CCI Updates: Inspection

Lukas Jünger – MachineWare GmbH

Peter de Jager - Intel Corporation

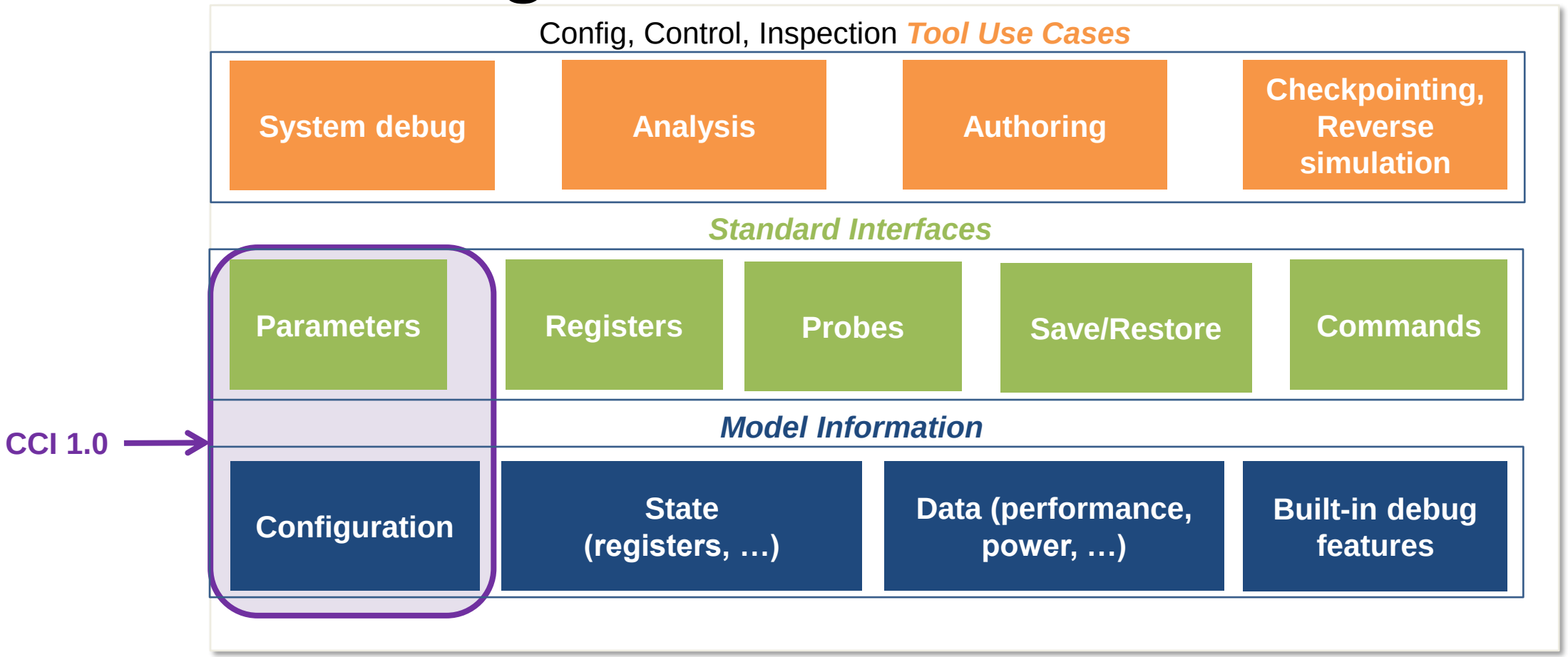
Copyright Permission

- A non-exclusive, irrevocable, royalty-free copyright permission is granted by **Intel Corporation** and **MachineWare GmbH** to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC CCI** standard, and in derivative works based on the standard.

