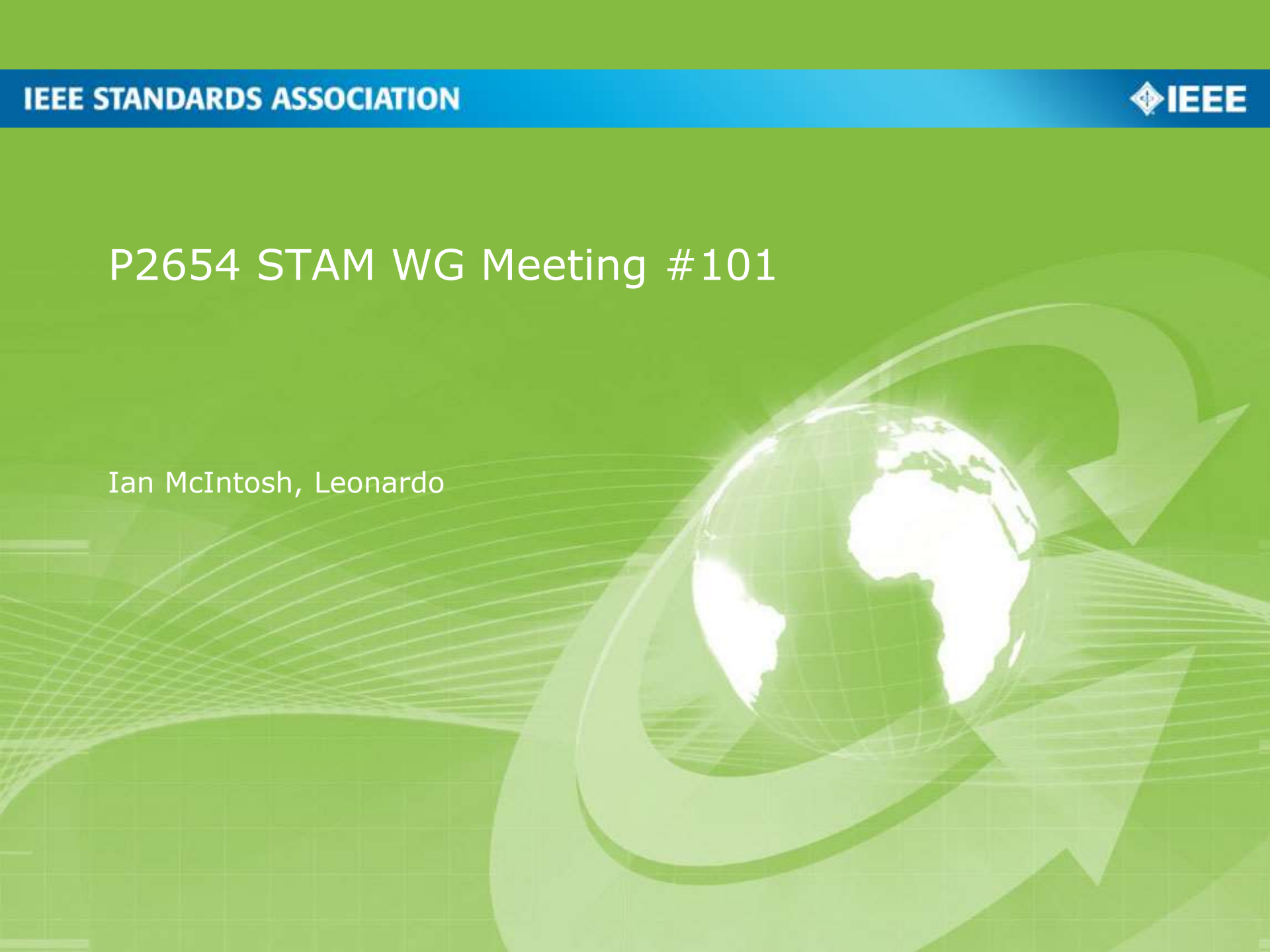
IEEE STANDARDS ASSOCIATION

IEEE

P2654 STAM WG Meeting #101

Ian McIntosh, Leonardo



Compliance with IEEE Standards Policies and Procedures

Subclause 5.2.1 of the *IEEE-SA Standards Board Bylaws* states, "While participating in IEEE standards development activities, all participants…shall act in accordance with all applicable laws (nation-based and international), the IEEE Code of Ethics, and with IEEE Standards policies and procedures."

The contributor acknowledges and accepts that this contribution is subject to

- The IEEE Standards copyright policy as stated in the *IEEE-SA Standards Board Bylaws*, section 7, http://standards.ieee.org/develop/policies/bylaws/sect6-7.html#7, and the *IEEE-SA Standards Board Operations Manual*, section 6.1, http://standards.ieee.org/develop/policies/opman/sect6.html
- The IEEE Standards patent policy as stated in the *IEEE-SA Standards Board Bylaws*, section 6, http://standards.ieee.org/guides/bylaws/sect6-7.html#6, and the *IEEE-SA Standards Board Operations Manual*, section 6.3, http://standards.ieee.org/develop/policies/opman/sect6.html

IEEE P2654
System Test Access Management
Ian McIntosh (chair)

Working Group Meeting #101			
Date: 2021-03-15			
Author(s):			
Name	**Affiliation**	**Phone [optional]**	**Email [optional]**
Ian McIntosh	Leonardo		

2. Agenda

1. Roll Call
2. Agenda
3. IEEE Patent and Copyright Slides
4. Review and approve previous minutes: March 8
5. Review open action items
6. Inter-group Collaboration
7. Discussion Topics:
    a. Primitives: What is needed for a generic group? – Continue from CUSTOM
8. Any other business
9. Key Takeaways from today's meeting
10. Glossary terms from this meeting
11. Schedule next meeting
12. Topic for next meeting
13. Reminders
14. List new action items
15. Adjourn

Participants have a duty to inform the IEEE

- **Participants shall inform the IEEE (or cause the IEEE to be informed) of the identity of each holder of any potential Essential Patent Claims of which they are personally aware if the claims are owned or controlled by the participant or the entity the participant is from, employed by, or otherwise represents**

- **Participants should inform the IEEE (or cause the IEEE to be informed) of the identity of any other holders of potential Essential Patent Claims**

Early identification of holders of potential Essential Patent Claims is encouraged

Ways to inform IEEE

- **Cause an LOA to be submitted to the IEEE-SA (patcom@ieee.org); or**

- **Provide the chair of this group with the identity of the holder(s) of any and all such claims as soon as possible; or**

- **Speak up now and respond to this Call for Potentially Essential Patents**

If anyone in this meeting is personally aware of the holder of any patent claims that are potentially essential to implementation of the proposed standard(s) under consideration by this group and that are not already the subject of an Accepted Letter of Assurance, please respond at this time by providing relevant information to the WG Chair

Other guidelines for IEEE WG meetings

- **All IEEE-SA standards meetings shall be conducted in compliance with all applicable laws, including antitrust and competition laws.**
  - **Don't discuss the interpretation, validity, or essentiality of patents/patent claims.**
  - **Don't discuss specific license rates, terms, or conditions.**
    - Relative costs of different technical approaches that include relative costs of patent licensing terms may be discussed in standards development meetings.
      - **Technical considerations remain the primary focus**
  - **Don't discuss or engage in the fixing of product prices, allocation of customers, or division of sales markets.**
  - **Don't discuss the status or substance of ongoing or threatened litigation.**
  - **Don't be silent if inappropriate topics are discussed … do formally object.**

For more details, see *IEEE-SA Standards Board Operations Manual*, clause 5.3.10 and *Antitrust and Competition Policy: What You Need to Know* at http://standards.ieee.org/develop/policies/antitrust.pdf

Patent-related information

The patent policy and the procedures used to execute that policy are documented in the:

- ***IEEE-SA Standards Board Bylaws***
  (http://standards.ieee.org/develop/policies/bylaws/sect6-7.html#6)
- ***IEEE-SA Standards Board Operations Manual***
  (http://standards.ieee.org/develop/policies/opman/sect6.html#6.3)

Material about the patent policy is available at *http://standards.ieee.org/about/sasb/patcom/materials.html*

If you have questions, contact the IEEE-SA Standards Board Patent Committee Administrator at patcom@ieee.org

IEEE SA Copyright Policy

By participating in this activity, you agree to comply with the IEEE Code of Ethics, all applicable laws, and all IEEE policies and procedures including, but not limited to, the IEEE SA Copyright Policy.

- Previously Published material (copyright assertion indicated) shall not be presented/submitted to the Working Group nor incorporated into a Working Group draft unless permission is granted.
- Prior to presentation or submission, you shall notify the Working Group Chair of previously Published material and should assist the Chair in obtaining copyright permission acceptable to IEEE SA.
- For material that is not previously Published, IEEE is automatically granted a license to use any material that is presented or submitted.

