある。[1]」 SHIM の視点からすると、トポロジはそれ以上に拡張される。電気回路の専門用語における要素に当たる、プロセッサコアやメモリデバイスに加えて、我々はプロセッサコアとメモリデバイスの特定の集まりを指す「クラスタ」を導入する。通常クラスタと他のハードウェア要素の間には電気的接続があるけれども、SHIM は必ずしも実際の電気接続を扱わないため、クラスタがいかなる接続も持たないかも知れない。とは言え、ソフトウェアツールにとっては、性能の違いを表すように、プロセッサコアとメモリデバイスがどのようにまとめられるのかを見ることは不可欠であり、したがって、SHIM はトポロジカルな表現の一部としてクラスタを導入している。

クラスタは他の（内部の）クラスタ、プロセッサコア、メモリデバイスのあらゆる組み合わせからなる。SHIM はこれらのものを分類し、名前をつける独自の方法がある（表１）。プロセッサコアは *MasterComponent* オブジェクトとして表される。表を見れば分かる通り、*MasterComponent* はある種のアクセラレータ（例：DMA コントローラ）でもありうる。*MasterComponent* の目的は、伝統的なマスタ-スレーブ機構におけるマスタ要素の役割を果たす電気的な要素を表現することである。

Table 1. SHIM representation of hardware components

SHIM の用語	ハードウェアの用語
ComponentSet	あらゆるレベルでのクラスタ（ハードウェアボードそのものもクラスタである）
MasterComponent	プロセッサコア、アクセラレータ、その他のマスタデバイス
SlaveComponent	メモリ

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

245 クラスタ或いは *ComponentSet* は、プロセッサコアクラスタのみならず、ハードウェアボードを表現するためにも用いることが
246 可能である。その延長として、複数のボードからなるシステムを表現することも可能である。この場合、最も外側のクラスタ
247 がシステム境界そのものである。

248 2.2 Memory - AddressSpaceSet

249 ソフトウェアプログラムは、アドレス空間と呼ばれる論理的な窓を通してメモリにアクセスする。プロセッサハードウェアは一般
250 的に、複数のアドレス空間（例えば、異なるアクセス権限に対して）をサポートしている。アドレス空間は、さらに複数の
251 サブ空間或いはアドレスブロックに分けることが可能である。プログラムがメモリデバイスにアクセスを起こすとき、メモリアクセ
252 スは、ソースアドレス或いは宛先アドレスがサブ空間のどれか 1 つに落ちるロード或いはストア命令を発行することによって
253 行われる。このメモリ機構を調整するために、SHIM は *AddressSpaceSet* と呼ばれるグループを有する。一つの
254 *AddressSpaceSet* は複数の *AddressSpace* を含むことが可能であり、各 *AddressSpace* は複数の *SubSpace* を含むこと
255 が可能である。

256 *Subspace* は物理的なメモリデバイス、すなわち *SlaveComponent*、に割り当てられ、クラスタ、すなわち *ComponentSet* の
257 中に存在する。どの *SlaveComponent* が特定の *SubSpace* に割り当てられるかの結合を表現するために、SHIM 仕様は
258 *MasterSlaveBinding* と呼ばれるオブジェクトを用いる。これは、メモリデバイスとメモリサブ空間の間の割り当てを表現する。
259 これは、どの *MasterComponent*（例：プロセッサコア）がメモリアクセスを有するかということも示す。複数の
260 *MasterComponent* がメモリの *SubSpace* へのアクセスを有することもあり得るため、複数の *MasterSlaveBinding* オブジェ
261 クトのまとめる *MasterSlaveBindingSet* というオブジェクトも定義されている。

262 *AddressSpaceSet* 下のオブジェクトを探索することによって、ツールはどのメモリ空間が利用可能で、どのプロセッサコア或い
263 はアクセラレータがそれらに対してどのような種類のアクセスを有するのかを見つけることが可能である。複数の
264 *AddressSpace* や *SubSpace* は同じ *SlaveComponent* を共有するかも知れない。もし、共有が物理メモリの一部でしか
265 起こらないのであれば、それは複数の *SlaveComponent* に分けられる。

266 2.3 Inter-core communication – CommunicationSet

267 ソフトウェアが複数のプロセッサコアやアクセラレータ上で連携して動作するために、しばしば共有メモリ領域を経由して利
268 用可能なデータを交換する。ソフトウェアはまた、トリガー、共有、或いは排他制御を何らかの方法で行わなければならな
269 い。共有メモリが利用可能ではない場合、何らかの形でコアからコア、すなわち *MasterComponent* から
270 *MasterComponent* への通信が必要となる。この状況を調整するため、SHIM は *CommunicationSet* と呼ばれるオブジェ
271 クトを定義している。ConnectionSet クラスの多様性は似通った通信方式を有する（表２）。

272 **Table 2. Inter-core communication classes**

CommunicationSet クラス	説明
SharedRegisterCommunication	共有レジスタに基づく通信。このようなハードウェアはしばしば複数のプロセッサコアからアクセス可能な一組のレジスタを提供する。
SharedMemoryCommunication	共有メモリに基づく通信。命令タイプが TAS（Test and set）、LLSC（Load-link/Store conditional）、CAX（Compare and exchange）、OTHER（定義されないその他の命令）から指定される。

EventCommunication	イベントは、しばしばレジスタビットマップに基づく通信として定義される。一つのプロセッサコアがイベント（論理値）を発生させると、他のコアに送信され、イベントレジスタ内に通知され、割り当てられたイベントとして見ることができる。割り込みを発生する場合もしない場合もある。
FIFOCommunication	FIFO は時々コア間通信に用いられ、FIFO レジスタとして実装される。それには深さを変化させるバッファを伴うかもしれない。
InterruptCommunication	これは典型的なコア間割り込みである。このオブジェクトは名前と *ConnectionSet* のみを持つ。

各クラスは固有のプロパティまたは属性を有する。すべてのクラスは、どの組のコアが特定の通信オブジェクトによって接続されているのかを表現する接続情報を含む。複数の接続が存在しうるので、オブジェクトは、順番に *Connection* をいくつも持つことが可能な *ConnectionSet* を含む。各 *Connection* は *MasterComponent* の組への参照を含む。

ソフトウェアツールはこの情報を用い、SHIM XML によって表される特定のハードウェア実装においてサポートされている *MasterComponent* から *MasterComponent* への通信方式の種類を取得することが可能である。接続が複数の ComponentSet の境界を、例えチップやハードウェアの境界をまたがろうと、超える可能性があることに注意すること。

2.4 Performance

2.4.1 General

予想されるように、異なるプロセッサハードウェアは異なった性能特性を持つ。性能特性はマルチコア/メニーコアのハードウェアでは非常に複雑になりえ、それがソフトウェア設計にとてつもない影響をもたらす。SHIM の原則はアーキテクチャの設計レベルでソフトウェアに影響する特性を捉えることであるため、そのような性能特性を含むことは本質的である（表３）。

Table 3. Performance properties in SHIM

性能特性	関連する SHIM オブジェクト	説明
命令実行	*CommonInstructionSet, Instruction*	プロセッサ命令の実行サイクル数。命令セットは LLVM IR で記述され、特定のプロセッサアーキテクチャのサイクル数は、それらの LLVM IR 命令に関して表現される。
メモリアクセス	*SubSpace, MasterSlaveBinding, Accessor, AccessType*	メモリアクセスに要するプロセッササイクル数。各プロセッサコアは、読み込みや書き込み、或いはバイト、ワード、ダブルワードによるアクセス等、異なるアクセスの種類に対して異なったサイクル数を持つ。
コア間通信	*CommunicationSet* クラス	*InterruptCommunication* など、特定の *Communication* クラスに関する特定の 2 つの *MasterComponent* の接続に要するプロセッササイクル数。

異なるプロセッサ、メモリ、相互接続アーキテクチャの間で、重大な性能の変動があり得るため、すべての性能特性は最良（best）、典型（typical）、最悪（worst）の 3 種のサイクル数で表現される。非常に決定的アーキテクチャがあり、ある命令の性能変化が殆どないことがあり得る。これは 3 つの値が、同じでなければ、似通った値を持つことで表現される。ソフトウェアツールはこの情報を、値の偏差を調べることによってハードウェアの動的特性或いは決定性を決定するために用いることが可能である。そのようなハードウェアでは、SHIM XML に基づく推定はとても正確になりえ、誤り率は SHIM ターゲットの 20%を下回る（1 桁台の誤差率になることすらあり得る）。例えば、ある命令を殆ど 2 サイクルで実行するが、とある場合に 200 サイクルの時もあるといった、明確な動的性能特性を持ちうるハードウェアもある。この動的振る舞いは、しばしば広範囲の投機的アルゴリズムや確率的アルゴリズムの利用に由来する。「保証」アプローチと対になっている、このような「ベストエフォート」アプローチは非常に人気であり、その傾向が続く。

Software View - what is in and what is not で述べるように、SHIM はソフトウェアに対して、下にあるハードウェアに対するより単純な視点を提供し、サポートされている投機的アルゴリズムや、その詳細なスペックの記述などは回避している。三つ組の性能表現は、注意深く 3 つの値を並べ、下層にある様々なハードウェア機構をカプセル化することによって、その動的特性に適合する窓を提供する。その結果、ハードウェア性能推定を 100%正確に達成することは、不可能でないとすれば、簡単ではない。考え方は、システムアーキテクチャ設計にとって十分な性能推定の正確さを手に入れることであり、残りは後のシステム開発の段階で最適化されるべきである。このアプローチは、プロジェクト進行において、システム開発段階の前、例えば設計段階、において最終的なソフトウェア群が利用可能ではない時には手頃である。

SHIM.xml は具体的なハードウェア設定（と、必要ならばシステムソフトウェア）について作成される。もし、例えば、quarity of service（QoS）が性能特性に誤り率の目標である 20%を超えるレベルで影響を与えるものである場合、複数の SHIM XML ファイルを記述するか、Common Configuration File を性能変化のために記述しなければならない。

2.4.2 Latency and Pitch

性能オブジェクトは 2 種類の三つ組で特徴付けられる。一つは「遅延時間」、もう一つは「ピッチ」と結びついている（図 6）。遅延時間、SHIM クラス用語では *Latency* は、ある処理を実行するプロセッササイクル数を示す。*Pitch* は手の込んだ処理である。これは、連続して処理を実行する際の、1 歩の大きさのことで、これもプロセッササイクル数で表現される。前述のとおり、現代のハードウェアは次のソフトウェアの行動が何であるかを投機する機構を持つ。例えば、メモリにアクセスする時、ハードウェアは、例え特定の「load」命令によってより小さい大きさのメモリを要求されたとしても、メモリをラインサイズで読むキャッシュを持っている。本質的に、次のメモリアドレスは前もって読むことができ、次の「load」命令が連続的なアドレスに続くことが望まれる（投機的フェッチと言われている）。もしもこの投機が真であるならば、次の読み込み処理はキャッシュから読むことで完了でき、より遅いメインメモリに実際のアクセスをしない。ハードウェアは他の、反復的なソフトウェアの振る舞いを利用する、すべての同様の機能をサポートしている。この動作は、似通った処理が何らかの方法で繰り返し行われるときに、処理あたりの平均の実行サイクルが繰り返しでない時の実行サイクル数よりも少なくなるような結果をもたらす。*Pitch* はこの性能特性を明確に意味している。

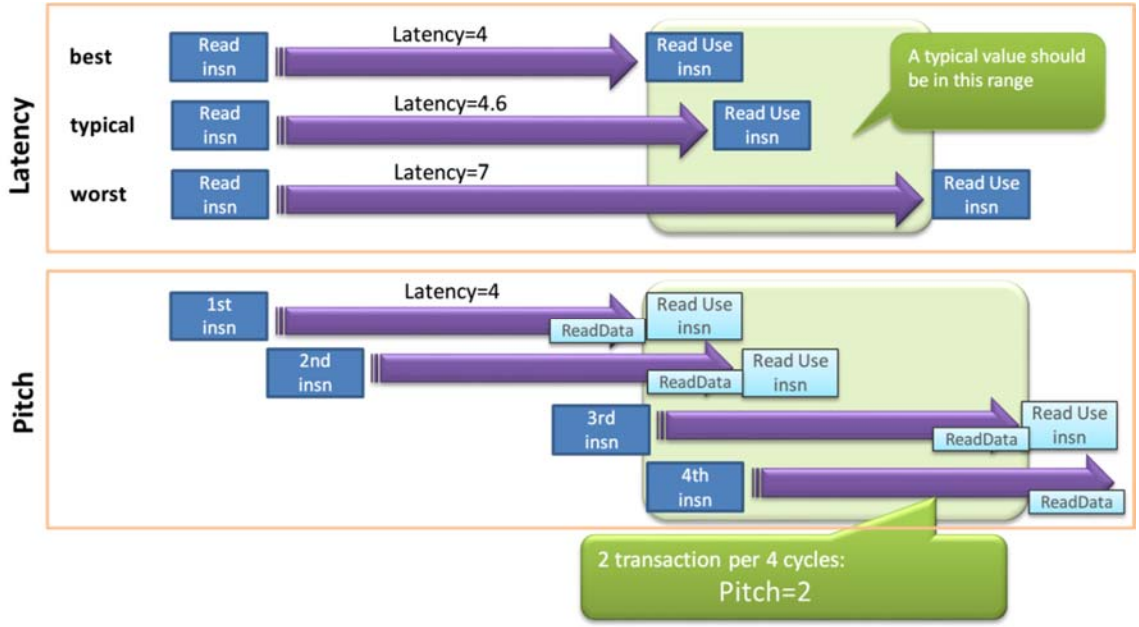


Figure 6. Latency and Pitch represent the primary performance characteristics.