Outline

- Background & Motivation
- History
- Definitions
- Inspection Proposal
- Demo
- Questions/Discussion

Background & Motivation



Goal: Standardizing interfaces between models and tools

History

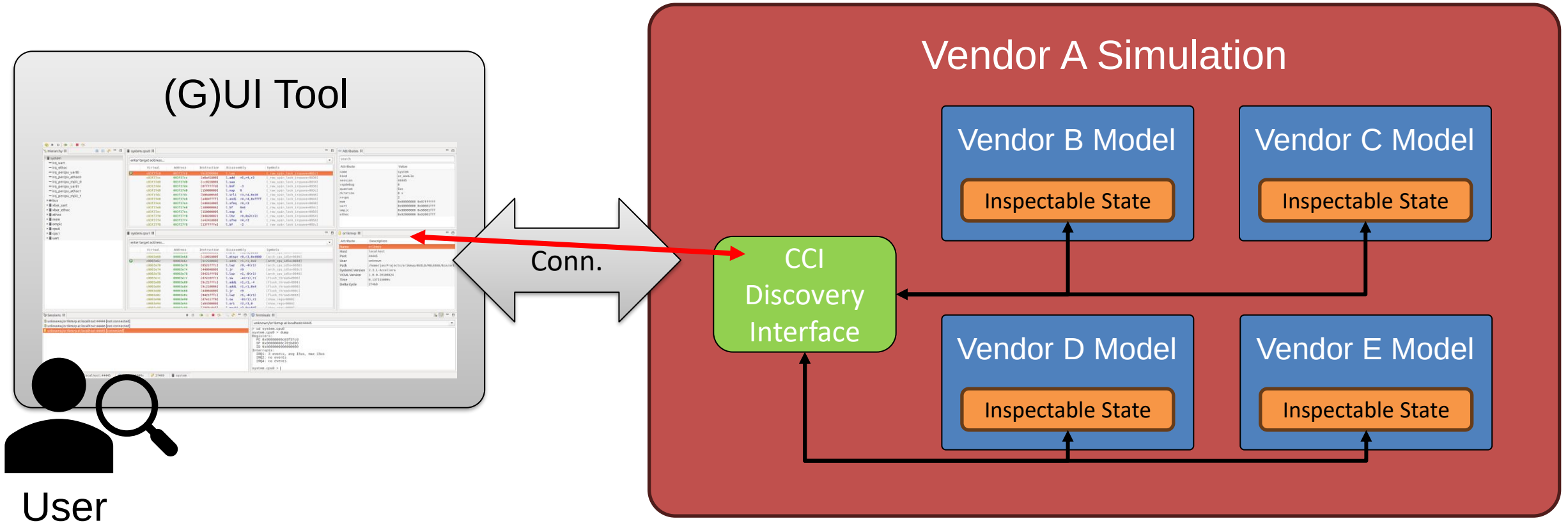
- *…. Even earlier….*
- 2017 SCED: '*Standardization Around Registers',*
  Mark Burton (GreenSocs), Jerome Cornet (ST), Ola Dahl (Ericsson), Philipp Hartmann (Intel)
  - Introduces the uses-cases, difficulties etc. Highlights that it is not a modeling standard!
    Tool access (inspection) is orthogonal to modeling functional behavior

- 2019 SCED: '*Re-Envisioning CCI Inspection'*,
  Bill Bunton & Philipp Hartmann (Intel), Michael Lebert & Ola Dahl (Ericsson)
  - What should be inspected, what is inspection etc. High-level proposal on inspection interfaces, portals etc.
- *… gap … (no active CCI-WG, pandemic etc.)*
- 2022 CCI-WG revamped
- 2023 SCED: 'SystemC CCI: What's new? What's next?'
  - a.o. mentions the memory-inspection proposal of NXP