IEEE SA Copyright Policy

- The IEEE SA Copyright Policy is described in the IEEE SA Standards Board Bylaws and IEEE SA Standards Board Operations Manual
  - IEEE SA Copyright Policy, see
    Clause 7 of the IEEE SA Standards Board Bylaws
    https://standards.ieee.org/about/policies/bylaws/sect6-7.html#7
    Clause 6.1 of the IEEE SA Standards Board Operations Manual
    https://standards.ieee.org/about/policies/opman/sect6.html
- IEEE SA Copyright Permission
  - https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/other/permissionltrs.zip
- IEEE SA Copyright FAQs
  - http://standards.ieee.org/faqs/copyrights.html/
- IEEE SA Best Practices for IEEE Standards Development
  - http://standards.ieee.org/develop/policies/best_practices_for_ieee_standards_development_051215.pdf
- Distribution of Draft Standards (see 6.1.3 of the SASB Operations Manual)
  - https://standards.ieee.org/about/policies/opman/sect6.html

4. Review and approve minutes

Working Group Meeting #100, March 8

Draft circulated March 8.

Corrections: 4th bullet in 7a: Change ‘Ned’ to ‘Need’.

Attendees:

Ian McIntosh (Leonardo) (chair)
Eric Cormack (DFT Solutions) (joined 11:33)
Terry Duepner (National Instruments)
Heiko Ehrenberg (GOEPEL Electronics)
Brian Erickson (JTAG Technologies)
Peter Horwood (Digital Development Consultants Ltd)
Joel Irby (AMD)
Richard Pistor (Curtiss-Wright)
Jon Stewart (Dell)
Louis Ungar (A.T.E. Solutions)
Brad Van Treuren (VT Enterprises Consulting Services)
Carl Walker (Cisco Systems)

5. Review open action items

Action Item Register:

http://files.sjtag.org/PostStudyGroup/ActionItemRegister.xlsx

Format of action number is

[Meeting#.Action# within that meeting]

No open actions.

6. Inter-group Collaboration

General:

- Nothing known prior to meeting

7. Discussion Topics [1]

7.a Primitives: What is needed for a generic group?

- Previously considered: REGISTER, INSTANCE, CHAIN, LINKER, MODELPOINT
- Email exchange with some P1687.1 members.
- Continue from CUSTOM

General References:

- Definitions from forum: http://forums.sjtag.org/viewforum.php?f=40
- Reference Pack (all previous 2020 material now consolidated): http://files.sjtag.org/P2654WG/P2654_Reference_Pack.pptx

Wrap-up items

8. Any other business
    - March 23 Next NTF session to be held
    - TAAA Workshop CfP circulated by email; P2654 representation?
    - https://github.com/bradfordvt/P2654Model Brad's first demo code
9. Today's Key Takeaways
10. Glossary terms from this meeting
11. Schedule next meeting
    - March 22 (11 AM EDT, 3 PM BDT)
12. Topic for next meeting
    - TBD
13. Reminders
14. List new action items
15. Adjourn

Discussion 1/11/2021

- Programmed vs. un-programmed
  - Board (or assembly, sub-system, system) state may differ depending on its position in a lifecycle:
    - Newly manufactured items are likely to be unprogrammed
    - But could have case where boards are built with pre-programmed parts
    - Items may become programmed part-way through the manufacturing process
    - Items may have "test programming" that is different to "in-service" programming
    - Field returns may not be responsive to any existing programming
      - Unresponsive
      - Corrupted programming
  - Test selection determined on level of programming
  - Programming time may be quite long (detect failure as early in process as possible)

Discussion 1/11/2021

- Programmed vs. un-programmed
  - FPGA programmed with test buses vs. CPU with test buses
  - Programmed determines what transformations are to be used by the model for that configuration
    - may need different configuration/transform for programmed vs. unprogrammed
    - Test data path may end up being different (Ethernet vs. JTAG)
  - Cascade issues
    - P1687.1 use of 1687 AccessLink is insufficient as it describes a single interface where multiple transformations may be required to move from 1687 Network to DPIC
    - P2654 uses a single transform most of the time as it transforms message data
  - There may be implication on portability as unit passed through lifecycle (fuse lock or not)

Discussion 1/11/2021

- Programmed vs. un-programmed
  - Security may impact capability of test access
  - Query based system may give provisions that may not be so wide open for end customer to get access to secret facilities
    - Need some means for host of testing to ask sub-assembly what is available for test and possibly what configuration (ports) need to be used
    - Knowing what process to use to unlock without giving away details (by reference to possibly external file or documentation)
  - One security switch Al Crouch mentioned uses a slow TCK to activate a deep feature
  - Why should user need to know they are using JTAG for the test or how it is configured
    - Remote loading/test facility to configure the firmware
    - Decoupling of test from infrastructure (TFCL)

Discussion 1/11/2021

- Programmed vs. un-programmed
  - Levels of knowledge of a test
    - Component level has some level of information exported to board level tester uses
    - Board level has some level of information exported to sub-assembly level tester uses
    - Third party sub-assemblies presents a different or at least a more complicated perspective on data sharing

Discussion 1/18/2021

- Programmed vs. un-programmed
  - Agreement that we need to be able to cope with both cases
    - Complete “products” may well be programmed
    - But “systems” may well be unprogrammed:
      - During manufacturing stages
      - Due to a fault
    - This may mean different transformations are required
  - Impact on standard
    - How do we get rules from this?
    - What does it mean for the “shape” of the standard? (do we different sections?)
  - Method remain the same, but input conditions may vary between programmed and un-programmed states (UUT becomes a different product between these two)
    - In rack probably has some sort of programmed condition
    - In production may not have any programmed state or just a partial programmed state
    - In system, some boards may be programmed and some may not – especially if a board is broken or corrupted (random failure failure)
    - Access to test program or data may be unavailable at time of testing
    - How much does the standard need to deal with these different cases? Or Is this something the tooling is responsible for resolving and just informative in the standard?
    - Is there a way to provide a minimum set of information or resources required for compliance to the standard

Discussion 1/18/2021

- Programmed vs. un-programmed
  - There needs to be a way to define a minimum set of capability it will provide
  - Opportunity for tooling to have some smartness in dealing with failing cases or different scenarios and conditions of the "system"
  - Current tooling does check for infrastructure of interfaces to determine if a test could be performed
  - Testing of the infrastructure is probably the most important aspect for this standard
  - Dealing with any test failures is really outside of the scope of this standard – resolution to failure in not STAM business
  - But are we not needing to give access to fail data stored on the device?
  - Diagnostics is really an application interpretation of the results domain and not a STAM domain
  - Different potentials for where these tests are being implemented – Factory floor, field failure, factory return, etc.
    - An engineer needs to be able to define what happens at each point
    - Reactive to the result of the test
    - As management we are dealing with the access and not the system operation or the test operation

Discussion 1/18/2021

- Programmed vs. un-programmed
  - Need to define the method we can get rules around
  - Programmed vs. un-programmed is not something that is going to be primary issue for the standard
  - Over specifying will prevent usage
  - Need to consider power configurations impacting available and not available access to circuit elements
    - Same problems as for other types of testing we do
    - May impact the process steps required to provide access and initialization of the UUT between process steps
  - Procedural steps required to enable the test
    - Dependent on where test is being run – impacts portability of a test
    - Dependent on what kind of controller or test equipment is between environments (free running TCK vs. gated TCK, etc.)
  - May need a section on test types in system level testing
    - On-line
    - Off-line
    - These are informative
    - May not be actually a difference other than a different method being used

Discussion 1/18/2021

- Programmed vs. un-programmed
  - Procedural steps required to enable the test
    - Dependent on where test is being run – impacts portability of a test
    - Dependent on what kind of controller or test equipment is between environments (free running TCK vs. gated TCK, etc.)
  - Are "available resources required" necessary to specify by the standard?
    - Impacts what can be executed
    - Impacts what can be diagnosed as the result
    - May just be something for informative section
  - Programmed vs. Un-programmed use of SRAM based
    - May need to access from different kinds of memory
      - Non-volatile
      - RAM
      - ROM in a package for self test
      - Etc.
      - Can encourage designers to do a better effort or DFT

Discussion 1/18/2021

- Programmed vs. un-programmed
  - May need to define terms for:
    - Pre-deployment
    - Post-deployment
  - Observation: points raised today are looking more like informative points and not normative points
  - Rules needed for:
    - Description syntax
    - Description semantics
    - Need to be able to cope with programmed and un-programmed cases
      - Programmed state is usually going to open up more opportunities for access and capability for test (programming may in fact lock things down too)
      - Require different configurations of the model
      - Are we going to support dynamic changes to the configuration (boards plugged and unplugged from a system)?
      - Is the standard needing to support application of multiple tests per configuration or single tests? 1149.1 does not go into the level of multiple tests, only how to achieve a single objective (how to apply the state change of the UUT)
      - 1149.1 describes a single static entity configuration – how much does STAM need to describe as the wider application (static configuration or adaptable)?