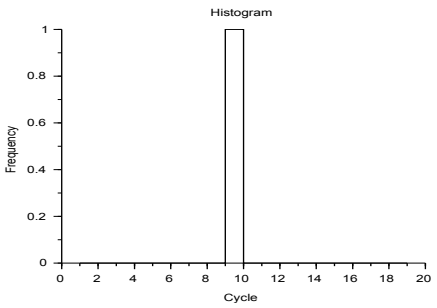
ソフトウェアツールの責務は、特定の処理が繰り返されているかを見て、*Latency* と *Pitch* の三つ組を適切に利用することである。

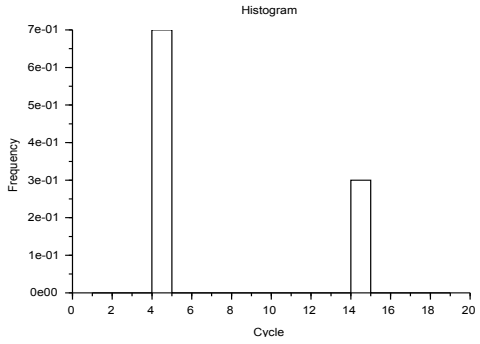
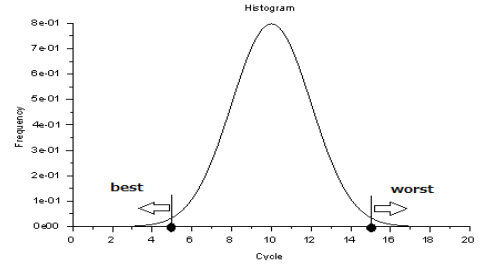
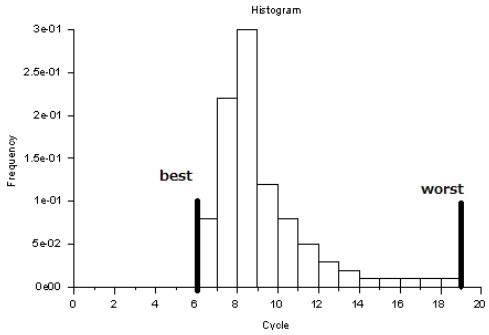
2.4.3 Using triplets

性能特性に影響する任意の要素は、性能変化を記述するために、「最良（best）」、「典型（typical）」、「最悪（worst）」の三つ組を用いて表現しなければならない。これは、表４に例と共に記述されている。

なお、確率用語でいえば、典型（typical）は平均ではなく最頻値(mode)を意味している。

Table 4. Using Triples

	1 つの数字である性能を記述できるような単純な場合、三つ組の表記法は明確に等しい値を 3 つ記述することは依然として有用である。 例：10.0, 10.0, 10.0

	分布が 2 値に存在し、3 値はより明確な表現の余地がある。「典型（typical）」値がより頻繁に発生する条件を表す。 例：5.0, 5.0, 15.0
	理想的な場合に正規分布が観測される場合、「典型（typical）」値は平均に等しい。最良（best）と最悪（worst）値は平均から標準偏差の 3 倍以上の値を表す。 例：5.1, 10.0, 14.9
	実際の分布では、マシンの性能限界の計測値は時々非常に難しくなる。このような場合、分布の 99.9%累積点（これは、正規分布における 3σ 点とほぼ等しい）を最悪（worst）値、0.1%を最良（best）値に採用する。典型（typical）値は分布の最頻値を表す。 例：6.0, 8.5, 18.9

2.5 Software View – what is in and what is not

Introduction で述べたとおり、ツールは SHIM を、マルチコア/メニーコアハードウェア上で動作するソフトウェア開発を手助けすることを主として用いるべきである。したがって、SHIM 仕様を定義する鍵となる戦略は、ツールが必要とする情報に限定してハードウェアを記述することである。我々はこれを「ソフトウェア視点」と呼び、「ハードウェア視点」と対を成すものとして考えている。ここで「ハードウェア視点」とは、一部のソフトウェア開発支援ツールにとって非常に重要である場合を除いて、プロセッシングコア間接続の物理的/電気的方法や、特定コアによるメモリ読み込み要求に対するルーティングに使われる NoC(Network on Chip)プロトコル、プロセッサパイプラインステージ数、キャッシュコヒーレンシプロトコル等に焦点をおくことを意味している。

ハードウェア特性を SHIM に追加することは、魅力的でかつ比較的容易であるのだが、これは SHIM XML をより複雑なものにすることとなり、スキーマを理解するのにより多くの努力を必要とし、ツールがこれらの情報を利用する努力を複雑にする。その上、最も重要なことは、SHIM XML を創出することなのである。このことが、SHIM 標準を限定的な採用に導いている。

基本的な原則はアーキテクチャ設計レベルでソフトウェアに影響する特性を捉えることである。すなわち、設計支援ツールが SHIM を用い、SHIM XML で記述された特定のハードウェアに適したソフトウェア設計を作り出したならば、システム開発のより後の段階においてソフトウェアアーキテクチャレベルでの変更が必要になるべきではない。

「ソフトウェアアーキテクチャ設計レベル」が基準線ではあるけれども、時として特定のハードウェア特性に対して必要性が議論になる場合がある。経験的には、実際のユースケースが（想像可能であったとしても）導くことが出来ない場合には、SHIM 仕様からは除外すべきである。

様々な理由で、かなりの数の潜在的なハードウェア特性が現在の仕様には含まれてこなかった。最重要な理由の一つは、除外されたハードウェア特性の種類が、既に仕様に含まれる特性の周辺だったことである。我々は発展的なアプローチを採用し、まずはより基本的な特性を安定させることにした。そのようなハードウェア特性は将来のバージョンに含まれる可能性はある。

含まれているものに選ばれた、最も基本となる特性は、トポロジ、アドレス空間、コア間通信、性能と構成設定である。

2.6 XML

SHIM インタフェースは拡張マークアップ言語（XML）を用いる。特に、SHIM XML は XML の構造を定義するために XML スキーマ定義（XSD）を用いる。XSD は本質的には UML クラス定義と同じである。各 SHIM XML ファイルは固有のハードウェアを表しているが、すべては SHIM XML スキーマに準じていなければならない。SHIM XML スキーマの UML クラス図による表現は図 3 に示した。

XML スキーマは SHIM XML の構造の定義を与えているが、XML を読むパーサを用いることにより、SHIM XML の正当性確認にも利用可能である。そのようなパーサは、オープンにおいても商用においても既に利用可能であり、しばしば様々な種類のプログラミング言語の XML 関連ライブラリとともにバンドルされている。

したがって、技術的に言うと、SHIM XML スキーマ、或いは shim.xsd は、SHIM の中心的なインタフェース定義である。

2.6.1 Data Binding

XML ファイルを読むための共通の技術は、SAX 或いは DOM ライブラリを経由することである。XSD を用いると、shim.xsd に対してスキーマコンパイラを実行することで、様々なプログラミング言語の選択肢に対してクラスライブラリを生成することが可能である。生成されたクラスライブラリは選択されたプログラミング言語におけるすべての SHIM XML クラスを含んでおり、データを取得及び設定するメソッド或いは関数が自動的に追加される。これはツールに対して、そのプログラミング言語における通常のオブジェクトに似た方法で SHIM XML で表現されたハードウェア特性にアクセスする方法を提供している。

2.6.2 Who Creates SHIM XML

ハードウェア提供者が SHIM XML を作成、提供し、それがソフトウェアツールによって使われることが期待されている。一方で、ハードウェア提供者は SHIM XML を提供しないかもしれない。もしも SHIM XML がハードウェア提供者しか作成できないのであれば、様々なハードウェアへの SHIM 適用に対して重大な障害となるだろう。そのため、Reference Authoring Tools が仕様に加えて無償公開された。もしユーザが基本的なテクニカルリファレンスマニュアルや、シミュレータや実ハードウェア（例：評価ボード）のいずれかにアクセスできるならば、Reference Authoring Tools は、殆どのマルチコア/メニーコアハードウェアの SHIM XML を 1～2 日以下で作る機会を提供する。

2.7 Configuration

2.7.1 General

ソフトウェアツールが必要とする、SHIM 関連の構成設定には 2 つの視点がある。1 つの視点は基本的なハードウェア特性（例：クラスタ構成、コア数、メモリサイズ、プロセッサ ISA）に基づくソフトウェアツールの構成設定である。これらは静的なハードウェア特性であり、ソフトウェアツールは SHIM XML ファイルを読み込み、それに応じて自分自身を構成することができる。もう 1 つの視点は、システム設計に応じて修正することが出来る、ハードウェアの動的特性（例：クロック周波数、転送アクセラレータの様々なモードや設定）の構成設定である。動的特性に対して、ツールの利用者はしばしば構成設定を入力する必要があり、したがってツールは（コマンド或いはグラフィカルな）ユーザーインタフェースを提供する必要がある。SHIM は、構成可能な属性の記述と同時にユーザーインタフェース定義に利用可能な、Common Configuration File (CCF)とよぶ仕組みを提供している。

構成設定を変更することは、しばしば性能特性に影響する。そのため、CCF は特定の設定構成要素に対する選択や入力値が性能特性にどのように影響を与えるかも記述することが可能である。

2.7.2 Common Configuration File (CCF)

CCF は構成可能なハードウェア要素を記述するとともに、サポートするツールに対し構成設定のユーザーインタフェースを生成する標準的な方法を定義するように SHIM を拡張する。CCF は構成可能な要素を CCF XML と呼ばれる、SHIM XML とは分離された XML ファイルに記述する。SHIM を使うソフトウェアツールは、この仕組みをツール内部にある Configuration tool user interface か、別のスタンドアロンなツールを提供することによって利用可能である。SHIM XML や CCF とともに構成設定ツールを実行することにより、ツール利用者による入力に応じて SHIM XML の特定の場所を変更する仕組みが提供され、ツールによって自動的に変更することも可能となる。

SHIM XML と CCF は XPath という XML パス言語（ある XML 文書からノードを選択するクエリ言語）を経由して相互に繋がっている。それに加えて、XPath は XML 文書の中身に応じて値（例：文字列、数字、あるいは論理値）を計算することに使われるかもしれない。

ここに実際の CCF の例を示す。

396

ccf-sample-for-shim.xml

```
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]
```

Unicode(UTF-8) 11pt

397 **Figure 7. CCF example**

398 この CCF は、CCF を利用できるツールで開いたとき、次のような GUI を動的に生成する（これは MCA からソースコード
399 提供されている CCF のサンプルアプリケーションである）。

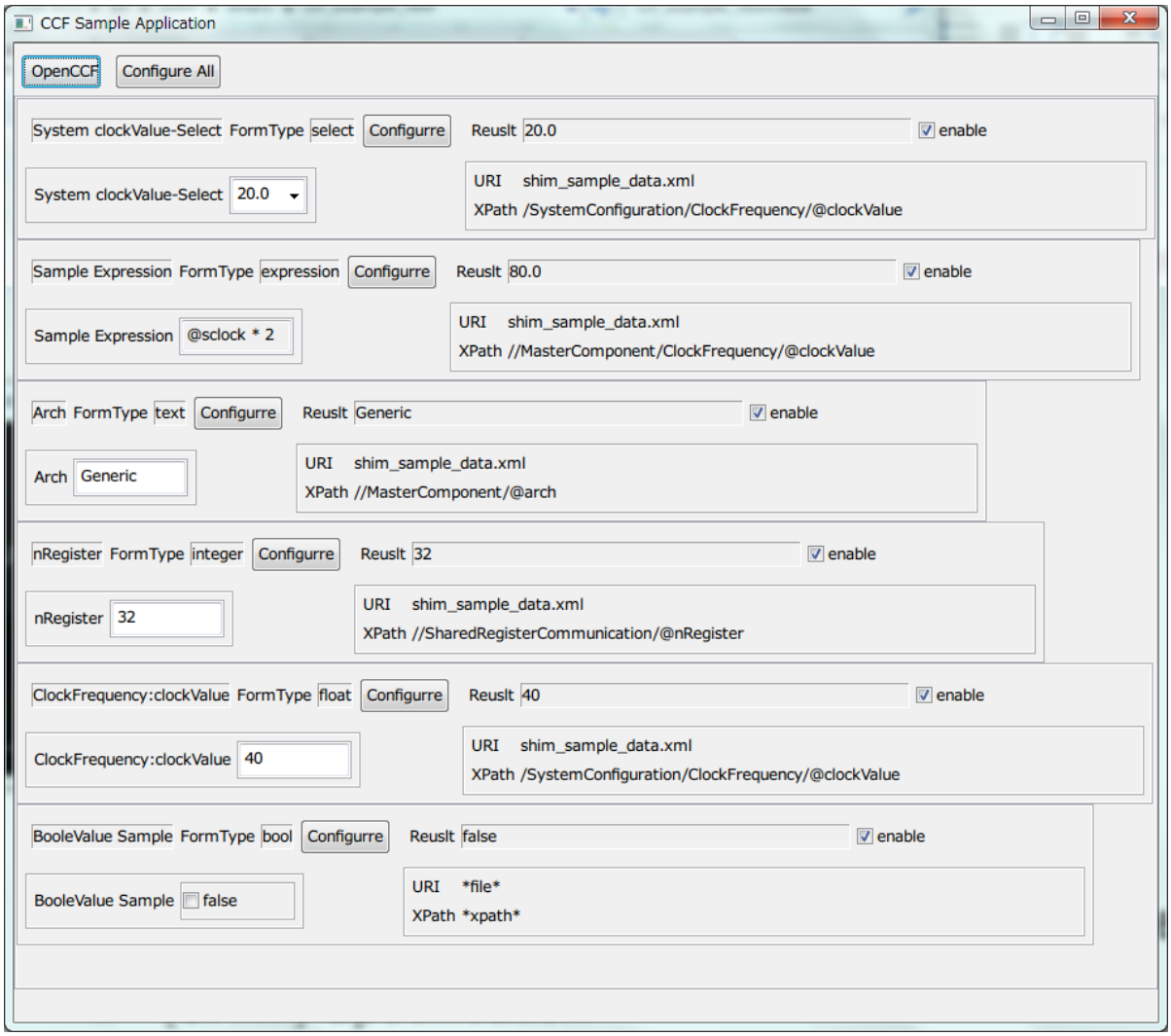


Figure 8. GUI generated by CCF

Common Configuration File を参照すること。

2.8 Reference Authoring Tools

仕様そのものに加えて、SHIM はリファレンスオーサリングツールも無償提供する。リファレンスとして、誰もが SHIM オーサリングツールの自身のバージョンを提供することが可能である。マルチコアアソシエーション（MCA）は、SHIM Editor というリファレンスオーサリングツールを次の理由で提供する。