Background & Motivation

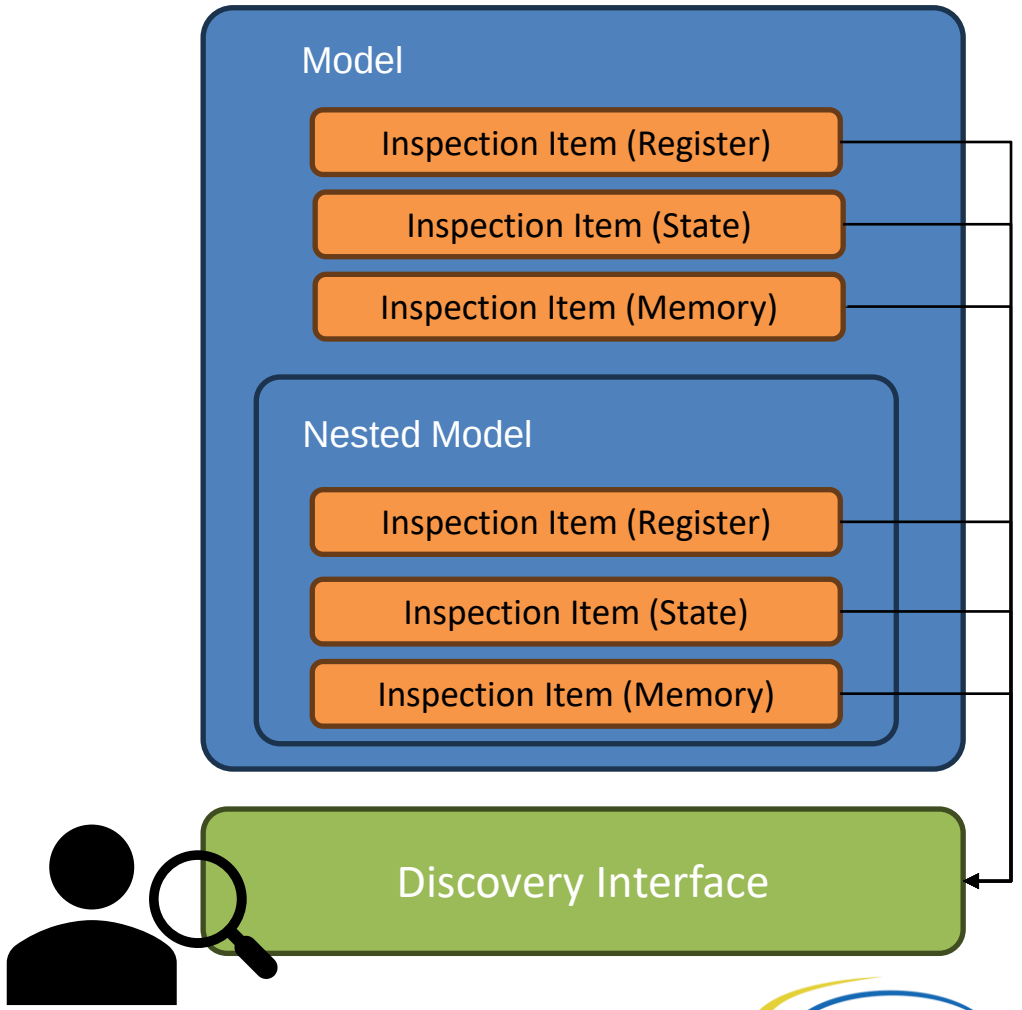
- VPs are combination of models from different sources
- Users want to configure, control and inspect the VP and its models
- Integrators need to provide means to do so
  - Need standardized interfaces on the models
  - How to handle different frameworks (external IP) that cannot be adapted?
  - How to handle legacy models?
- It must be ‘easy’ to use the proposed CCI inspection interfaces
  - Otherwise risk of no adoption

CCI: Inspection Scenario



Definitions

- Inspection
  - Inspect (peek), modify (poke), notification (callback), metadata (name, hierarchy, description?)
  - Metadata should be minimalized since that can also be provided using other means (IP-XACT?) if name/hierarchy can be matched.
- Inspection items
  - Memory, registers (memory-mapped and internal), model variables, simulation-only variables etc.
- Hierarchy
  - An item is located in a modelling-hierarchy which *may or may not* be identical/coincide with the implemented (SystemC) hierarchy
  - Inspection hierarchy is the virtual view on the VP as meant for the user
  - Inspection hierarchy should ideally be identical to user-documentation of the system represented in the VP
- Discovery (previously “portal”)
  - Maintain registered items and provide lookup-mechanism based on metadata (e.g. name)



Inspection Proposal

- Header-only, PoC implementation provided by CCI-WG
- Can be used independently from SystemC (i.e. does not use SystemC code/types)

- Item interface: `cci::inspection::item_if`
  - Defines (pure) virtual methods for name, hierarchy, peek/poke, type (REG, REG_BANK, MEM, Other) and capabilities (R/RW, …)
    - Peek/poke use byte arrays, offset and size
  - No callbacks (yet)
  - Meant for inspection items of any type, within a SystemC module, in external C++ libraries, … (hybrid simulations)
    - Inspection target does not need to be in address map
  - Hierarchy may be virtual (i.e. not corresponding to physical or SystemC hierarchy)
  - Type & capabilities are to be used in tool (display, access, …) to prevent tool using methods that will fail (return false)