Discussion 1/18/2021

- Programmed vs. un-programmed
  - Caution about over prescribing in the standard that would limit what tool providers can do

Discussion 1/25/2021

Rules required (or scope of rules) in standard

- Any questions arising from Jan 18 discussion?
  - None noted
- Rules are probably independent of programming state – universally applicable
  - So what are the rules (normative parts of the standard)?
  - Aims of the rules:
    - Definitive about what needs to be described (aspects)
      - Entities (nodes in the topology tree model)
        - Software Interfaces
          - AccessInterfaces to neighbours (used by client and host interfaces to transport a message to another node)
          - Client Interface
          - Host Interface
          - Transformation Strategy Interface (Interface to transformation algorithm taking in lower level message to upper level messages – i.e. Plugin interface to external software defining the transformation strategy to use for a node)
          - Transparent translation (data coming in does not get inverted on the way out)
            - Needs to be deterministic and repeatable at the leaf (P2929 discovered issue)
        - Dependencies (especially selectors for LINKER type devices)

Discussion 1/25/2021

Rules required (or scope of rules) in standard

- Rules are probably independent of programming state – universally applicable
  - Aims of the rules:
    - Definitive about what needs to be described (aspects)
      - Entities (cont) [Proposed categories to consider by BGVT]
        - Registers (Leaf node)
          - Safe value (similar to what BSDL defines for cells)
          - Size (number of cells making up the register storage)
        - ModelPoint (connection to external tooling for the model)(Leaf node)
        - Chain (e.g., jtag path, 1687 network)
        - Linker (e.g., TAP, BSCAN2, Gateway)
        - Instance (reference to another description file)
        - Root (top level where communication with hardware driver takes place)
        - Custom (user defined or black box)

Discussion 1/25/2021

Rules required (or scope of rules) in standard

- Rules are probably independent of programming state – universally applicable
  - Aims of the rules:
    - Definitive about what needs to be described (aspects)
      - Hardware Interfaces
        - Protocol
        - TCK requirements (e.g., Gated vs. free running vs. specific frequencies)
        - Registers
          - Type: Scan or Digital (parallel)
- Applications will usually be dependent on a given programming state (which tests and other operations will be possible)
  - This probably needs an informative mention

Discussion 2/1/2021

Rules: Use of the Descriptions (possibly extend Descriptions)

- How would we use the descriptions?
  - Depends on user case perspective (ATE Device Test vs. Integrator Test)
  - May not be able to adopt Michele's user cases from P1687.1 for P2654
    - Test Engineer for system does not have concept of vectors and may just be string commands between entities
      - E.g., Mixed Signal Test is example of a non-vector based test
    - System Level Test (SLT) may be requiring very different roles from what SoC Test requires (e.g., programmed vs. un-programmed states)
    - Preserve state (e.g., MBIST) to restore state after test or provide a way to restore to a known good state (reset) – probably out of direct scope of P2654 but provided by standard
    - Bottom line is the roles have to deal with intent of use by the user
  - Are there more than attributes/properties needing description for each node in the model tree?
    - Relationships to other nodes:
      - Dependencies on other nodes for selectors defining behaviour control (Binary, One-hot, N-hot, One-hot w/ no Zero state, N-Hot w/ no Zero state, Table, etc.)
    - Detail of description becomes increasingly necessary the deeper the architecture is provided as the decomposition of the target "system"
    - Should not cross over into other domains from other standards (e.g., BSDL attributes)

Discussion 2/1/2021

Rules: Use of the Descriptions (possibly extend Descriptions)

- How would we use the descriptions?
- Does the standard need to define where the descriptions come from or just what they contain?

Discussion 02/08/2021

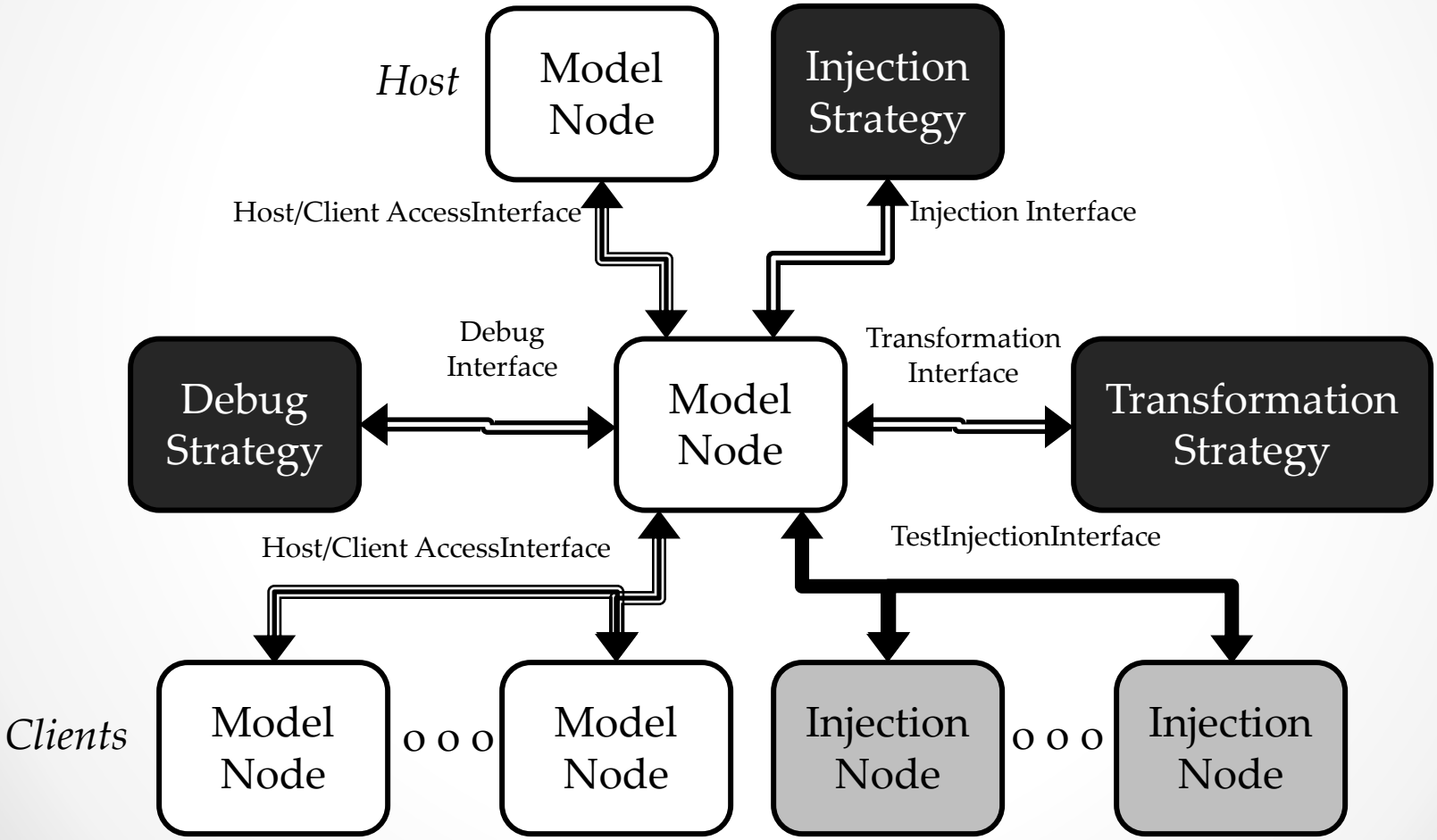
How much transparency do we need at each level?

- To what extent does the boundary of an entity (level or node) need to expose detail of entities that are hierarchically below it?
  - What are the implications of not exposing those details?
- What does the standard need to say about how would we use the descriptions?