1. SHIM XML 適用を推進するための簡単なオーサリング
2. サンプル SHIM アプリケーションのソースコード提供

2.9 Roadmap

SHIM の最初のバージョンは基本かつ重要なハードウェア特性を含んでおり、多くのツールにとって有用だろう。しかしながら、Software View - what is in and what is not で述べたとおり、この最初のバージョンでは網羅されてこなかった要素もある。SHIM はオープンな技術であり、より広範囲への適用が革新的な利用を巻き起こすだろう。これには仕様の拡張が必要かもしれない。我々はこれらの変更をオープンにしておきたいと思っている。

将来の仕様に含まれることが考慮中の特性：デバッグ/トレース、電力消費、基本的な周辺機能。また、SHIM XML のコンポーネント化、ハードウェアに関係したソフトウェア特性、そして XML をより小さくするためのスキーマ改良のような、要素についてはここで説明しておく価値が有るだろう。

2.9.1 Componentization of SHIM XML

現在のバージョンの SHIM はスタンドアロンでなければならず、これは、SHIM XML ファイルに記載されるハードウェアプラットフォーム（例：仮想シミュレータや実評価ボード）は実行可能でなければならないことを意味する。しかしながら、各ボードに向けて専用設計されるマルチコア/メニーコアチップは稀である。したがって、同じチップがしばしば複数のボードに配置される。この場合、現状では、共通のマルチコア/メニーコアチップに対する SHIM XML 記述が冗長になるとしても、それぞれのボードに対して別々の SHIM XML ファイルを作成することになる。

ひとたび SHIM の利用が広まり始めれば、複数の SHIM XML ファイルにわたってファイル内部の特定のコンポーネント記述を再利用することは自然である。このことは本質的に SHIM XML のコンポーネント化であり、既に次のメジャーバージョンの SHIM において含まれることが考慮中である。一方で、対象のハードウェアに似ている既存の SHIM XML ファイルをもとに新たな SHIM XML を書くことも可能である。SHIM エディタは、勿論既存の SHIM XML ファイルを編集することが可能である。

SHIM XML のコンポーネント化への課題は、*MasterComponent* のような SHIM XML クラスが、それが含まれているハードウェアボード設計に対して不変な特性も含んでおり、特定のハードウェアボード上への搭載に依存する特性も含んでいることである。この 2 つの特性は *MasterComponent* の SHIM XML を再利用するために切り離されなければならない。1 つの考え方は Common Configuration File（CCF）を利用することであり、これにより、最終的な SHIM XML ファイルもしくは CCF 内部のどこかで見つけられる他の情報に基づいて性能値を適合させることができる。

SHIM XML 自身のコンポーネント化に加えて、SHIM と IP-XACT を適切な場面で提携させる可能性を検討中である。これにより、IP-XACT XML ファイルに含まれる情報を用いて半自動的に SHIM XML を記述するために、SHM オーサリングツールが IP-XACT XML ファイルをインポートできるようにする。*ComponentSet* に含まれるオブジェクトは、最低でもトポロジ部分とオブジェクト名はインポート可能であるべきであり、一方で他の部分は SHIM XML に固有なものであるため、追加する必要がある。

2.9.2 Hardware-Related Software Properties

SHIM が記述するハードウェア特性に加え、OS や更にはミドルウェアといったシステムソフトウェアの特性に依存するツールも存在する。例えば、並列化コンパイラなどといった並列設計支援ツールにとって OS の排他制御プリミティブ性能は、特定の処理において適切なロック機構を選ぶために重要である。同様に、ツールはいくつかのメッセージ受け渡し機構の性能についても知る必要があるかもしれない。現在は、この種の情報は SHIM には含まれていない。理由の一部は、異な

444 るシステムソフトウェア実装とライブラリインタフェース定義に対して、別々の SHIM XML ファイルを用意する必要があるか
445 らである。将来のバージョンの SHIM はこの種の情報を網羅するよう拡張するかもしれない。

446 2.9.3 Schema Refinement for Smaller XML

447 SHIM スキーマは単純であろうとしているが、同時にホモジニアスおよびヘテロジニアスハードウェアの両方をサポートするこ
448 とを許している。これは、複数の同一クラスタ構成からなるハードウェアのように、複数の同じコンポーネントのインスタンスが
449 あるホモジニアスハードウェアの XML に対し、同じ記述の繰り返しを招く。もしもクラスタが、各クラスタが異なるプロセッサ
450 コアの構成を持つヘテロジニアスであるならば、XML の行数は変わらないが同じ記述の繰り返しにはならない。もしも
451 SHIM がスキーマ内の冗長記述を表現する機構を提供できれば、ホモジニアスなハードウェアに対する SHIM XML の
452 大きさを減らすことが出来る。Componentization of SHIM XML に加えて、このことを考慮する予定である。

3. SHIM Interface

SHIM インターフェースの主要部は SHIM XML 自身である。それゆえインターフェースを理解することは XML スキーマを理解することである。基本は SHIM Concept の章に記載されており、次のグループに分かれている。

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

次節以降において、上述の各グループに対するスキーマおよび記述内容が説明される。各グループに含まれるオブジェクトおよび XML 要素に対し、記述内容と例が与えられる。

スキーマは、様々なスキーマコンパイラを用い、異なるプログラム言語に変換される。ツールやユースケースに依存して変わるため、SHIM 仕様はプログラム言語仕様を含まない。しかしながら、Java を主たる言語の一つと想定し、Java クラスの SHIM API ライブラリを提供する。いくつかのユーティリティは Java による参考実装によって定義されており、SHIM クラスライブラリがプログラミングを容易にする。