- Discovery interface: `cci::inspection::discovery_if`
  - Defines (pure) virtual methods for (de)registering *items* at a hierarchical location and lookup mechanism (all items, items @ hierarchy level)

- User entry point: `discovery_if& get_discovery_if()`

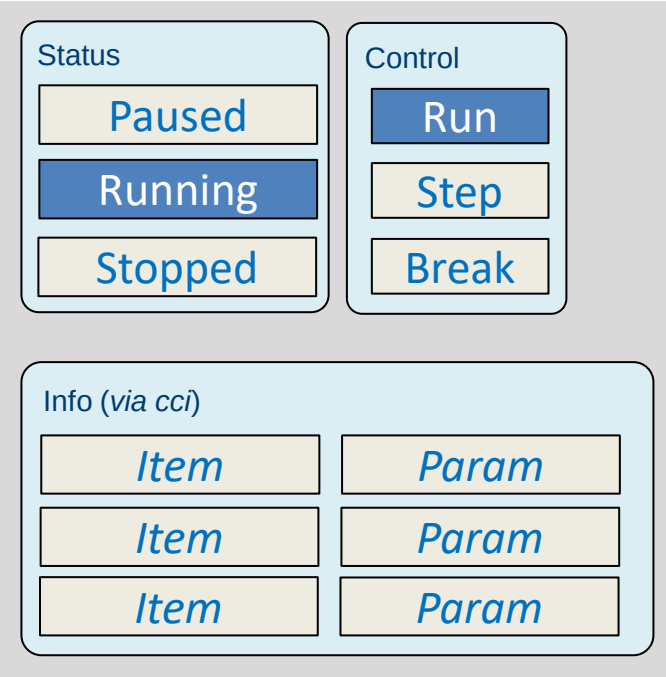
Inspection Proposal: Current Status

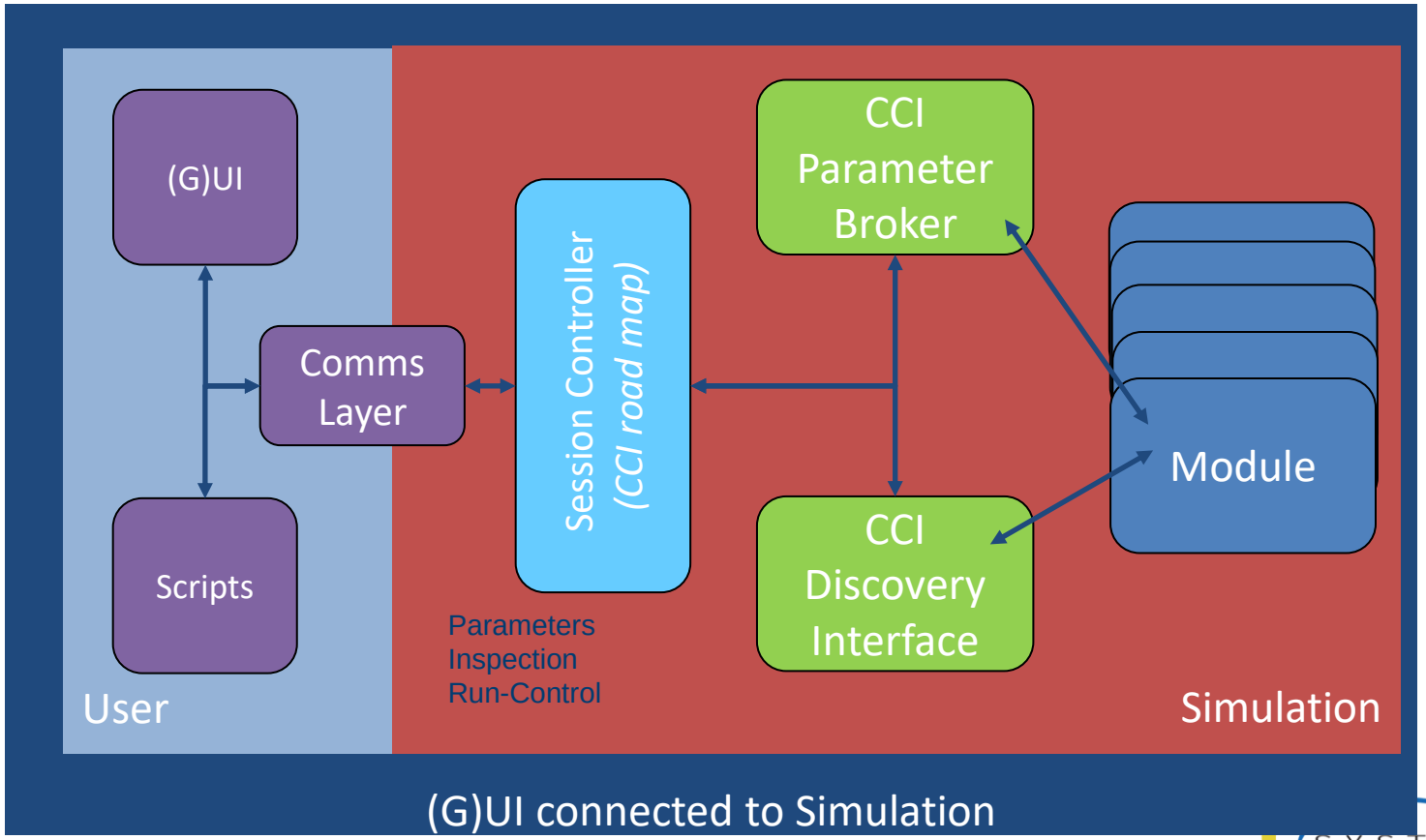
- What's inside?
  - CCI Inspection header: ”standard to be” header for CCI Inspection, C++11 compatible, < 100 LOC
  - PoC CCI Inspection library: PoC implementation of discovery interface
  - Example inspection items and adapters
- What’s next?
  - Further alignment on inspection item and discovery interface (callbacks, hierarchy, …)
  - CCI Inspection Standard draft