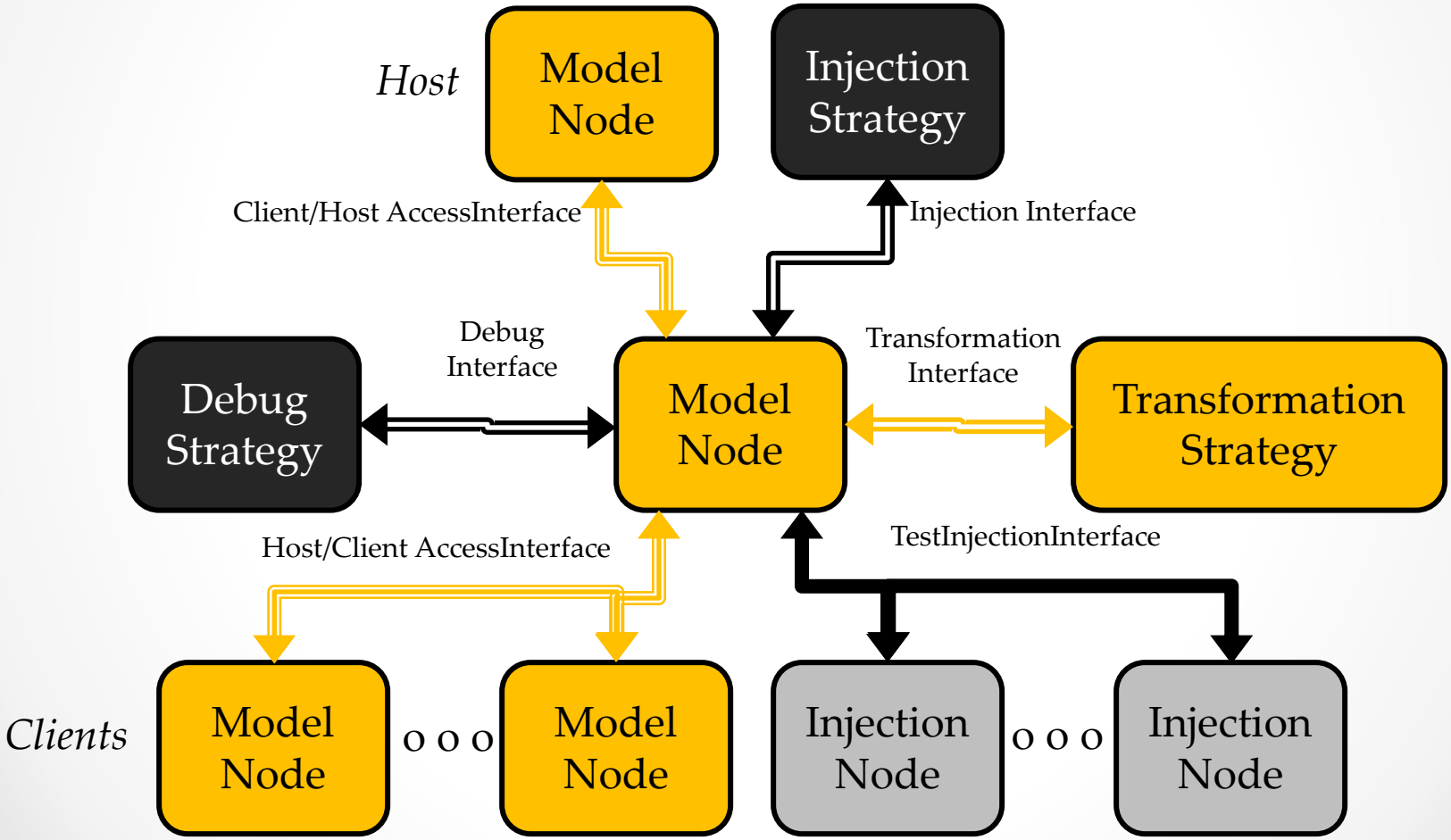
Discussion 02/08/2021

- The following slides (33 – 49) were shared by Bradford G. Van Treuren during the meeting today
- These slides are from the presentation entitled ***P2654/P1687.1 Unified Concepts Analysis***
- These slides are used with permission of VT Enterprises Consulting Services in use for the P2654 Working Group activities
- The slide notes include additional important information about the topic described by the slide

Model Node Interfaces



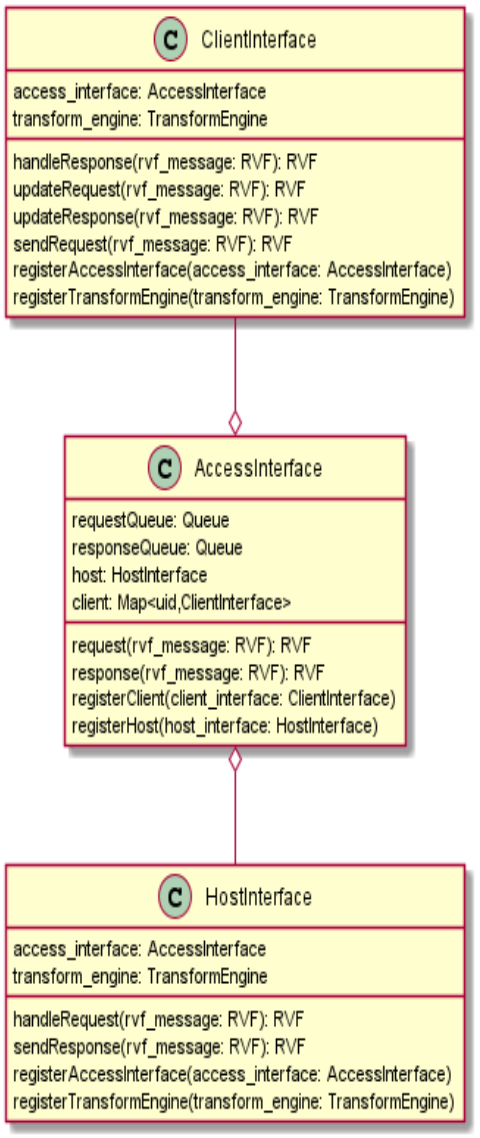
Model Node Interfaces



Transformation Flow

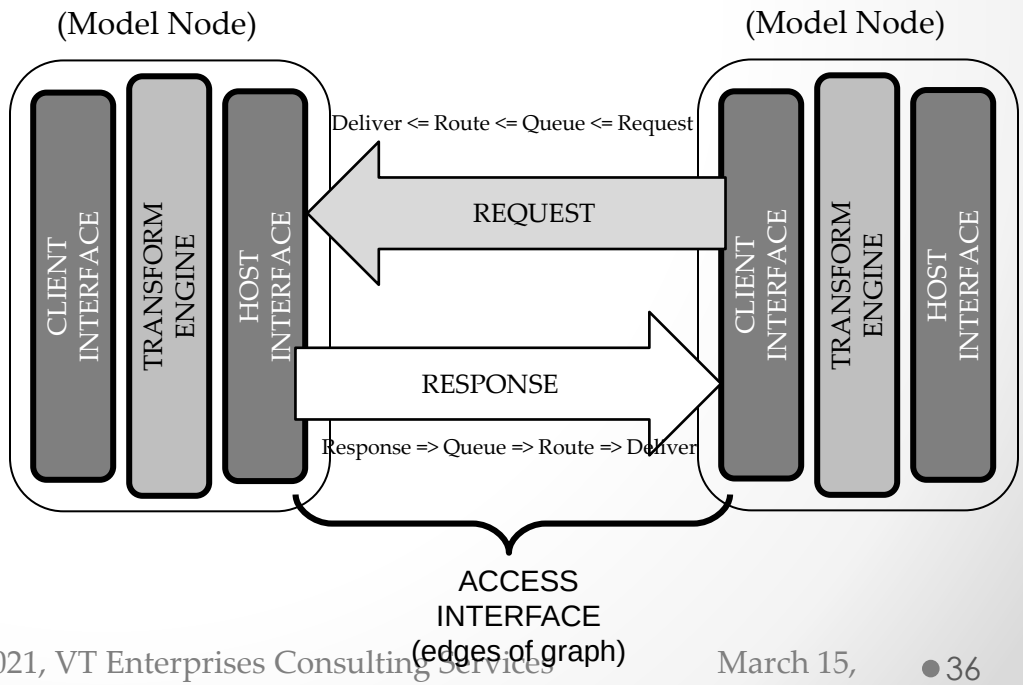
- A Request message is received by the Host Interface and needs to be transformed into a set of Client side messages to the next higher level
- Message is handed off to the Transform Strategy for processing
- Client requests are queued and sent to next level one-by-one
- Upper levels process requests and return responses for each message
- Transform Strategy inverse transforms responses into Host side response messages
- Transform Strategy sends responses to children of the host interface