以降では、各グループの XML 要素、属性、Java クラスライブラリへのポインタを説明する。

3.1 shim.xsd

SHIM XML スキーマファイルのことである。要素記述については以降を参照されたい。

```
<?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">
```

```
545         <xs:sequence>
546                 <xs:element name="MemoryConsistencyModel" type="MemoryConsistencyModel"
547       minOccurs="0" maxOccurs="unbounded"/>
548                 <xs:element name="MasterSlaveBindingSet" type="MasterSlaveBindingSet"
549       minOccurs="0" maxOccurs="1"/>
550         </xs:sequence>
551         <xs:attribute name="name" use="required" type="xs:string"/>
552         <xs:attribute name="id" use="required" type="xs:ID"/>
553         <xs:attribute name="start" use="required" type="xs:long"/>
554         <xs:attribute name="end" use="required" type="xs:long"/>
555         <xs:attribute name="endian" use="optional" type="EndianType"/>
556       </xs:complexType>
557       <xs:simpleType name="MasterType">
558         <xs:restriction base="xs:string">
559                 <xs:enumeration value="PU">
560                         <xs:annotation>
561                                 <xs:documentation>Processing Unit</xs:documentation>
562                         </xs:annotation>
563                 </xs:enumeration>
564                 <xs:enumeration value="TU">
565                         <xs:annotation>
566                                 <xs:documentation>Transffer Unit</xs:documentation>
567                         </xs:annotation>
568                 </xs:enumeration>
569                 <xs:enumeration value="OTHER"/>
570         </xs:restriction>
571       </xs:simpleType>
572       <xs:element name="Instruction" type="Instruction"/>
573       <xs:complexType name="Instruction">
574         <xs:sequence>
575                 <xs:element name="Performance" type="Performance" minOccurs="1" maxOccurs="1"/>
576         </xs:sequence>
577         <xs:attribute name="name" use="required" type="xs:string"/>
578       </xs:complexType>
579       <xs:element name="InterruptCommunication" type="InterruptCommunication"/>
580       <xs:complexType name="InterruptCommunication">
581         <xs:complexContent>
582                 <xs:extension base="AbstractCommunication">
583                         <xs:sequence/>
584                 </xs:extension>
585         </xs:complexContent>
586       </xs:complexType>
587       <xs:element name="Latency" type="Latency"/>
588       <xs:complexType name="Latency">
589         <xs:complexContent>
590                 <xs:extension base="AbstractPerformance">
591                         <xs:sequence/>
592                 </xs:extension>
593         </xs:complexContent>
594       </xs:complexType>
595       <xs:element name="AbstractPerformance" type="AbstractPerformance"/>
596       <xs:complexType name="AbstractPerformance" abstract="true">
597         <xs:sequence/>
598         <xs:attribute name="best" use="optional" type="xs:float"/>
599         <xs:attribute name="typical" use="required" type="xs:float"/>
600         <xs:attribute name="worst" use="optional" type="xs:float"/>
601       </xs:complexType>
602       <xs:element name="Pitch" type="Pitch"/>
603       <xs:complexType name="Pitch">
604         <xs:complexContent>
605                 <xs:extension base="AbstractPerformance">
606                         <xs:sequence/>
607                 </xs:extension>
608         </xs:complexContent>
609       </xs:complexType>
610       <xs:element name="MasterSlaveBinding" type="MasterSlaveBinding"/>
611       <xs:complexType name="MasterSlaveBinding">
612         <xs:sequence>
613                 <xs:element name="Accessor" type="Accessor" minOccurs="1" maxOccurs="unbounded"/>
614         </xs:sequence>
615         <xs:attribute name="slaveComponentRef" use="required" type="xs:IDREF"/>
616       </xs:complexType>
617       <xs:element name="CommunicationSet" type="CommunicationSet"/>
618       <xs:complexType name="CommunicationSet">
619         <xs:sequence>
```

```
620             <xs:element name="SharedRegisterCommunication" type="SharedRegisterCommunication"
621     minOccurs="0" maxOccurs="unbounded"/>
622             <xs:element name="SharedMemoryCommunication" type="SharedMemoryCommunication"
623     minOccurs="0" maxOccurs="unbounded"/>
624             <xs:element name="EventCommunication" type="EventCommunication" minOccurs="0"
625     maxOccurs="unbounded"/>
626             <xs:element name="FIFOCommunication" type="FIFOCommunication" minOccurs="0"
627     maxOccurs="unbounded"/>
628             <xs:element name="InterruptCommunication" type="InterruptCommunication"
629     minOccurs="0" maxOccurs="unbounded"/>
630        </xs:sequence>
631     </xs:complexType>
632     <xs:element name="AbstractCommunication" type="AbstractCommunication"/>
633     <xs:complexType name="AbstractCommunication" abstract="true">
634        <xs:sequence>
635             <xs:element name="ConnectionSet" type="ConnectionSet" minOccurs="0"
636     maxOccurs="1"/>
637        </xs:sequence>
638        <xs:attribute name="name" use="required" type="xs:string"/>
639     </xs:complexType>
640     <xs:element name="Connection" type="Connection"/>
641     <xs:complexType name="Connection">
642        <xs:sequence>
643             <xs:element name="Performance" type="Performance" minOccurs="0"
644     maxOccurs="unbounded"/>
645        </xs:sequence>
646        <xs:attribute name="from" use="required" type="xs:IDREF">
647             <xs:annotation>
648                 <xs:documentation>Reference to the instance of
649     MasterComponent</xs:documentation>
650             </xs:annotation>
651        </xs:attribute>
652        <xs:attribute name="to" use="required" type="xs:IDREF">
653             <xs:annotation>
654                 <xs:documentation>Reference to the instance of
655     MasterComponent</xs:documentation>
656             </xs:annotation>
657        </xs:attribute>
658     </xs:complexType>
659     <xs:element name="PerformanceSet" type="PerformanceSet"/>
660     <xs:complexType name="PerformanceSet">
661        <xs:sequence>
662             <xs:element name="Performance" type="Performance" minOccurs="0"
663     maxOccurs="unbounded"/>
664        </xs:sequence>
665     </xs:complexType>
666     <xs:element name="FIFOCommunication" type="FIFOCommunication"/>
667     <xs:complexType name="FIFOCommunication">
668        <xs:complexContent>
669             <xs:extension base="AbstractCommunication">
670                 <xs:sequence/>
671                 <xs:attribute name="dataSize" use="required" type="xs:int"/>
672                 <xs:attribute name="dataSizeUnit" use="optional" type="SizeUnitType"/>
673                 <xs:attribute name="queueSize" use="required" type="xs:int"/>
674             </xs:extension>
675        </xs:complexContent>
676     </xs:complexType>
677     <xs:element name="CommonInstructionSet" type="CommonInstructionSet"/>
678     <xs:complexType name="CommonInstructionSet">
679        <xs:sequence>
680             <xs:element name="Instruction" type="Instruction" minOccurs="1"
681     maxOccurs="unbounded"/>
682        </xs:sequence>
683        <xs:attribute name="name" use="required" type="xs:string"/>
684     </xs:complexType>
685     <xs:element name="Cache" type="Cache"/>
686     <xs:complexType name="Cache">
687        <xs:sequence>
688             <xs:element name="cacheRef" type="xs:IDREF" minOccurs="0" maxOccurs="unbounded"/>
689        </xs:sequence>
690        <xs:attribute name="name" use="required" type="xs:string"/>
691        <xs:attribute name="id" use="required" type="xs:ID"/>
692        <xs:attribute name="cacheType" use="required" type="CacheType">
693             <xs:annotation>
694                 <xs:documentation>soft / hard</xs:documentation>
```

```
695                 </xs:annotation>
696             </xs:attribute>
697             <xs:attribute name="cacheCoherency" use="required" type="CacheCoherencyType"/>
698             <xs:attribute name="size" use="required" type="xs:int"/>
699             <xs:attribute name="sizeUnit" use="required" type="SizeUnitType"/>
700             <xs:attribute name="nWay" use="optional" type="xs:int"/>
701             <xs:attribute name="lineSize" use="optional" type="xs:int"/>
702             <xs:attribute name="lockDownType" use="optional" type="LockDownType"/>
703         </xs:complexType>
704         <xs:element name="SystemConfiguration" type="SystemConfiguration"/>
705         <xs:complexType name="SystemConfiguration">
706             <xs:sequence>
707                 <xs:element name="ComponentSet" type="ComponentSet" minOccurs="1" maxOccurs="1"/>
708                 <xs:element name="CommunicationSet" type="CommunicationSet" minOccurs="0"
709         maxOccurs="1"/>
710                 <xs:element name="AddressSpaceSet" type="AddressSpaceSet" minOccurs="0"
711         maxOccurs="1"/>
712                 <xs:element name="ClockFrequency" type="ClockFrequency" minOccurs="1"
713         maxOccurs="1"/>
714             </xs:sequence>
715             <xs:attribute name="name" use="required" type="xs:string"/>
716             <xs:attribute name="shimVersion" use="required" type="xs:string"/>
717         </xs:complexType>
718         <xs:element name="ConnectionSet" type="ConnectionSet"/>
719         <xs:complexType name="ConnectionSet">
720             <xs:sequence>
721                 <xs:element name="Connection" type="Connection" minOccurs="1"
722         maxOccurs="unbounded"/>
723             </xs:sequence>
724         </xs:complexType>
725         <xs:simpleType name="CacheCoherencyType">
726             <xs:restriction base="xs:string">
727                 <xs:enumeration value="SOFT"/>
728                 <xs:enumeration value="HARD"/>
729             </xs:restriction>
730         </xs:simpleType>
731         <xs:element name="MemoryConsistencyModel" type="MemoryConsistencyModel"/>
732         <xs:complexType name="MemoryConsistencyModel">
733             <xs:sequence/>
734             <xs:attribute name="rawOrdering" use="optional" type="OrderingType">
735                 <xs:annotation>
736                     <xs:documentation>Read After Write</xs:documentation>
737                 </xs:annotation>
738             </xs:attribute>
739             <xs:attribute name="warOrdering" use="optional" type="OrderingType">
740                 <xs:annotation>
741                     <xs:documentation>Write After Read</xs:documentation>
742                 </xs:annotation>
743             </xs:attribute>
744             <xs:attribute name="wawOrdering" use="optional" type="OrderingType">
745                 <xs:annotation>
746                     <xs:documentation>Write After Write</xs:documentation>
747                 </xs:annotation>
748             </xs:attribute>
749             <xs:attribute name="rarOrdering" use="optional" type="OrderingType"/>
750         </xs:complexType>
751         <xs:simpleType name="OrderingType">
752             <xs:restriction base="xs:string">
753                 <xs:enumeration value="ORDERD"/>
754                 <xs:enumeration value="UNORDERD"/>
755             </xs:restriction>
756         </xs:simpleType>
757         <xs:simpleType name="EndianType">
758             <xs:restriction base="xs:string">
759                 <xs:enumeration value="LITTLE"/>
760                 <xs:enumeration value="BIG"/>
761             </xs:restriction>
762         </xs:simpleType>
763         <xs:element name="SharedRegisterCommunication" type="SharedRegisterCommunication"/>
764         <xs:complexType name="SharedRegisterCommunication">
765             <xs:complexContent>
766                 <xs:extension base="AbstractCommunication">
767                     <xs:sequence/>
768                     <xs:attribute name="dataSize" use="required" type="xs:int"/>
769                     <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:extension>
            </xs:complexContent>
        </xs:complexType>
        <xs:element name="EventCommunication" type="EventCommunication"/>
        <xs:complexType name="EventCommunication">
            <xs:complexContent>
                <xs:extension base="AbstractCommunication">
                        <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: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

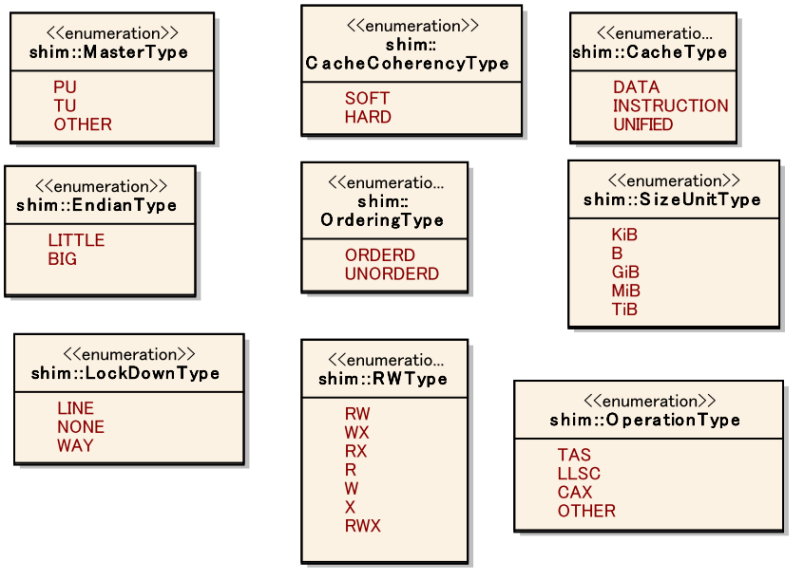
- インターフェースは *Enumeration*, *SystemConfiguration*, *ComponentSet*, *AddressSpaceSet*, *CommunicationSet*. にグループ化される。各グループでは **SCHEMA** と **DESCRIPTION** が記載される。

- 各グループは、別々の小節においてオブジェクトが記述される。その中では **DESCRIPTION** と **EXAMPLE** が記載される。

- オブジェクトと属性が**太字** で記載され、型は*斜字*で記載される。

3.3 Enumeration

SCHEMA



DESCRIPTION

Enumeration は SHIM オブジェクト属性において使われる様々な定数を定義する特別なグループである。オブジェクトはいくつかの属性において定数を使う。属性がある enumeration を値として取る時、その属性タイプには使用する enumeration タイプが記載される。

定義されている enumeration タイプを以下に記述する。

- ***MasterType*** は *MasterComponent* のタイプを示す。値は PU (Processor Unit, 例えば CPU), TU (Transfer Unit, 例えば DMA), OTHER のいずれかである。
- ***EndianType*** はエンディアン、すなわち byte-order を示す。
- ***LockDownType*** はサポートされているキャッシュ lockdown 処理の種類を示す。可能な値は LINE (line-lockdown), WAY (way-lockdown), and NONE (サポート無)のいずれかである。
- ***CacheCoherencyType*** s はサポートされているキャッシュ一貫性機構を示す。可能な値は HARD（ハードウェアベースの一貫性支援）または SOFT（ソフトウェアベースの一貫性支援）である。
- ***OrderingType*** はサポートされているメモリコンシステンシモデルを示す。可能な値は ORDERED（ordered memory consistency）または UNORDERED (unordered memory consistency)である。
- ***RWType*** はメモリアクセスの種別を示す。可能な値は R (read), W (write), X (execute), RW (R かつ W), RWX (R, W, X すべて), WX (W かつ X), and RX (R かつ X)である。

- ***CacheType*** はキャッシュの種別を示す。可能な値は DATA（データキャッシュ）, INSTRUCTION（命令キャッシュ）, UNIFIED（統合キャッシュ）である。

- ***SizeUnitType*** はデータサイズとして記載される数値の単位を示す。可能な値は B (byte), KiB (kilo binary byte), MiB (binary byte), GiB (giga binary byte), TiB (tera binary byte)である。

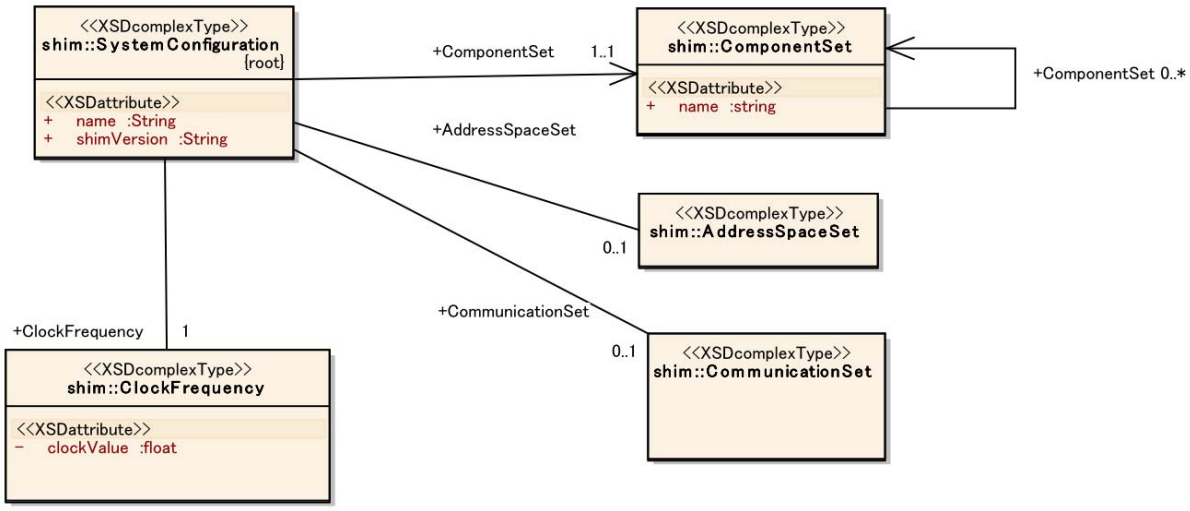
- ***OperationType*** は共有メモリ通信種別を示す。可能な値は TAS (Test and Set), LLSC (Load-link/Store Conditional), CAX (Compare and Exchange), OTHER（その他）である。

EXAMPLE

これらの値を用いた例については次節以降を参照されたい。

3.4 SystemConfiguration

SCHEMA



DESCRIPTION

SystemConfiguration は root オブジェクトである。すべての SHIM XML は root としてこのオブジェクトを持つ。

- **SystemConfiguration** (必須): root オブジェクト。String 型の名前、一つの *ComponentSet*, 一つの *ClockFrequency*, 0 組以上の *AddressSpaceSet* と *CommunicationSet*.を持つ。
- **name** (必須; type: *string*): この SHIM 記述の名前。
- **shim Version** (必須; type: *string*): SHIM インターフェース仕様のバージョン。本仕様書の SHIM バージョンは "1.0"である。後ろにマイナーリビジョン番号をつけてもよい (例"1.0.1")。
- ComponentSet, AddressSpaceSet, CommunicationSet を参照。

EXAMPLE

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

3.5 ClockFrequency

DESCRIPTION

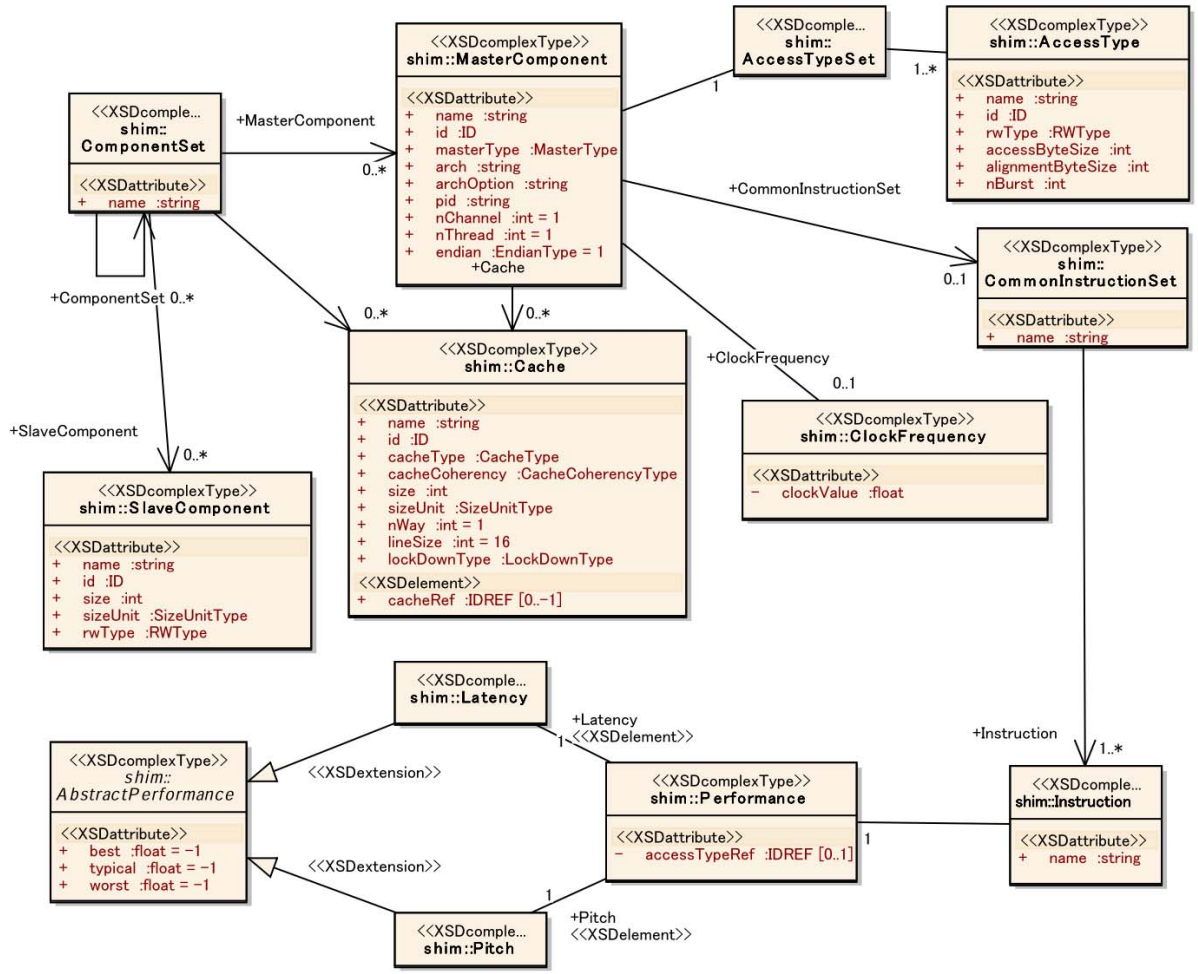
`ClockFrequency`: 後に続くオブジェクトや属性に対するシステムクロック周波数。

- **clockValue** (必須; type *float*): クロック周波数（単位：Hz）
- **EXAMPLE**

SystemConfiguration 参照

3.6 ComponentSet

SCHEMA



DESCRIPTION

ComponentSet はハードウェアコンポーネントに対するトポロジ記述の root である。**name** (必須)属性を持つ。
MasterComponent, Cache, *SlaveComponent*, 他の *ComponentSet* を持つことができる。

3.6.1 MasterComponent

DESCRIPTION

MasterComponent はプロセッサコア、アクセラレータ (DMA アクセラレータを含む), その他マスタとなり得るすべての種類のコンポーネントである。次のオブジェクト、属性を持つ。

- **AccessTypeSet** (必須): AccessTypeSet 参照。
- **CommonInstructionSet** (任意): CommonInstructionSet 参照。
- **name** (必須; type *string*): オブジェクトの名前。ハードウェアリファレンスマニュアルと同じテキスト名を使うべきである。
- **id** (必須; type *ID*): オブジェクトの ID。
- **masterType** (必須): マスタの種類。型は ***MasterType***。
- **arch** (必須; type *string*): はアーキテクチャの名前を記載する。主にプロセッサの命令セットアーキテクチャの記述を意図している。アーキテクチャリファレンスマニュアルやそれに類する文書に一般的に記載されている、公式に ISA を認識できる名前を使うことを勧める。
- **archOption (任意):** 追加のアーキテクチャ属性。
- **pid** (任意): この *MasterComponent* の ID。複数コアが存在するときに、この *MasterComponent* を識別するためのプロセッサ ID として使われることを意図している。プロセッサ ID の識別方法は様々な方法があり得るが、ここではアーキテクチャリファレンスマニュアルに使われている方法に従うべきである。ID は整数値でない場合もあり得るので、*string* 型としている。
- **nChannel** (任意; type *int*): チャネル数を記載する。**masterType** が **TU** である場合に DMA のチャネル数を記載することを想定している。
- **nThread** (任意; type *int*): ハードウェアスレッドをサポートするプロセッサにおいてハードウェアスレッド数を記載する。
- **translation** (任意; type *string*): MMU のようなアドレス変換機構の有無を記載する。
- **protection** (任意; type *string*): メモリ保護機能を持つプロセッサにおいて、保護の種別を記載する。
- **endian** (任意; type *EndianType*): オブジェクトのエンディアン種類。

EXAMPLE