- Example inspection items:
  - Direct Inheritance
  - Custom Register Class Adapter
  - SystemC TLM-2.0 Debug Interface Adapter
  - VCML Register Adapter

Inspection Proposal: Example Scenario

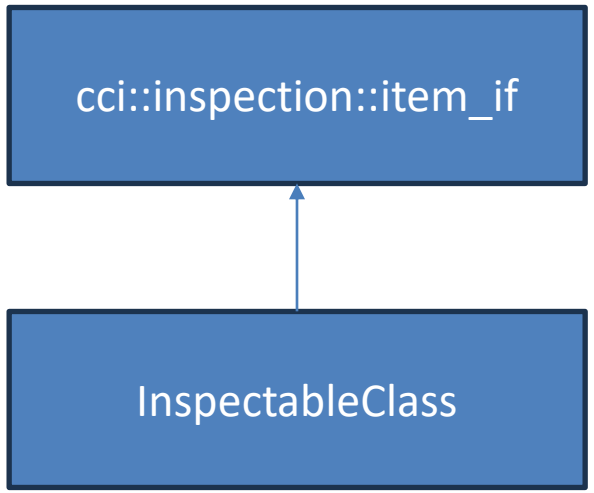
- User scenario: (G)UI for controlling/inspecting simulation
  - Based on MachineWare VCML to enable ViPER, PyVP use with CCI configuration and inspection interface