Simplified AccessInterface and Node Diagram

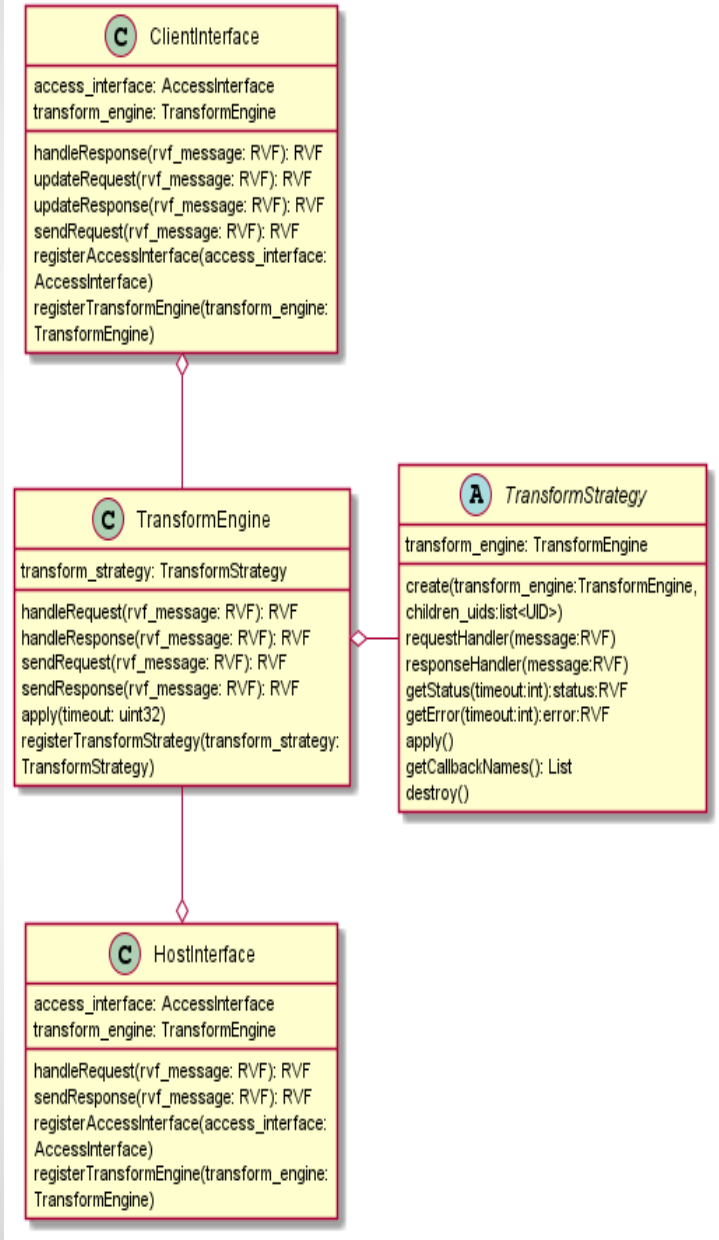


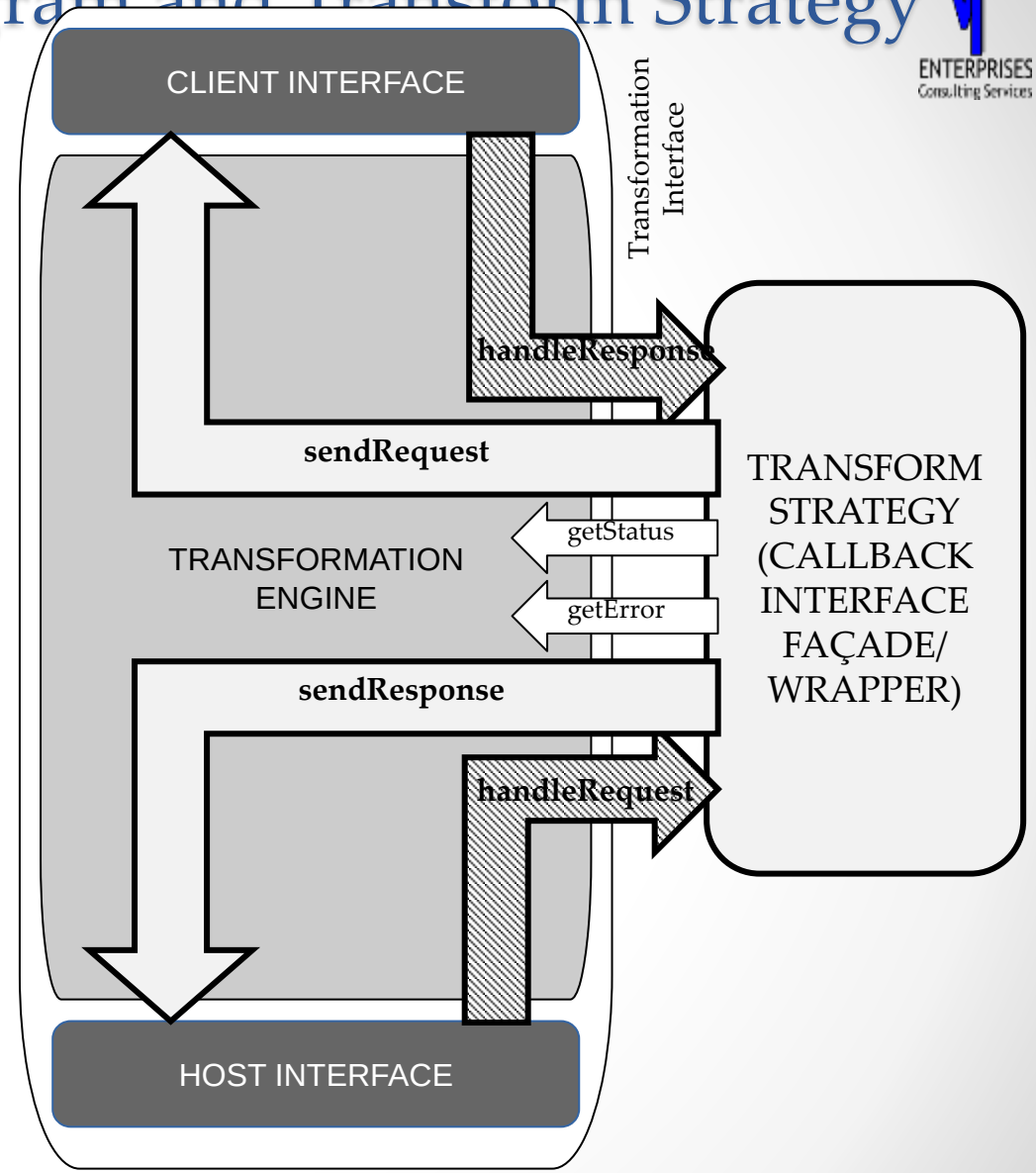
- Abstracts the connections between Client/Host
- Standardizes the communications interface
- Routes messages between Client/Host and Host/Client
- RVF agnostic
- Buffer message groups of same context
- Not transformation mechanism
- Not router to handler callback (Client/Host Interfaces are)
- Not synchronizing agent for model (transform vs. retarget motive)

Transformations in P2654 are nothing more than a specialized Transformation Engine that may or may not be P1687.1 compliant



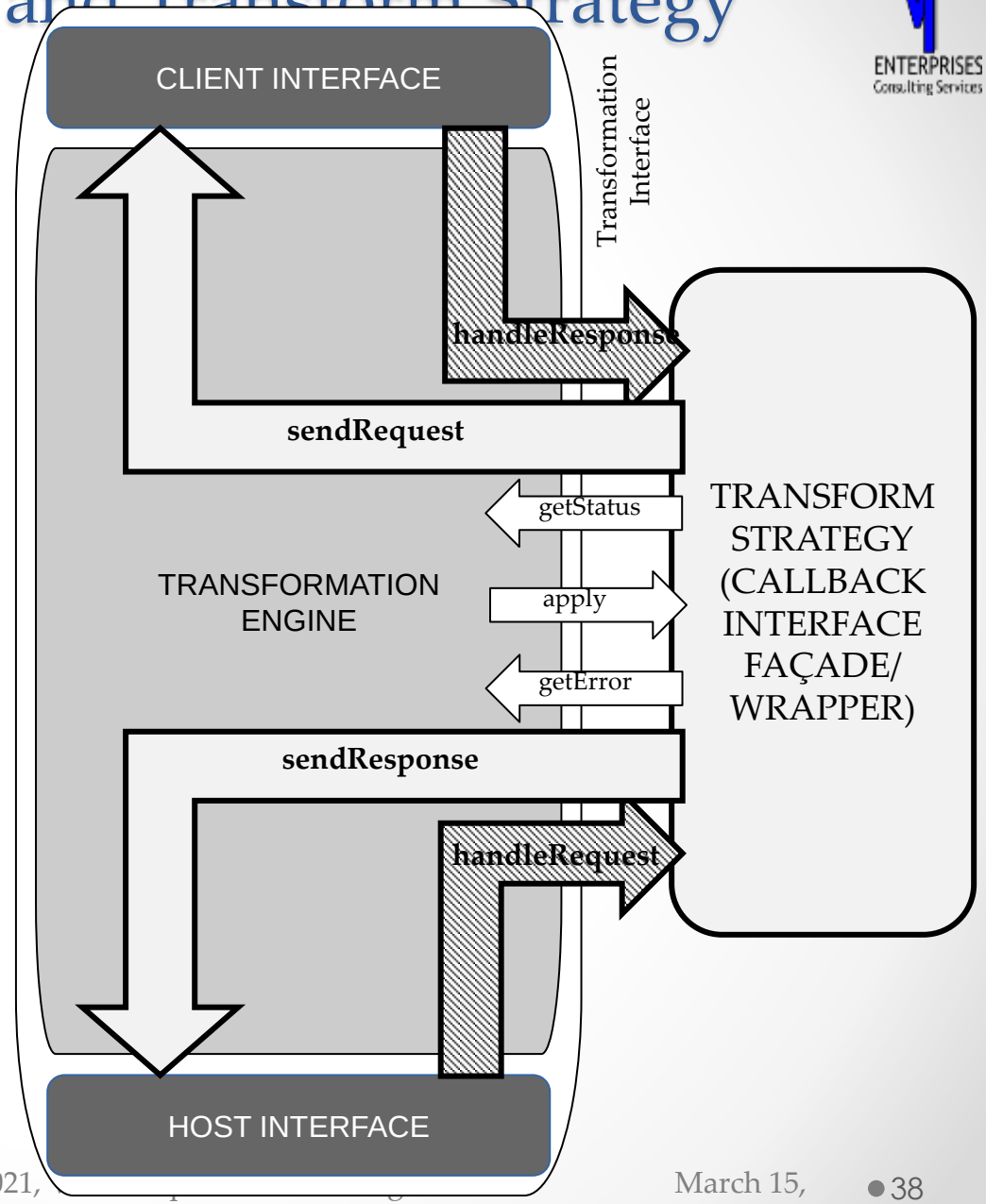
Simplified Node Diagram and Transform Strategy