```
<MasterComponent name="Core_0_0_0" id="SHIMEDITOR2533184940882014100513014297 4" 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="SHIMEDITOR2723742239312014100513014555 1" rwType="R"
    accessByteSize="4" alignmentByteSize="4" nBurst="8"/>
    ...
   </AccessTypeSet>
</MasterComponent>
```

3.6.2 SlaveComponent

DESCRIPTION

SlaveComponent はメモリのようなスレーブとなるコンポーネントを記載する。以下のオブジェクト、属性を持つ。

- **name** (必須; type *string*): オブジェクトの名前。
- **id** (必須; type *ID*): オブジェクトの ID。
- **size** (必須; type *int*): メモリのサイズ。
- **sizeUnit** (必須; type *SizeUnitType*): 上記 **size** の単位。
- **rwType** (必須; type *RWType*): メモリの read/write 属性。

EXAMPLE

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

3.6.3 Cache

DESCRIPTION

ここではキャッシュを記述する。以下のオブジェクト、属性を持つ。

- **cacheRef** (任意; type *IDREF*): **MasterComponent** から 1 レベル遠い **Cache** の **id** を記載する。
- **name** (必須; type *string*): オブジェクトの名前。
- **id** (必須; type *ID*): オブジェクトの ID。
- **cacheType** (必須; type *CacheType*): キャッシュの種別。

- **cacheCoherency** (必須; type *CacheCoherencyType*): キャッシュ一貫性機構の種類。
- **size** (必須; type *int*): キャッシュのサイズ。
- **sizeUnit (**必須; type *SizeUnitType*)**:** 上記 **size** の単位。
- **nWay** (任意; type *int*): キャッシュウェイ数。
- **lineSize** (任意; type *int*): キャッシュ 1 ラインのサイズ。
- **lockDownType** (任意; type *LockDownType*): サポートされる lockdown 処理の種類。

EXAMPLE

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

3.6.4 AccessTypeSet

DESCRIPTION

一つ以上の AccessType をセットにする。次のようなオブジェクト、属性を持つ。

- **AccessType** (必須): AccessType 参照。

EXAMPLE

```
<AccessTypeSet>
   <AccessType name="AT_0_0_0_0" id="SHIMEDITOR272374223931201410051301455551" rwType="R"
  accessByteSize="4" alignmentByteSize="4" nBurst="8"/>
   <AccessType name="AT_0_0_0_1" id="SHIMEDITOR298051298213201410051301455552" rwType="R"
  accessByteSize="8" alignmentByteSize="8" nBurst="8"/>
</AccessTypeSet>
```

3.6.5 AccessType

DESCRIPTION

アクセスに関する属性を記述する。メモリアクセスを意図しているがそれに限定しているわけではない。次のようなオブジェクト、属性を持つ。

- **name** (必須; type *string*): オブジェクトの名前。
- **id** (必須; type *ID*): オブジェクトの ID。
- **rwType** (任意; type *RWType*): アクセス種別を記載する。
- **accessByteSize** (任意; type *int*): アクセスのデータサイズを記載する。（単位：byte）
- **alignmentByteSize** (任意; type *int*) アクセスのアラインメント制約を記載する。（単位：byte）

- **nBurst** (任意; type *int*): バース長を記載する。バーストサイズは accessByteSize とする。masterType=TU を想定している。

EXAMPLE

See AccessTypeSet.

3.6.6 CommonInstructionSet

DESCRIPTION

このオブジェクトには *MasterComponent* においてサポートされる命令を表す Instruction の集合を記載する。XML スキーマには陽に書かれていないが、常に LLVM 命令セット[2]を記載する必要があり、必要ならば拡張することもできる。LLVM 命令セットの各命令は、*MasterComponent* の命令もしくは命令列に対応する。次のオブジェクトもしくは属性を持つ。

- **Instruction** (必須): Instruction 参照
- **name** (必須; type *string*): オブジェクトの名前。

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>
```

3.6.7 Instruction

DESCRIPTION

命令を記載する。次のオブジェクトもしくは属性を持つ。

- **Performance** (必須): Performance 参照。
- **name** (必須; type *string*): オブジェクトの名前。

[2] http://llvm.org/docs/LangRef.html#instruction-reference

EXAMPLE

CommonInstructionSet 参照。

3.6.8 Performance

DESCRIPTION

性能について記載する。次のオブジェクトもしくは属性を持つ。

- **Latency** (必須): Latency 参照。
- **Pitch** (必須): Pitch 参照。
- **accessTypeRef** (任意; type *IDREF*) AccessType **id** への参照。メモリアクセス性能を記述する際に使うことを想定している。

EXAMPLE

CommonInstructionSet 参照。

3.6.9 Latency

DESCRIPTION

次のオブジェクトもしくは属性を持つ。Latency and Pitch を参照されたい。

- **best** (任意; type *float*): 最良(best)ケースのプロセッササイクル数。
- **typical** (必須; type *float*): 典型(typical)ケースのプロセッササイクル数。
- **worst** (任意; type *float*): 最悪(worst)ケースのプロセッササイクル数。

EXAMPLE

CommonInstructionSet 参照。

3.6.10 Pitch

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

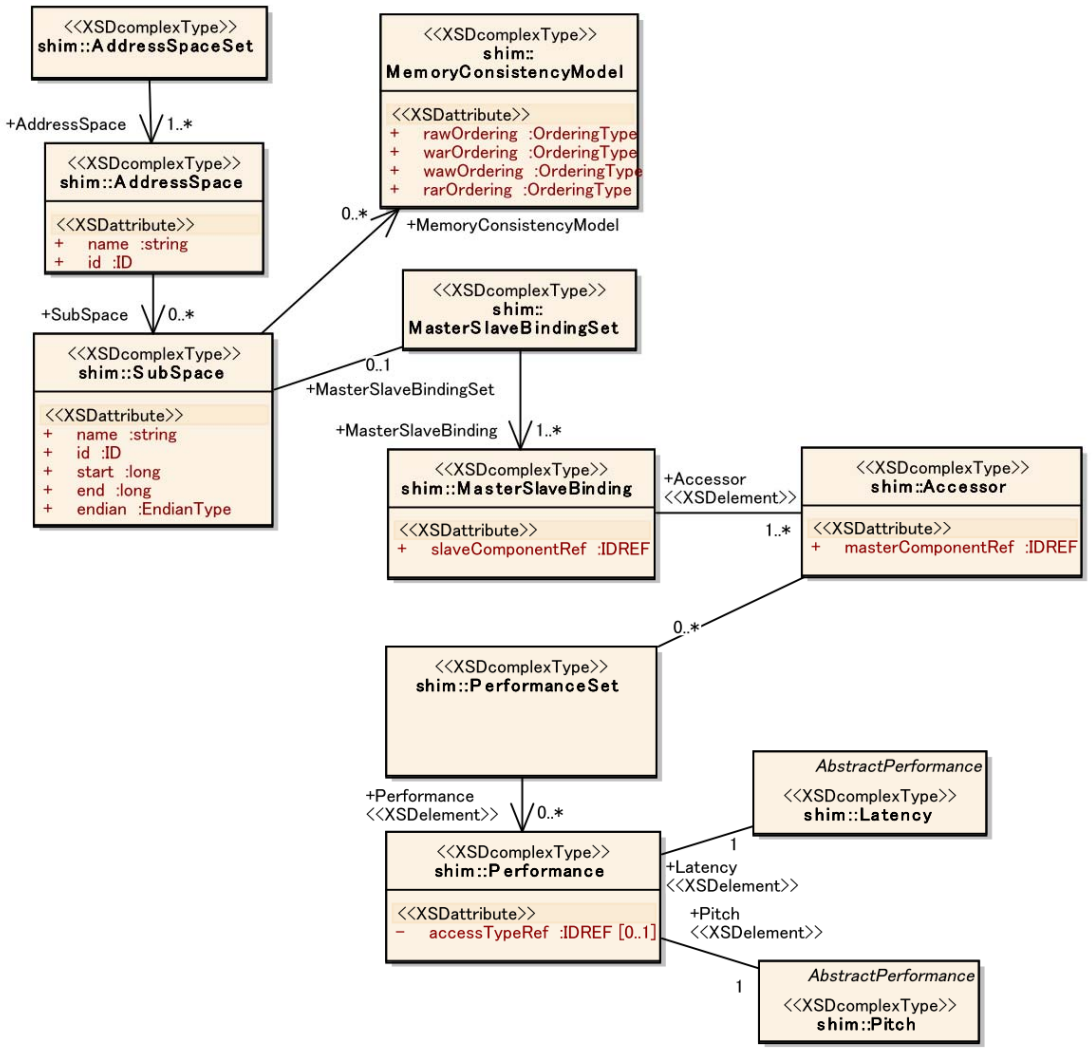
- **best** (任意; type *float*): 最良(best)ケースのプロセッササイクル数。
- **typical** (必須; type *float*): 典型(typical)ケースのプロセッササイクル数。
- **worst** (任意; type *float*): 最悪(worst)ケースのプロセッササイクル数。

EXAMPLE

CommonInstructionSet 参照。

3.7 AddressSpaceSet

SCHEMA



DESCRIPTION

AddressSpaceSet にはメモリアドレス空間の構成、および *MasterComponent* と *SlaveComponent* との対応関係について記載する。

3.7.1 AddressSpace

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **SubSpace** (任意): SubSpace 参照。
- **name** (必須; type *string*): オブジェクトの名前。
- **id** (必須; type *ID*): オブジェクトの ID。

EXAMPLE

```
<AddressSpaceSet>
   <AddressSpace name="AS_0_0" id="SHIMEDITOR22375265206920141005130143354">
      <SubSpace name="SS_0_0_0" id="SHIMEDITOR914093813232014100513014335 5" 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="SHIMEDITOR272374223931201410051301455 51">
                       <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>
```

3.7.2 SubSpace

DESCRIPTION

このオブジェクトでは *AddressSpace* における一つの区間について記載する。次のオブジェクトもしくは属性を持つ。

- **MasterSlaveBindingSet** (必須): MasterSlaveBindingSet 参照。
- **MemoryConsistencyModel** (必須): MemoryConsistencyModel 参照。
- **name** (必須; type *string*): オブジェクトの名前。
- **id** (必須; type *ID*): オブジェクトの ID。
- **start** (必須; type *long*): 開始アドレス。
- **end** (必須; type *long*): 終了アドレス。
- **endian** (任意; type *EndianType*): エンディアン。