Scenario: Direct Inheritance

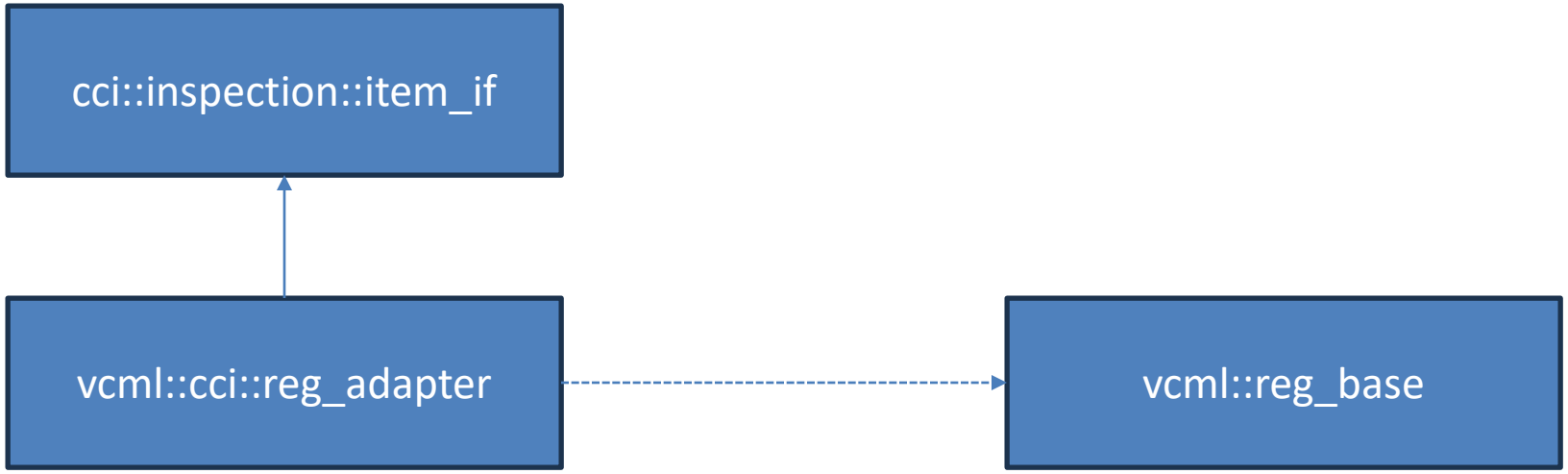
- Inspectable class inheriting from cci_inspection::item_if to give access to internal state
  - InspectableClass implements cci_inspection::item_if directly
- Simple, but need to be able to change InspectableClass (e.g. Register)



Demo

Scenario: VCML Register Adapter

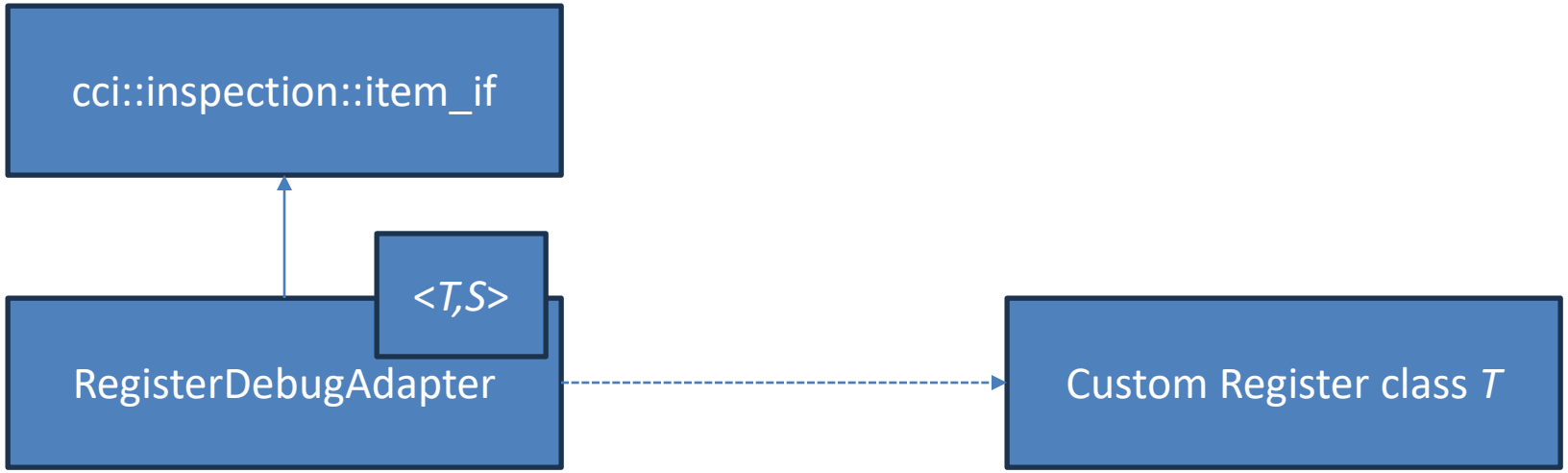
- Adapters inheriting from cci_inspection::item_if to interface with VCML registers
  - class reg_adapter : public ::cci::inspection::item_if
  - reg_adapter has reference to VCML register implementation to access value
- VCML registers are enumerated and registered to the discovery interface at the start of simulation
  - This could be replaced by a way of providing metadata for item registration, e.g. to register only specific registers