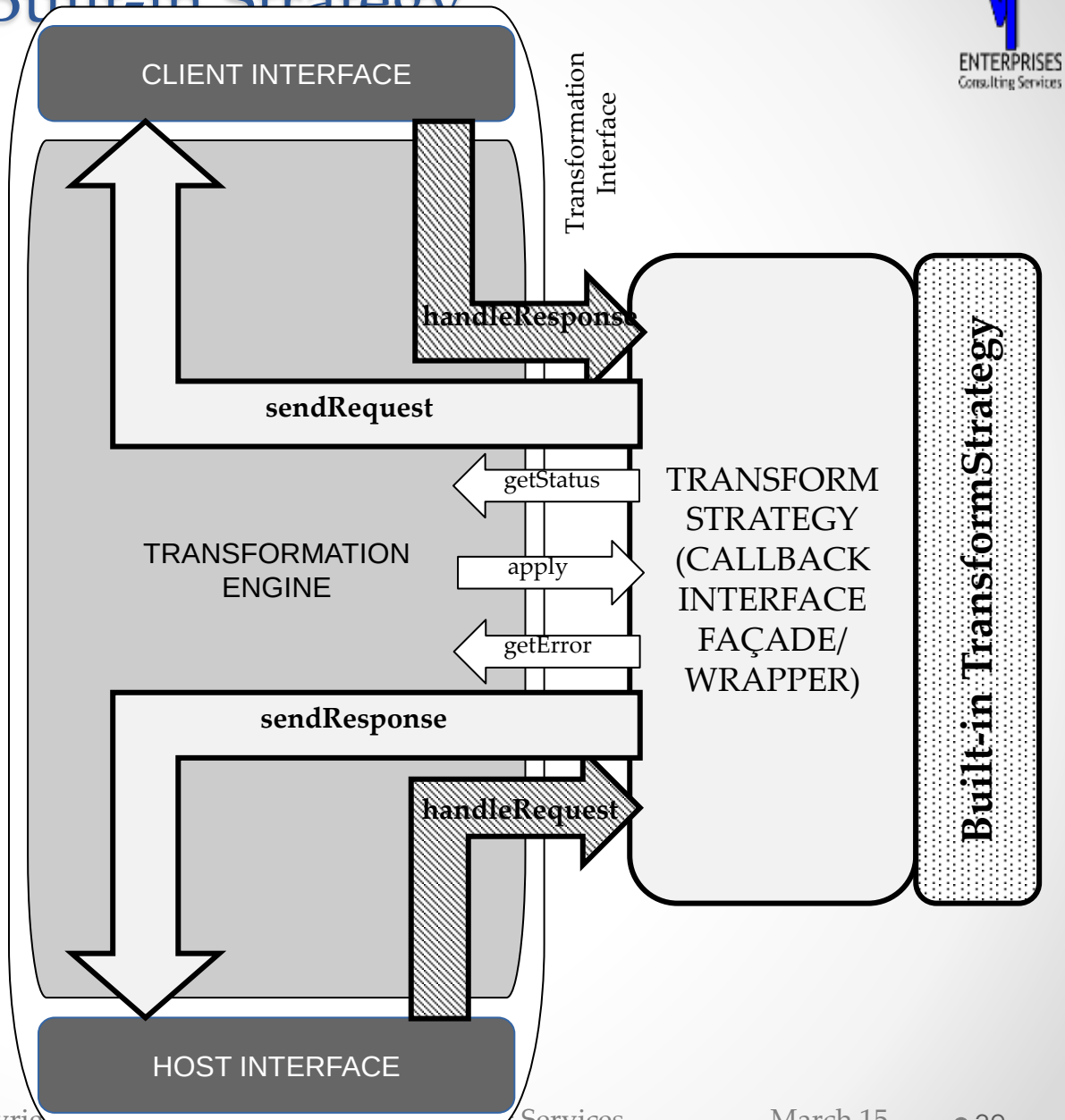
Simplified Node and Transform Strategy

- TransformEngine abstracts the connections between Client/Host and the underlying TransformStrategy
- TransformEngine standardizes the communications interface between Host/Client and TransformStrategy
- RVF agnostic
- TransformEngine is synchronizing agent for model (apply)
- TransformStrategy wrapper for protobuffer callbacks
- TransformStrategy single point of entry for all callbacks (see handlers)
- TransformEngine monitors all messages passed between interfaces (See DebugStrategy)



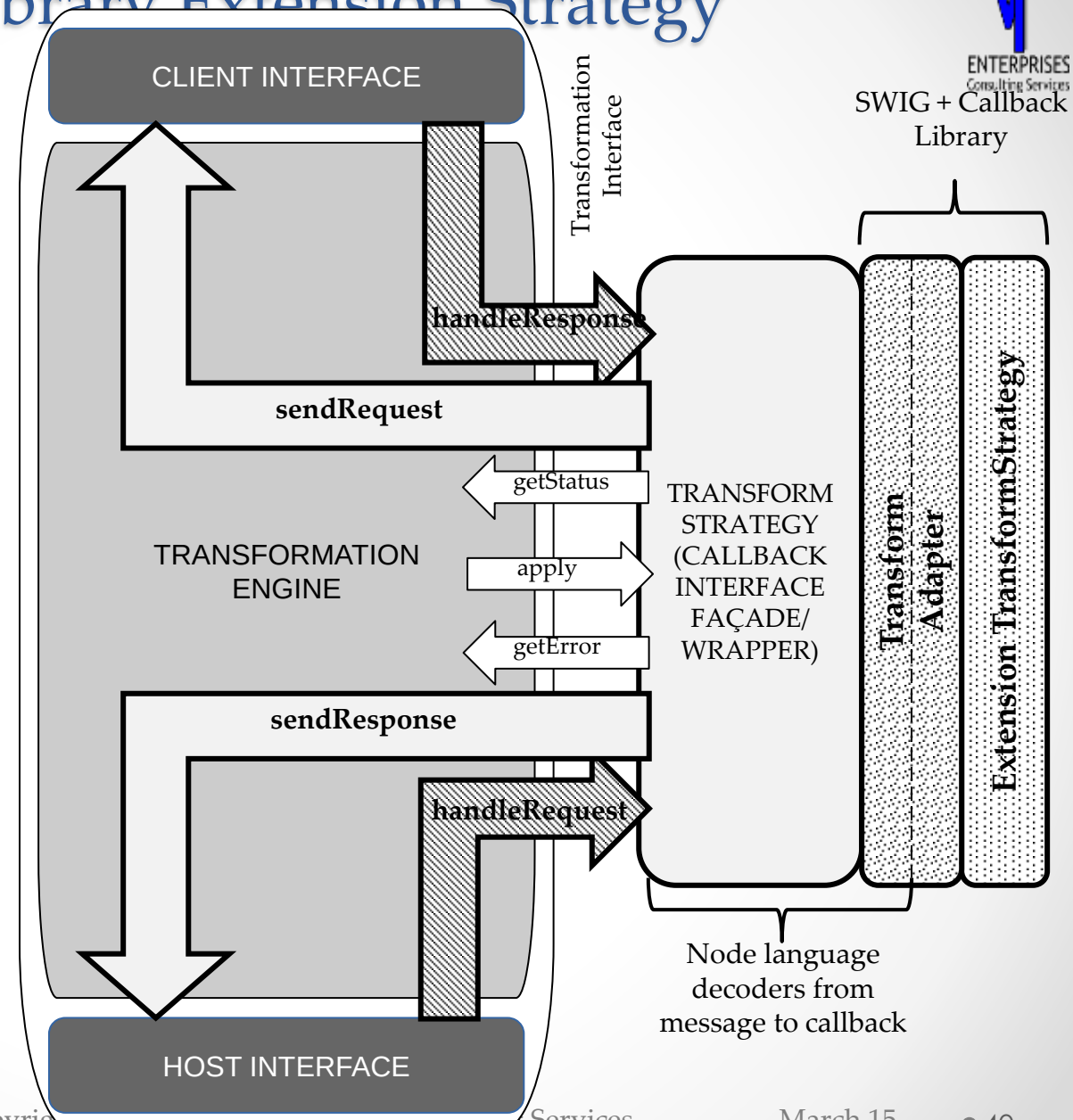
Built-in Strategy

- Built-in implements special "service" functions from protoc compiled in same programming language as system model
- Direct call to callback functions from TransformStrategy
- Used mainly for standardized primitive elements defined by the standard (e.g., SIB, LINKER, MUX)