EXAMPLE

AddressSpace 参照。

3.7.3 MemoryConsistencyModel

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **rawOrdering** (任意; type *OrderingType*): read-after-write アクセスにおけるメモリオーダリングを記載する。
- **warOrdering** (任意; type *OrderingType*): write-after-read アクセスにおけるメモリオーダリングを記載する。
- **wawOrdering** (任意; type *OrderingType*) write-after-write アクセスにおけるメモリオーダリングを記載する。
- **rarOrdering** (任意; type *OrderingType*): read-after-read アクセスにおけるメモリオーダリングを記載する。

EXAMPLE

AddressSpace 参照。

3.7.4 MasterSlaveBindingSet

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **MasterSlaveBinding** (必須): MasterSlaveBinding 参照。

EXAMPLE

AddressSpace 参照。

3.7.5 MasterSlaveBinding

DESCRIPTION

MasterComponent と *SlaveComponent* との対応付けを記載する。次のオブジェクトもしくは属性を持つ。

- **Accessor** (必須): Accessor 参照。
- **slaveComponentRef** (必須; type *IDREF*): *SlaveComponent* の **id** を記載する。

EXAMPLE

AddressSpace 参照。

3.7.6 Accessor

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **PerformanceSet** (任意): PerformanceSet 参照。
- **masterComponentRef** (必須; type *IDREF*): *MasterComponent.*の **id** を記載する。

EXAMPLE

AddressSpace 参照。

3.7.7 PerformanceSet

DESCRIPTION

１つ以上の Performance をまとめる。次のオブジェクトもしくは属性を持つ。

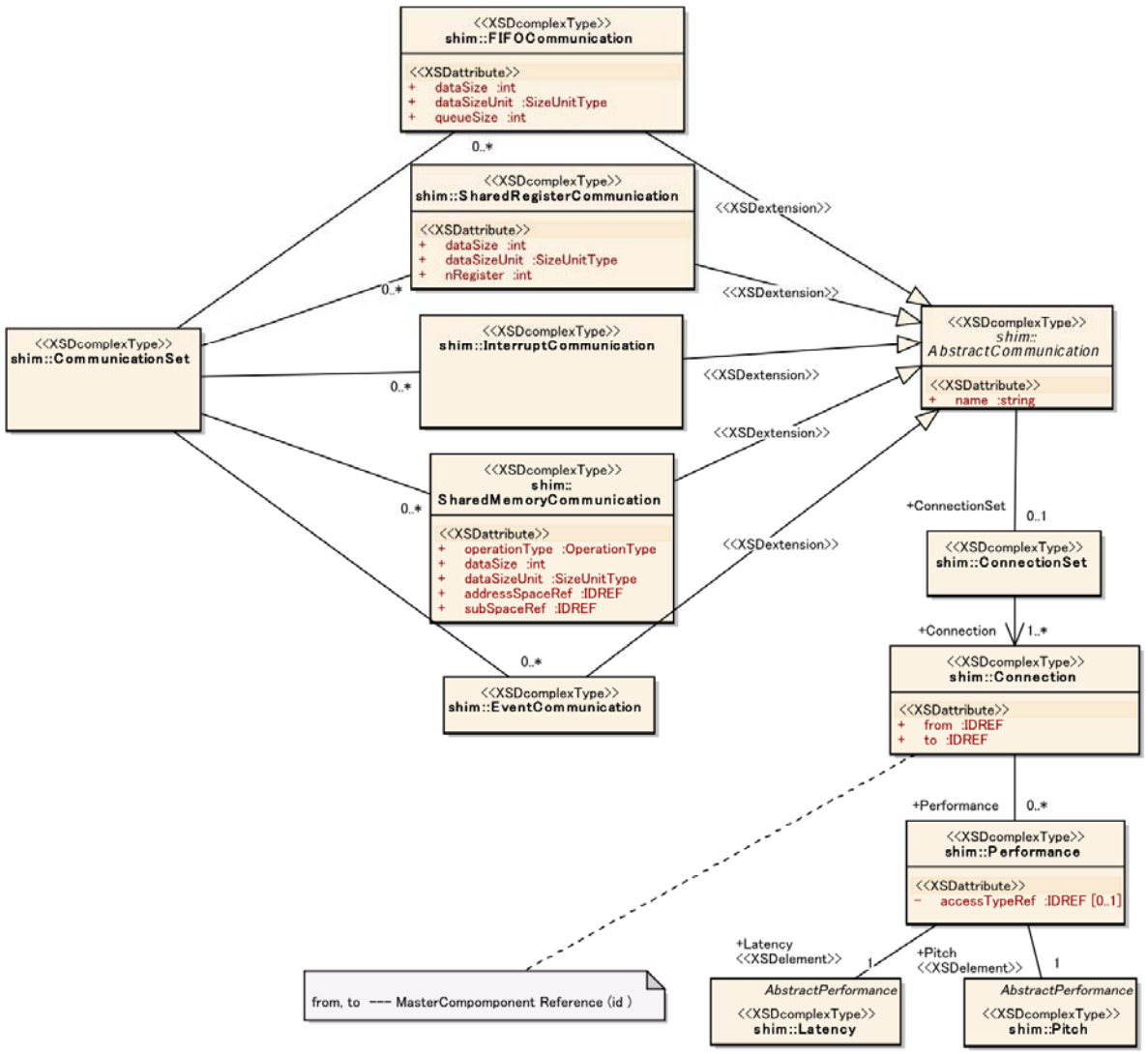
- **Performance** (任意): Performance 参照。

EXAMPLE

AddressSpace 参照。

3.8 CommunicationSet

SCHEMA



DESCRIPTION

CommunicationSet は利用可能な *MasterComponent*-to-*MasterComponent* 通信を記載する。異なる通信タイプを記載するための 6 つのオブジェクトがある。

3.8.1 FIFOCommunication

DESCRIPTION

FIFO ベースの通信を記載する。次のオブジェクトもしくは属性を持つ。

- **ConnectionSet** (必須): ConnectionSet 参照。
- **name** (必須; type *string*): オブジェクトの名前。

- **dataSize** (必須; type *int*): FIFO のデータサイズ。
- **dataSizeUnit** (必須; type *SizeUnitType*): 上記 **dataSize** の単位。
- **queueSize** (必須; type *int*): キューのサイズ(単位は上記 dataSize)。従って FIFO の全体容量は dataSize * dataSizeUnit * queueSize となる。

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>
```

3.8.2 SharedRegisterCommunication

DESCRIPTION

共有レジスタベースの通信を記載する。次のオブジェクトもしくは属性を持つ。

- **ConnectionSet** (必須): ConnectionSet 参照。
- **name** (必須; type *string*): オブジェクトの名前。
- **dataSize** (必須; type *int*): 共有レジスタのデータサイズ。
- **dataSizeUnit** (必須; type *SizeUnitType*): 上記 **dataSize** の単位。
- **nRegister** (必須; type *int*): 共有レジスタ数。

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>
```

3.8.3 InterruptCommunication

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **ConnectionSet** (必須): ConnectionSet 参照。

- **name** (必須; type *string*): オブジェクトの名前。

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>
```

3.8.4 SharedMemoryCommunication

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **ConnectionSet** (必須): ConnectionSet 参照。

- **name** (必須; type *string*): オブジェクトの名前。

- **operationType** (任意; type *OperationType*): 共有メモリ通信の種別。

- **dataSize** (任意; type *int*): *SharedMemoryCommunication* のデータサイズ。

- **dataSizeUnit** (必須; type *SizeUnitType*): 上記 **dataSize** の単位。

- **addressSpaceRef** (任意; type *IDREF*): このオブジェクトが使用する共有メモリに対する *AddressSpace* の **id** を記載する。もしもこの属性が記載されておらず、かつ次の *subSpaceRef* も記載されていない場合には、この *SharedMemoryCommunication* 機構はすべての *AddressSpace* と *SubSpace* に対して有効であるものとする。

- **subSpaceRef** (任意; type *IDREF*): この *SharedMemoryCommunication* がサポートする *SubSpace* の **id** を記載する。この属性が記載されるときには必ず上述の *addressSpaceRef* も記載されている必要がある。上述の *addressSpaceRef* が記載されており、この属性が省略されているときには、この *SharedMemoryCommunication* 機構は上述の *AddressSpace* に属するすべての *SubSpace* に対して有効であるものとする。もしも *addressSpaceRef* が省略され、*subSpaceRef* が記載されている場合には、解釈は未定義であるため、使用されるべきではない。

EXAMPLE

```
<SharedMemoryCommunication dataSize="128" dataSizeUnit="KiB"
    addressSpaceRef="SHIMEDITOR223752652069201410051301433 54"
    subSpaceRef="SHIMEDITOR914093813232014100513014335 5" 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>
```

3.8.5 EventCommunication

DESCRIPTION

イベントベースの通信を記載する。次のオブジェクトもしくは属性を持つ。

- **ConnectionSet** (必須): ConnectionSet 参照。
- **name** (必須; type *string*): オブジェクトの名前。

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>
```

3.8.6 ConnectionSet

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **Connection** (必須): Connection 参照。

EXAMPLE

各種通信オブジェクトの例を参照。

3.8.7 Connection

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **Performance** (optional): Performance 参照。

- **from** (必須; type *IDREF*): コネクションの開始点となる *MasterComponen* の **id**。
- **to** (必須; type *IDREF*): コネクションの終端となる *MasterComponen* の **id**。

EXAMPLE

各種通信オブジェクトの例を参照。

4. Use Cases

ここでは SHIM XML に含まれる情報の扱い方を例としたいくつかのユースケースを示す。ここで紹介するツールのカテゴリは例であり、ユーザが概念を理解する手助けを行う。この例は別タイプのツールにも適用できるかもしれない。

4.1 Performance Estimation: Auto-Parallelizing Compiler

Table 5. Performance Estimation Use case

説明するツール(ID)	自動並列コンパイラ(APC1)
適用可能性	マルチ・メニーコアハードウェアの性能特徴を知ることで恩恵を受けるツール
説明する SHIM 要素	*ClockFrequency*, *MasterComponent*, *CommonInstructionSet*, Latency, Pitch, Cache, FIFOCommunication
ツール処理の概要	コンパイラは逐次 C コードを用いてスレッドレベルの並列 C コードを出力する。入力されるコードは最初に解析され、コードに含まれる命令、用いるメモリのサイズ、アクセスタイプ(rwx とアクセス幅)、制御フロー、全体のデータフローを分析する。このフロー分析に基づいて、コードは、利用可能なコア数に合わせ、各スレッドがおおよそ同じサイクル数を消費するように、複数のスレッドに分割される。データ配置は、各コアで利用可能なキャッシュサイズ、メモリアクセスレイテンシ、コア間通信レイテンシに基づき最適化される。

4.1.1 Using “CommonInstructionSet”

- “*MasterComponent*” (プロセッサコアなど)はそれぞれ“*ClockFrequency*”と“*CommonInstructionSet*”をもつ
- “*CommonInstructionSet*”は LLVM IR 命令として定義される
- “*CommonInstructionSet*”はそれぞれの命令の性能(プロセッサのサイクル数)の、最良サイクル数、典型サイクル数、最悪サイクル数を保持している
- クロック値とサイクル情報があれば、ハードウェア上の特定の命令の実行時間を見積もることが可能である

4.1.2 Using “PerformanceSet”

- メモリの read/write 操作、もしくはロード/ストア命令において、性能値はそのデータが存在する *SubSpace* のレイテンシとピッチから計算される
- レイテンシ、ピッチは最良/典型/最悪サイクル数を持つ。通常は典型値が用いられるが、ツールが繰り返しのメモリアクセスを認識することが可能ならば、最良値が用いられる
- メモリの性能特徴とデータ使用方法に基づいて、コンパイラはデータを配置すべき最適なメモリを選択する

4.1.3 Using “Cache”

- まず、ツールは Cache::cacheCoherency を読み、ハードウェアのキャッシュコーヒレンシがサポートされているかどうかを決定する。もしサポートされていなければ必要な箇所にソフトウェアベースのコヒーレンシ操作を挿入するが、一方でその操作が最小になるようにデータやスレッドをコアに配置する
- Cache::blockSize はキャッシュラインのサイズである。ツールによってデータ配置最適化に用いられる
- Cache::size はキャッシュのサイズであり、ツールによって最適ワークデータサイズの判断に用いられる

4.1.4 Using “FIFOCommunication”

- *FIFOCommunication* を含む全ての *CommunicationSet* 要素は、どのペアの *MasterComponents* がこの通信によって接続されているかということを示した Connection(s)を含んだ *ConnectionSet* を持っている
- *FIFOCommunication* はデータサイズとキューサイズを持ち、ツールによってデータ転送単位を決定するために用いられる
- すべての *XXXCommunication* はサイクル単位で表されたレイテンシとピッチを含む性能情報を持つ。これを用いることでツールはその通信チャネルを用いたデータ転送の実行コストを決めることができる

4.2 Tool Configuration - RTOS Configuration Tool

Table 6. Tool Configuration Use case

説明するツール(ID)	RTOS やミドルウェアのようなランタイムソフトウェアの設定ツール(RTS1)
適用可能性	このモデルを利用することでランタイムソフトウェアの特定のコンフィギュレーションファイルを作ることが出来る。このようなコンフィギュレーションファイルを必要とする他のホストツールにも適用することができる。
説明する SHIM 要素	*ClockFrequency*, *SlaveComponent*, *SubSpace*, *MasterSlaveBinding*, Common Configuration File (CCF)
ツール処理の概要	RTOS はコンフィギュレータを持つ。コンフィギュレータは RTOS コンフィギュレーション C コードを生成し、それらの C コードは後にコンパイルされ RTOS のライブラリにリンクされる。このコンフィギュレータは GUI を持ち、それによりユーザは RTOS のブートコードによりセットされる PU クロック、および RTOS メモリプールのメモリアドレスとサイズを選択/指定することができる