Demo

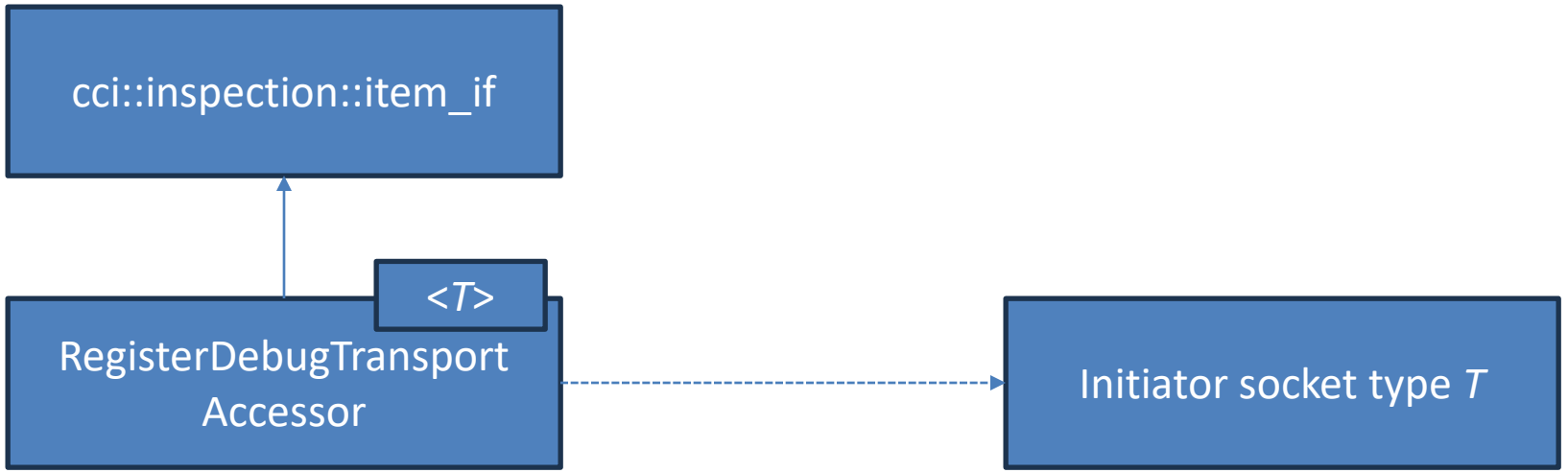
Scenario: Custom Register Adapter

- Adapters inheriting from cci_inspection::item_if to interface with classes that cannot be modified
  - template <typename T, typename StorageType> class RegisterDebugAdapter : public cci::inspection::item_if : accesses register class T using StorageType (e.g. uint32_t)
  - Requires peek/poke like-functionality on class T



Scenario: TLM2.0 Debug Interface Adapter

- Adapters inheriting from cci_inspection::item_if to interface with classes that cannot be modified
  - template <typename T> class RegisterDebugTransportAccessor : public cci::inspection::item_if : accesses (part of) item (@ address, size bytes) via tlm-2 debug transport
    - template <typename T> class RegisterDebugTransportAccessor : public cci::inspection::item_if : accesses item via TLM-2 debug transport
    - Useful if you want to provide access to certain locations as a register/memory without exposing details about the underlying model(s) in your simulation.
    - Only requires (filtered) memory-map of your system
      - Could be generated via IP-XACT (out of scope)



Additional Concepts Demonstrated

- Metadata provider
  - DebugTransportGenerator module
  - Connects to target-socket of VP-system
  - Can read CSV-file on startup that defines hierarchy, name, memory location and size of the items
  - Creates the RegisterDebugTransportAccessor items and registers these to portal

- Runtime access
  - DebugRuntime module
  - Can access portal and reads/modifies items @ runtime

- Session controller
  - Modified version of MachineWare VCML to enable ViPER use with CCI parameters and inspection interface

Demo

Call To Action

- Summary
  - Goal: Standard interface for inspection of model state, memories, registers, …
  - C++11 header-only inspection interface ready for feedback
  - PoC and several examples already done, ready for testing

- **If you find this interesting**
  - **Join the CCI-WG call (every 2nd week Tuesday evening)**
  - **Provide your feedback on the CCI mailing list**

- Questions?