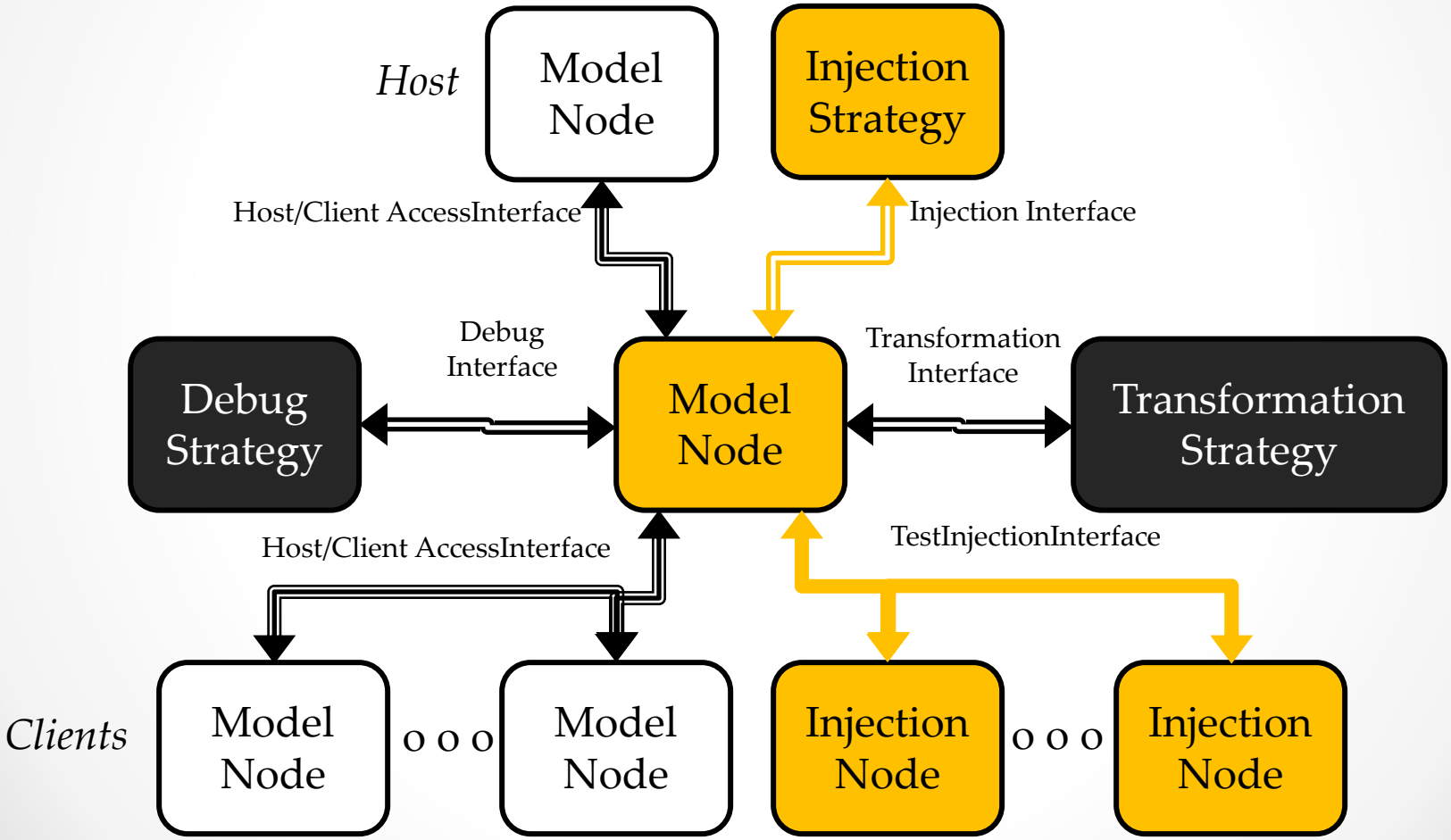


C/C++ Library Extension Strategy

- Leverages protobuffer programming language to support model language and C/C++
- Direct call to callback functions from TransformStrategy
- Service functions from protoc compile in C or C++ code
- SWIG generated or hand crafted to adapt model code to C/C++ callbacks



Model Node Interfaces

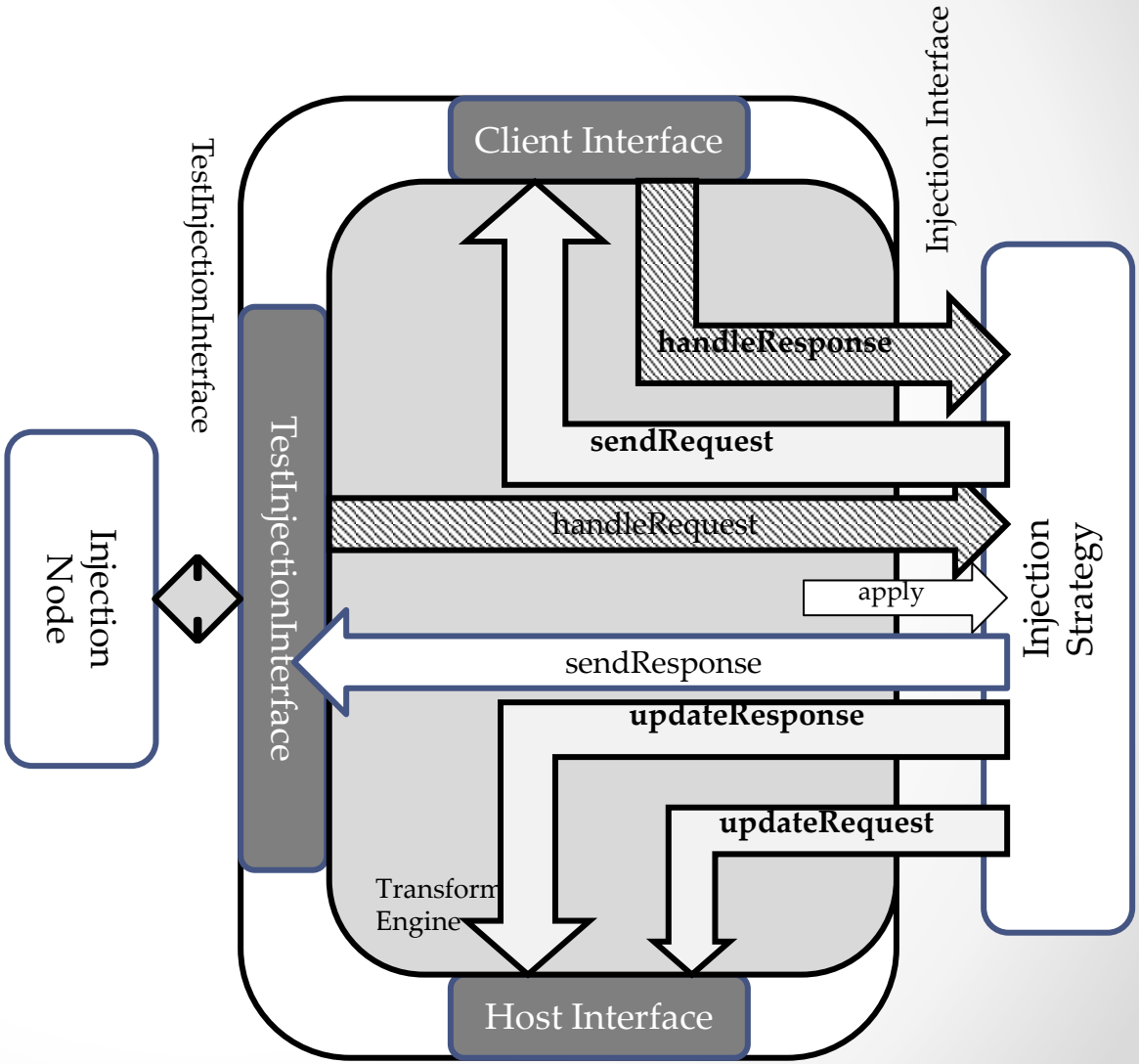


Injection Flow

- Injection Node represents data injection sources (e.g., System Application, SVF Player, PDL Player)
- Parallel source of messages for Model Node
- Injection Node passes messages in the Client Interface context
- Injection Strategy responsible for decomposing message into update messages for children of the Model Node to synchronize state changes in the hardware based on injection messages with the software model
- Transformation Engine is responsible for routing injected Request messages and Client Response messages to the Injection Strategy when processing injected data. Otherwise, the Transformation Engine routes messages to and from the Transformation Strategy

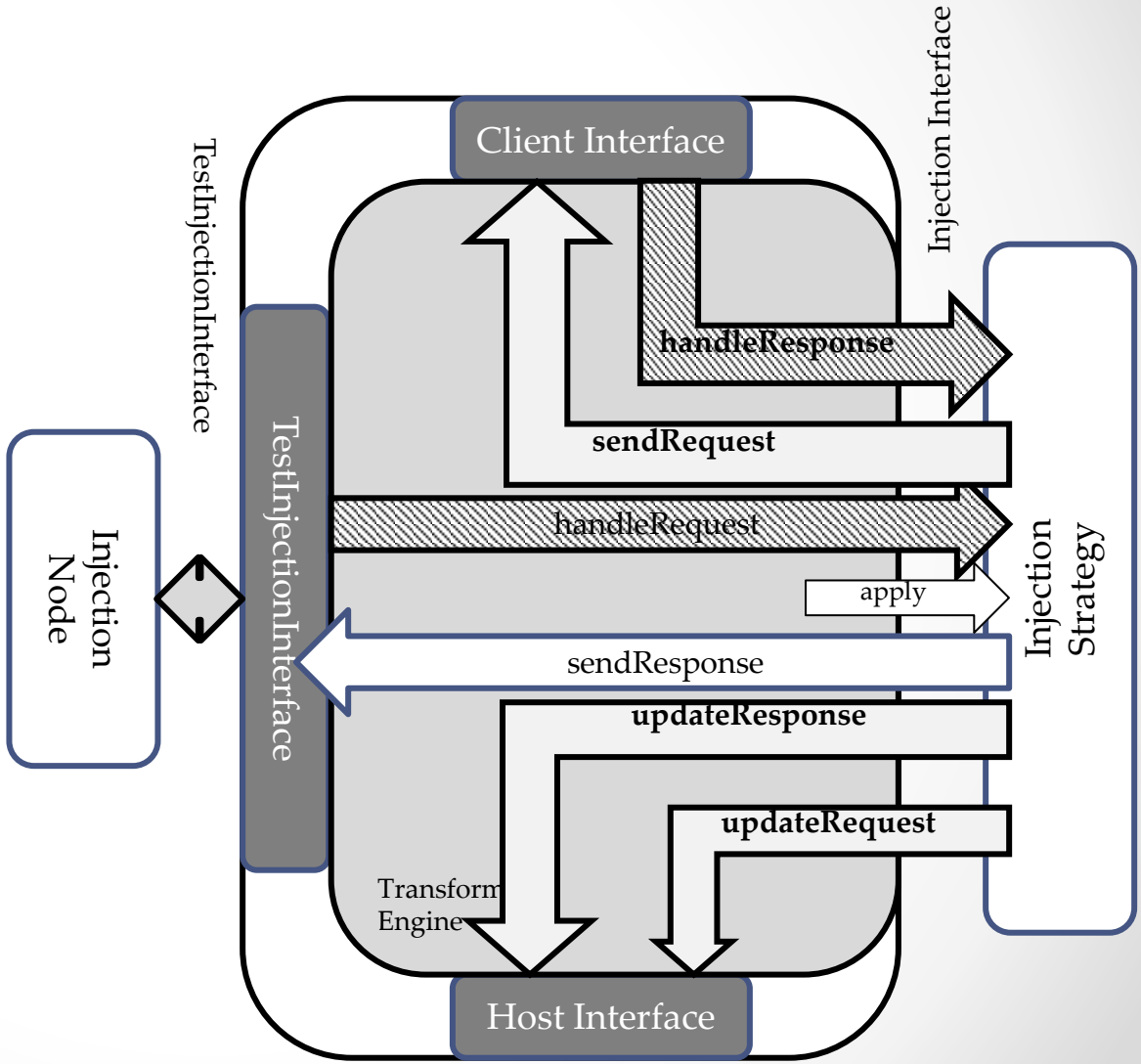
Stimulus Injection Process

- Injection Node sends request for Client Interface context to Injection Strategy
- Injection Strategy verifies and forwards message to Client Interface via TransformEngine
- Injection Strategy transforms request into updateRequests for children of the Host Interface
- Injection Strategy handles response from Client Interface
- Injection Strategy passed Client response data to Injection Node