4.2.1 Using “ClockFrequency”

- それぞれの“*MasterComponent*”(プロセッサコアなど)は“*ClockFrequency*”をもつ
- 値の属性は XPath 表現を含むことができる。XPath は設定値を表すことに用いる別の XML ファイル CCF (Common Configuration File) を指し示す
- このシナリオでは、選択可能な値は CCF の一部に記載され、それは *select* と呼ばれる FormType を持つとする

- コンフィギュレータは CCF を読み、*select* FormType を読み込んだとき、選択可能な値が記入されたコンボボックス GUI コントロールオブジェクトを動的に表示する
- GUI のテキストラベルは *ClockFrequency* の“name”属性から得ることができ、親テキストラベルは関連する *MasterComponent* から得ることができる
- これらの名前は生成される C コード内の #define シンボルのもとにすることができる
- したがって、ツールはクロック周波数の設定を知ることなく、その機能を提供している

注:他の設定可能な要素も同じ方法を用いるか、もしくはツールが XML から意図的に対応する要素を見つけその値を用いることにより設定できる。

4.2.2 Using “SubSpace”

- コンフィギュレータはアドレスとサイズに基づいた RTOS メモリプールを設定することができる
- そのためには、利用可能なメモリのアドレス、サイズを知る必要がある
- ツールはまず“*ComponentSet*”内の全ての“*SlaveComponents*”を確認し、“RWType”属性が“rw”であるものに対し、その *SlaveComponent* の名前を記録する
- さらに、ツールは“*AddressSpace*”下の“*SubSpace*”に進み、その *SubSpace* に関連する“*MasterSlaveBinding*”をチェックする
- *MasterSlaveBinding* は“*SlaveComponentRef*”属性を含んでいるが、ツールはそれが記録した名前とマッチするか確認を行わなければならない
- マッチする *SubSpace* が見つかれば、利用可能なメモリ空間としてユーザに名前、始点アドレス、終点アドレスを表示する

4.3 Hardware Modeling

このユースケースは Performance estimation - auto-parallelizing compiler と同じ SHIM クラスを用いる。よって、SHIM オブジェクトにアクセスするための実際の手順の説明は割愛する。表 7 には基本の考え方を示す。

Table 7. Hardware Modeling Use Case

説明するツール(ID)	軽快でシンプルなハードウェアモデリングツール(HM1)
適用可能性	このモデルは仮想ハードウェア機能を提供するどのようなツールにも適用可能である。もしそのようなツールが SHIM XML をインポートできれば、モデリング機能自体がより高度な機能を提供するように発展するだろう
説明する SHIM 要素	*ComponentSet*, Latency/Pitch 他, Performance estimation - auto-parallelizing compiler に示された要素

ツール処理の概要	ツールはあるマルチコアハードウェアが記述された SHIM XML から始まる。性能分析ツールが SHIM XML とソフトウェア一式を受け取る。APC1 内で述べたような処理により、SHIM XML に記述されたハードウェア上でのソフトウェアの粗い静的性能解析ができる。ハードウェアモデリングツールは“*ComponentSet*”を操作することにより、特定の性能を有するプロセッサコアの追加や、メモリのレイテンシ/ピッチの変更をおこなう。修正された SHIM XML はもう一度静的性能解析ツールに入力され、新しいハードウェアモデルの性能の変化を見ることができる

1392

5. SHIM XML Authoring Rules and Guidelines

この章では新しい SHIM XML ファイルを作成するための規則とガイドラインを定義する。規則は従わなければならないこと、ガイドラインは推奨である。以降の節には規則[Rules]かガイドライン[Guidelines]を示す。

SHIM XML を使用するソフトウェアツールはこれらの規則（可能ならばガイドラインも）に従う SHIM XML ファイルを要求する。ガイドラインに従わないことは可能だが、SHIM XML 提供者は期待される使用法を保証する責任があり、SHIM XML を使用するツールは特定のガイドラインに従わないファイルでも問題なく処理されなければならない。

5.1 File Name [Rule]

SHIM Editor は自動的に SHIM XML ファイルの名前を作成しないので、適切な名前を指定する必要がある。ファイル名は SHIM XML がどのようなものかを具体的に表す。多くの場合は、SHIM XML はハードウェアボードを表しており、そこには複数のチップやメモリが搭載されているかもしれない。ファイル名はハードウェアの最外部要素を表す。そのため、もしもそれがハードウェアボードであるならば、ボードの名前、もしくはハードウェアボードの特徴を現す一意の名前を用いる。もし SHIM XML が実際のボードではなく仮想ハードウェアプラットフォームの場合は、仮想プラットフォームを表す特定の名前を使用する。

また、ハードウェア要素のほかにバージョン情報もファイル名に必要である。バージョン管理システムを使用している場合であっても、SHIM XML ファイルをバージョン管理システムの管理外に持ち出すとき混乱しないように、バージョン情報を追加する。バージョン情報は英数字で表し、同じ SHIM XML ファイルの異なるバージョンと区別できるようにする。例えばバージョン管理システムのリビジョンや yyyy-mm-dd 形式の修正日付、独自のものなどがある。

もし対象ハードウェアのプロセッサアーキテクチャに対応しているコンパイラが複数ある場合、パフォーマンスを計測した際に使われたコンパイラの情報を入れることには価値がある。同時にバージョンやリビジョンなどの情報もファイル名に載せるべきである。また、もし SHIM で記述されているハードウェアが多種プロセッサ（複数 ISA）を搭載し、複数のコンパイラが使われている場合、compilerName と compilerVersion は、上記複数のツールを統合、パッケージ化した SDK やツールチェインを表すこととする。

ファイル名は一意でなくてはならない。そのためにはインターネットのドメイン名を Java パッケージ[3]名に使用するような方法が最善である。上記の説明内容をすべて合わせると、SHIM XML ファイル名の具体例は次のようになる。

domainName.hardwarePlatformName.platformVersion.compilerName.compilerVersion.xml

例えば、XYZ 社が製造した ABC 評価ボードのバージョン 1.0.0、インターネットのドメイン名が www.xyz.com、性能計測したコンパイラが gcc 4.9.0 の場合の SHIM XML ファイル名は以下のようになる。

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

[3] http://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html

ハードウェアプラットフォームの名前が漠然としている場合、名前を区別するため、例えば搭載プロセッサ名と統合した名前にするようにプラットフォーム名を拡張するのが適切である。

また、SHIM XML ファイルを分類するためにプラットフォーム特有の情報をハードウェアプラットフォーム名に含むように拡張しなければならない。これは、ハードウェアプラットフォームに複数の動作モードがあり、これらをそれぞれ記述する複数の SHIM XML 作成する場合に有効である。別の方法として構成変更可能な単一の SHIM XML ファイルを利用するために CCF を使う方法があるが、その場合には CCF と CCF 互換な SHIM XML ファイルを記述する必要がある。どちらの方法にも賛否両論があるが、どちらの方法を採用するかは裁量次第である。経験則では、安定した SHIM XML ファイルがあり、似ているが別の SHIM XML ファイルを作成しようとしている場合、そのようなことが今後多くあることがわかっている場合、複数のマイナーな変更を加えて SHIM XML ファイルを再利用することが期待されている場合には CCF を使用するべきであり、それは CCF モデルの効率的適用になるだろう。

5.2 Naming of Various Objects [Rule]

すべての SHIM オブジェクトは XML 絶対パスで表現したときに一意になるように命名しなければならない。ちょうどファイルシステムにおいて、ディレクトリ名が異なっている限り、別のディレクトリに同じ名前のファイルを持つことができることと同じである。つまり、*ComponentSet* 名が異なっている限り、同じ名前の *MasterComponent* を持つことができる。

オブジェクトの命名について一貫性も考慮する必要がある。これは SHIM ならではの話ではないが、SHIM XML 内の一貫性のある命名を維持するために重要である。もし、複数の SHIM XML ファイルを作成している場合、それらの一貫性も必要である。SHIM の仕様では、オブジェクトを区別する目的以外にオブジェクト名を必要としない。しかし、いくつかの状況で、SHIM 仕様に定義されていない重要な情報を名前が伝えることが効果的である（将来のバージョンの SHIM に追加するだろう）。それまでの間は、一貫性のある命名が隔たりを埋めるだろう。また、SHIM XML やその一部を再利用する場合に一貫性は重要である。SHIM が SHIM XML のコンポーネント化(Componentization of SHIM XML)に対応する場合、一貫性によって新しい仕様を採用することが簡単になるだろう。

5.3 Level of Detail and Precision [Guideline]

原則として、SHIM で表すことができるすべてのハードウェア特性が記述されるべきである。もし、ハードウェアマニュアルに指定された名前がある場合には、コンポーネントの名前をマニュアルの名前と一致させることを勧める。

注：ハードウェア特性の記述の省略は、必ずしもソフトウェアツールが機能しなくなることにはならない。ツールは SHIM XML ファイルを読み取ったままに扱い、何らかの機能するハードウェアとして説明されている限り、省略したかどうかを判断することはできない。よって、何を公開して何を公開しないかなどは SHIM XML 作成者の裁量範囲である。

6. Common Configuration File (CCF)

この章では構成変更可能なハードウェア（及びソフトウェア）要素を記述するための、強力で柔軟な手段を提供する共通構成設定ファイル（Common Configuration File (CCF)）について説明する。CCF は異なるハードウェア構成で同じ SHIM XML ファイルの再利用を可能にし、また一貫性のある構成設定インタフェースを提供する。さらに、ベンダー固有の機能を記述するときに利用可能であり、例えば、まだ SHIM XML スキーマには含まれていない特別な動作モードを提供するような場合、この機能を有効化したときの性能の変化を SHIM XML に記述することができる。

CCF に対応することを強く推奨するが、CCF はオプションであり、ソフトウェアツールは CCF に対応せずとも SHIM を利用することができる。CCF に対応していない場合、ツールの基本機能は SHIM XML で記述されたデフォルト設定に限定される。なお、SHIM XML が一つも CCF を参照しない場合は、逆に CCF XML が SHIM XML ファイルを参照しない場合に限定される。

6.1 Concept

6.1.1 Multiple Hardware Configuration

ハードウェアプラットフォームは多くの場合、複数の構成（例えばシステムまたはプロセッサのクロック周波数、構成可能なキャッシュサイズ、FIFO のサイズ、いくつかの動作モード）を持つ。SHIM は、異なるハードウェアを単一のインターフェースで記述できるよう、ハードウェアモデルを一般化しようとしている。しかし、クロック周波数のように、設定可能な一般的な項目がまだ多くある。もし、SHIM が設定可能なクロック周波数を表現できない場合、*ClockFrequency* のみが異なる SHIM XML ファイルを別々に作成しなければならない。

CCF は CCF XML（SHIM XML とは別の XML ファイル）と呼ばれるファイルで設定可能な項目を記述する。SHIM を使用するソフトウェアツールは、ツール内もしくは別の独立ツールとして構成設定ユーザーインタフェース（Configuration tool user interface）を提供することで、この仕組みを利用することができる。この構成設定ツールを SHIM XML や CCF と共に実行することにより、ユーザ入力やツールの自動入力に応じて SHIM XML の特定の部分を変更することができる。つまり CCF は、SHIM XML 作成者が設定項目の値だけが異なる同じ SHIM XML ファイルを記述することを軽減しつつ、ツール開発者に対して構成設定ユーザーインタフェースの開発を支援している。

6.1.2 Vendor-Specific Hardware Features Affecting SHIM Objects

スキーマで定義されていないハードウェア機構を SHIM XML に含めることはできない。この特徴によってハードウェア性能が異なってしまう結果となる。SHIM はプロセッサやメモリ、通信によってパフォーマンス特性を記述するため、定義されていない機構を記述できないことで SHIM の目標とされる 20%のエラーレートを超えて不正確なパフォーマンス評価を示してしまう。CCF を使用することで、そのようなベンダー固有のハードウェア機構を記述し、ソフトウェアツールにその機構の構成設定インタフェースを提供することができる。SHIM XML 作成者は CCF の中に、構成設定ツールのユーザ入力によって SHIM XML を修正するよう記述することができる。構成設定ユーザーインタフェースは CCF によって動的に作成されるので、ソフトウェアツールはベンダー固有の機構を把握する必要はなく、ハードウェアベンダーがそのような機構を記述する。SHIM XML は CCF によって修正され、ツールは SHIM XML の修正結果を使用すればよい。

6.1.3 Configuration Tool User Interface

しばしば、ソフトウェアツールはユーザインタフェース（UI）をグラフィカルであろうがなかろうが、一般には両方を、提供しなければならない。商用ツールは多種多様なプラットフォームに対応し、その結果として技術とビジネスの継続的な発展に必要な数のユーザの発掘を達成している。このことは、非常に幅広いハードウェアや COTS（民生利用の）ソフトウェアコンポーネントを有する組込みシステム市場において特に当てはまる。したがって、ツールにとってはこれらの構成設定を支援する効果的かつ経済的な方法を得ることが極めて重要である。CCF はこれを達成する標準的な方法の提供を目的としている。

このように多様なユーザーインタフェースを提供する問題の本質は、実際の構成がどのような項目でも固有となることである。共通の項目があっても、一般的には下位項目が異なり、時には交ざりあっている。もし、特定の変数項目ごとに構成設定インターフェースを実装する方法をツールに採用したならば、非常にコストがかかる（例えば、構成管理コスト、ソフトウェアの配布、品質維持の問題）。

この問題を緩和するための鍵は、UI の設計記述と個々の設定項目の記述を結び付け、結び付けられた記述に対して同じアルゴリズムおよびコードを使って解釈し、動的に UI を生成することである。このアプローチはすでにかなり評判があり、実証されている。問題はこれを記述するための標準がないことであり、たとえ 2 つのツールが同じ設定項目を使用していても、標準化されていないため、似ているが異なる記述を作り出すことである。CCF はこの状況を改善する。

CCF では 6 種の構成入力インタフェースオブジェクト *select*, *bool*, *text*, *integer*, *hex_integer*, *expression*（後述）を定義している。ツール開発者は構成入力インタフェースオブジェクトに対応させる UI コントロールを決めなければいけないが、select はコンボボックス、text はテキストフィールド、integer は整数フィールドのように暗黙に想定されている。コントロールは任意の組み合わせでグループ化することも、特定の設定項目（多くの場合は上位階層にある設定項目）の入力によって設定項目の一部を変えることもできる。多くのコントロールオブジェクト、すなわち CCF で *Form-Type* とよばれるオブジェクトは簡単に使用でき、組み合わせて使用することにより必要な構成項目をほとんど記述することができる。場合によって設定項目が他の複数の設定項目の値に依存するかもしれないが、その場合には算術的、論理的な方法で関係を表現する必要がある。最後に、*expression* は特殊なオブジェクトであり、様々な XPath 式の文字オブジェクトを格納するバケツのように機能する擬似インタフェースオブジェクトである。XPath によって基本的な算術演算を記述することができるため、他の項目の設定値に依存する値を計算するのに使われる。XPath を使用した算術演算記述、設定項目のグループ化と階層記述を含む FormType の支援により、CCF は、ほぼすべての構成設定インターフェースと記述に必要なものを網羅するシンプルだが強力な方法を提供する。

6.2 Interface

6.2.1 XML Schema

CCF クラス図を Figure 9. Common Configuration File (CCF) class diagram に示す。

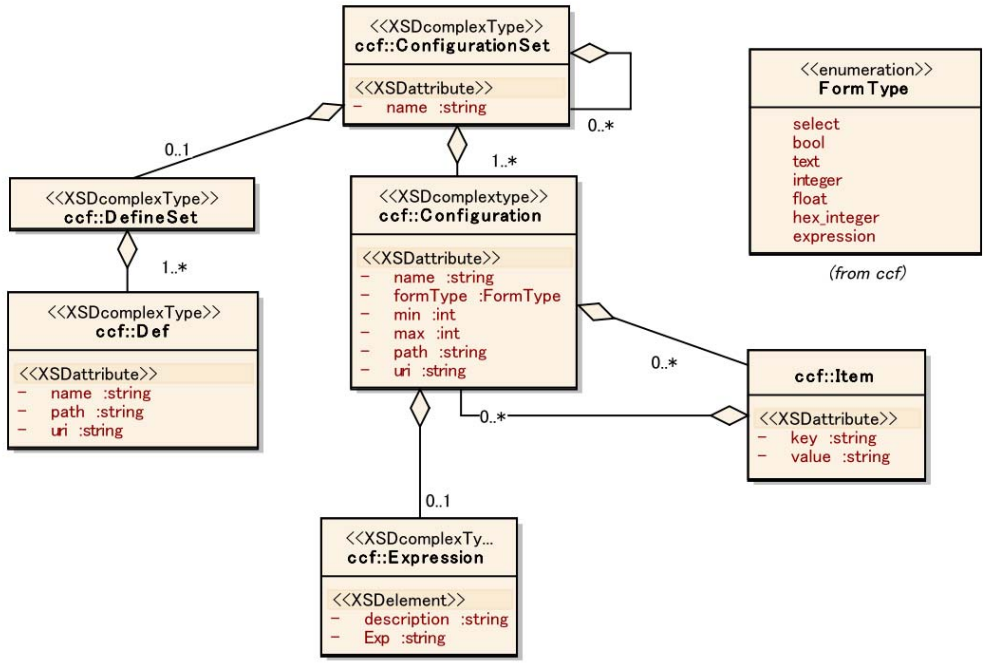


Figure 9. Common Configuration File (CCF) class diagram

```
<?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 は、使用する *FormType* を示す *Configuration* オブジェクトを少なくとも一つ含む最上位オブジェクトである。*select FormType* については、コンボボックスコントロールのエントリに含まれる複数 *Item* オブジェクトがリスト化されている。エントリとして表示するテキストとしてキー属性が使用される。この値が実際の設定値となる。値自体は多くの場合、説明を必要としないものであり、キーと値は同じである。親 *Configuration* オブジェクトの名前属性はコントロールのラベルとして使用することができる。*Configuration* オブジェクトの *FormType* が *integer* の場合は、ユーザが入力できる最大値と最小値をそれぞれ最大属性、最小属性で定義する。*ConfigurationSet* オブジェクトは、設定項目の階層構造を作るために入れ子にすることができる。また、*Item* オブジェクトは別の *Configuration* オブジェクトを下に持つことができる。これは、*FormType* が *select* の場合に、ユーザがある特定の *Item* オブジェクトを選択したときに特定の *Configuration* オブジェクト集合を必要とする際に便利である。この階層構造は特定の設定項目をグループ化するのに使用することができ、これに応じてツールは構成 UI コントロールをグループ化できる。

FormType の *Configuration* オブジェクトが *expression* の場合、XPath 式に文字通り使用される Exp 属性を持つ *Expression* オブジェクトが定義される。XPath によって CCF は基本的な計算や別の XML ファイルの値をパラメータとして使用することができる。*ConfigurationSet* オブジェクトは、*DefineSet*（C 言語の#define と同じ機能）と呼ばれるオブジェクトを含むことができる。XPath 式では、ある特定の設定項目の値をしばしば参照する。*DefineSet* が含む Def オブジェクトは短い文字列で XPath 式の中で使うことができる。この文字列は、Exp 属性や、同じ親 *ConfigurationSet* オブジェクトを共有する *Configuration* オブジェクトのパス属性の中でも使うことができる。

すべての *Configuration* オブジェクトは、それぞれの *FormType* の結果当てはめるべき場所を示す“path”属性と“uri”属性を持つ。パスは XPath 式であり、URI はターゲットの XML ファイルの位置である。CCF 作成者は *FormType* の種類とターゲットの XPath 式が一致していることに責任を負う。構成設定ツールもまた、パスと URI を使用してターゲット XML から現在の値を読み込み、デフォルト設定値を得る。したがって、構成設定ツールが起動するとき、パスと URI で明示されたターゲット XML から読み込んだ値で入力フィールドが初期化される。

expression *FormType* は、一般的には XML ファイル（通常 SHIM XML）から 1 つ以上の値を取得する。しかし、これらの値は同じ CCF の別の *Configuration* オブジェクトによって変更されるかもしれない。そのため、どの *Configuration* オブジェクトが処理されているかが重要である。CCF は上から下に処理され、この順番を意識して CCF を作成しなければならない。

6.2.3 FormType

DESCRIPTION

CCF form type の enumeration（定数）である

- **select** はコンボボックス form type である
- **bool** はチェックボックス form type である
- **text** はテキストフィールド form type である
- **integer** は整数（10 進）form type である
- **float** は浮動小数点（10 進）form type である

- **hex_integer** は整数（16 進） form type である
- **expression** は Expression form type である。Expression 参照

6.2.4 ConfigurationSet

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **ConfigurationSet** (任意).
- **Configuration** (必須): Configuration 参照
- **name** (必須; type *string*): このオブジェクトの名前
- (**DefineSet**（任意）: DefineSet 参照）（訳者注：この行はオリジナルの英語版には存在しない）

6.2.5 Configuration

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **Item** (任意): Item 参照
- **Expression** (任意): Expression 参照
- **name** (必須; type *string*): このオブジェクトの名前
- **formType** (必須; type *FormType*) この configuration オブジェクトが使う form の種類
- **min** (任意; type *int*): **formType** が integer もしくは hex_integer の場合、この configuration の最小値
- **max** (任意; type *int*): **formType** が integer もしくは hex_integer の場合、この configuration の最大値
- **path** (任意; type *string*): **formType** に対応する結果の configuration のターゲットを記述する XPath 式
- **uri** (optional; type *string*): **path** が適用される XML ファイル

6.2.6 Item

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **Configuration** (任意): Configuration 参照
- **key** (必須; type *string*): この設定項目の名前

- **value** (必須; type *string*): この設定項目の値

6.2.7 Expression

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **description** (必須; type *string*): この expression の説明
- **Exp** (必須): Exp 参照

6.2.8 (DefineSet)（訳者注: この行はオリジナルの英語版には存在しない）

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **Def** (mandatory;): Def 参照

6.2.9 Def（訳者注: この項目はオリジナルの英語版では 6.2.8 である）

DESCRIPTION

次のオブジェクトもしくは属性を持つ。

- **name** (必須; type *string*): このオブジェクトの名前
- **path** (必須; type *string*): **name** が示す XPath 式
- **uri** (任意; type *string*): **path** が適用される XML ファイル

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

次の例は入れ子構造の構成を表している。システムクロック周波数の選択によって、異なるプロセッサのクロック周波数（*MasterComponent*）の選択肢が表示、設定される。

```
<?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 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>
  </Configuration>
</ConfigurationSet>
```

7. FAQ

Q: なぜ SHIM ワーキンググループはマルチ・メニーコアアーキテクチャ/デバイスの表現に XML スキーマを用いているのか？

A: XML スキーマを用いることにしたのは、XML データバインディングと呼ばれる技術を用いることができるためである。データバインディングにより、XML の要素や属性としてではなくデータオブジェクトとして、SHIM XML データを扱うクラスライブラリを生成することができる。例えば、SHIM XML から *MasterComponent* と呼ばれる C++や Java のオブジェクトを生成し、まさに C++/Java オブジェクトのメンバ変数を参照したり読んだりするように *MasterComponent* 要素の属性にアクセスすることができるようになる。現在 XML データバインディングを扱う多くのオープンソースツールが利用されている。データバインディング技術以外にも、SAX/DOM のレガシな XML ライブラリを通して SHIM XML にアクセスすることは可能である。しかし基本的に、XML をファイルとして読み、XML の各要素や属性に対して繰り返し操作することは非常に退屈なプログラミングであり、またコードも与えられた XML 構造に依存して移植性も下がってしまう。XML データバインディングを使えば、SHIM のスペックが更新されても、高い可能性で既存ツールのコードがそのまま動作するだろう。*SHIM Concepts*, *XML* も参照されたい。

Q: SHIM と IP_XACT の違いは何か？

A: IP_XACT は基本的に「設計」言語である。それは主に複数のハードウェア IP コンポーネントが現実にどのように電気的に接続されているかを表現することに重点を置いている。それに対し、SHIM は「記述」言語であり、主にソフトウェア開発ツールが必要とするハードウェア属性を記述することだけに重点を置いている。したがって、SHIM は内部接続やバスの種類を直接的には記述していない。SHIM は master/slave IP コンポーネントと slave コンポーネントを階層的に記述するが、どのように(例えば古典的なバス、クロスバー、NoC(Network on Chip)のいずれを用いて)それらが繋がっているかに対する特別な記載はない。SHIM では、IP コンポーネントは、レイテンシのようなメモリアクセス属性、FIFO レジスタのような master 間通信、それにクロック、命令セット(ABI)、キャッシュサイズとタイプといった基本的なプロセッサ属性を記述するために並べられているのであり、これら全ての要素は設定値を見積もるためにソフトウェアツールに用いられることになる。IP-XACT との連携の可能性については *Componentization of SHIM XML* も参照されたい。

Q: OpenMPI HWloc と SHIM との比較についてどう考えればよいのか？

A: HWloc[4]は静的なチップ内 IP 構成を扱うという点において SHIM と少し似ている。しかし、いくつかの大きな相違点がある。相違点の一つは、HWloc が、OS ランタイムのインターフェースを通して取得した情報に依存し、その情報は HWloc が定義した標準 API を通して提供されることであるように思われる。SHIM は主にシステムを動作させることなく使われることを想定しており、SHIM の情報は OS コンフィギュレーションを構築することに使われる。このことにより、OS インターフェースを通して HWloc が得る情報を作り出すために SHIM が使われていることになる。すなわち、HWloc はソフトウェア目線からのハードウェア基本記述ではなく、ハードウェアトポロジを読み取るためのランタイム API の標準化に重点を置いていることになる。SHIM と異なり、HWloc はハードウェアの性能情報を扱っているようではない。HWloc は HW トポロジに重点を置いているようであり、例えば、HWloc ライブラリを用いるアプリケーションが、ある特定のコアにスレッド/プ

[4] http://www.open-mpi.org/projects/hwloc/

1758 ロセスを割り当てるため、HWloc が提供する情報を用いることができる。実際これは SHIM.xml の一つの可能なユース
1759 ケースではあるが、むしろ我々は性能見積もり、ツールコンフィギュレーション、ハードウェアモデリングのようなツールユースケ
1760 ースに重点を置いている。HWloc はコマンドもしくはテキストを用いて仮想ハードウェアを記述することができるようだが、そ
1761 の可能性は限られているようだ。

1762 上記のように述べたが、*SHIM* 仕様は *XML* スキーマによって定義されており、スキーマコンパイラを通し、*C/C++*ライブラリ
1763 も生成できる。多くのツールの助けを借りて、*XML* パーサや *SHIM XML* ファイルを格納するファイルシステムを必要とする
1764 ことなく、そのようなライブラリの軽量実装を提供することは難しくない。これによってターゲットランタイムシステムから *SHIM*
1765 が利用可能となる。方法論については言及しないが、*HWloc* との連携も確かに可能ではある。

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. Satyadhyan 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