Stimulus Injection Process

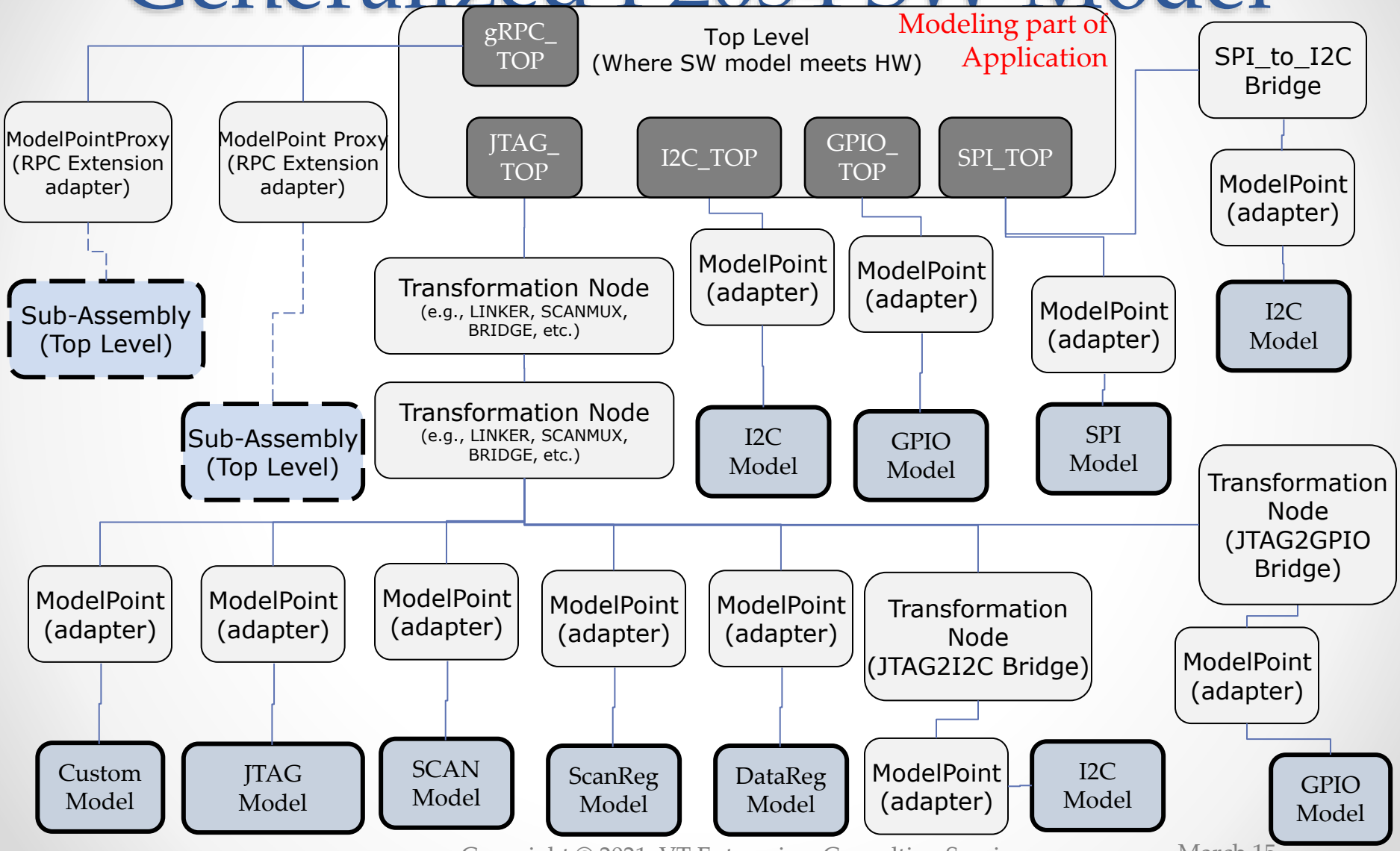
- Injection Strategy send an inverse transformed message for the children of Host Interface to update child states
- Injection Strategy follows the same implementation/extension mechanism used for Transform Strategy



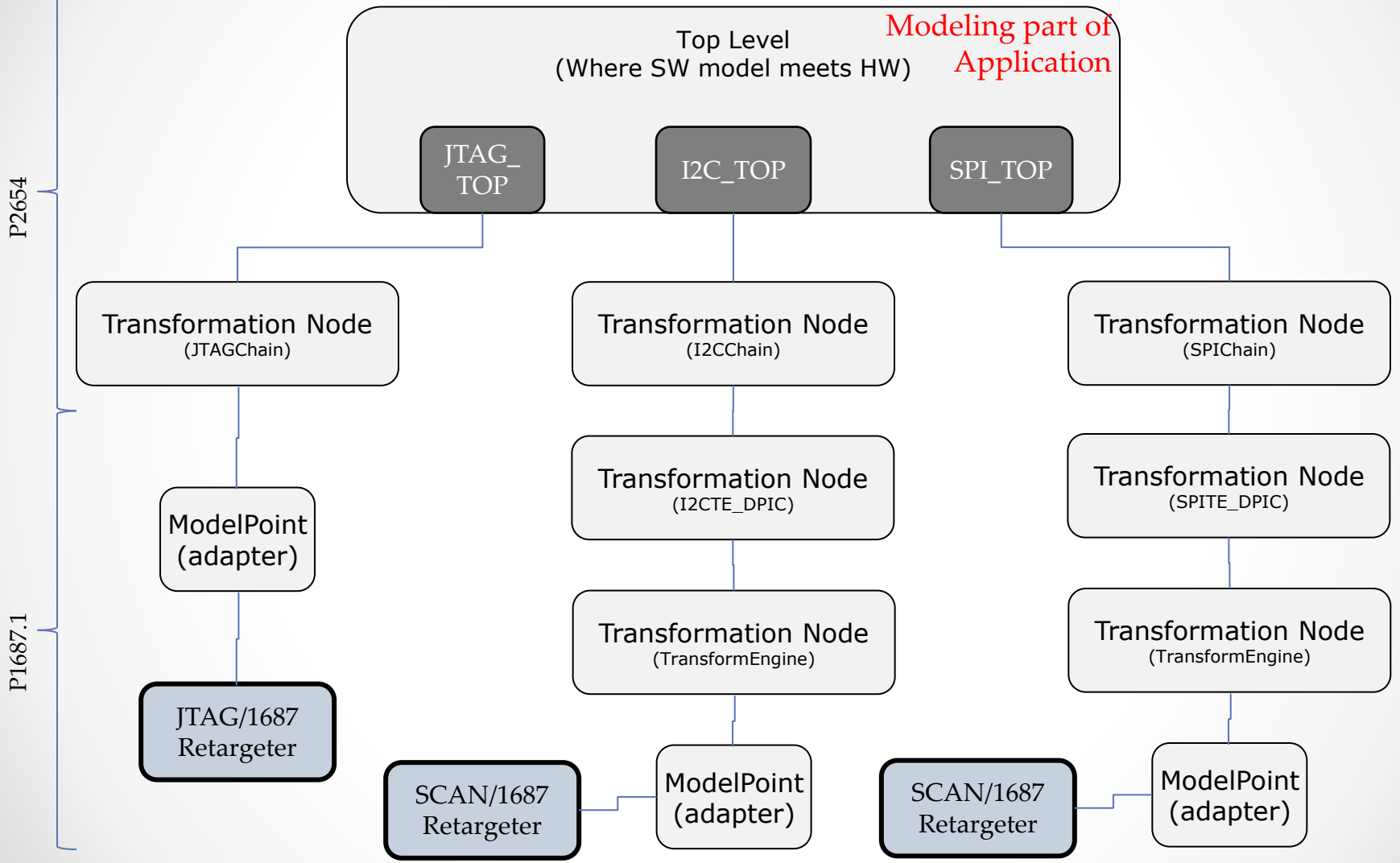
Integrating With Tooling

ModelPoint Adapter Between System Model and External Tools

Generalized P2654 SW Model

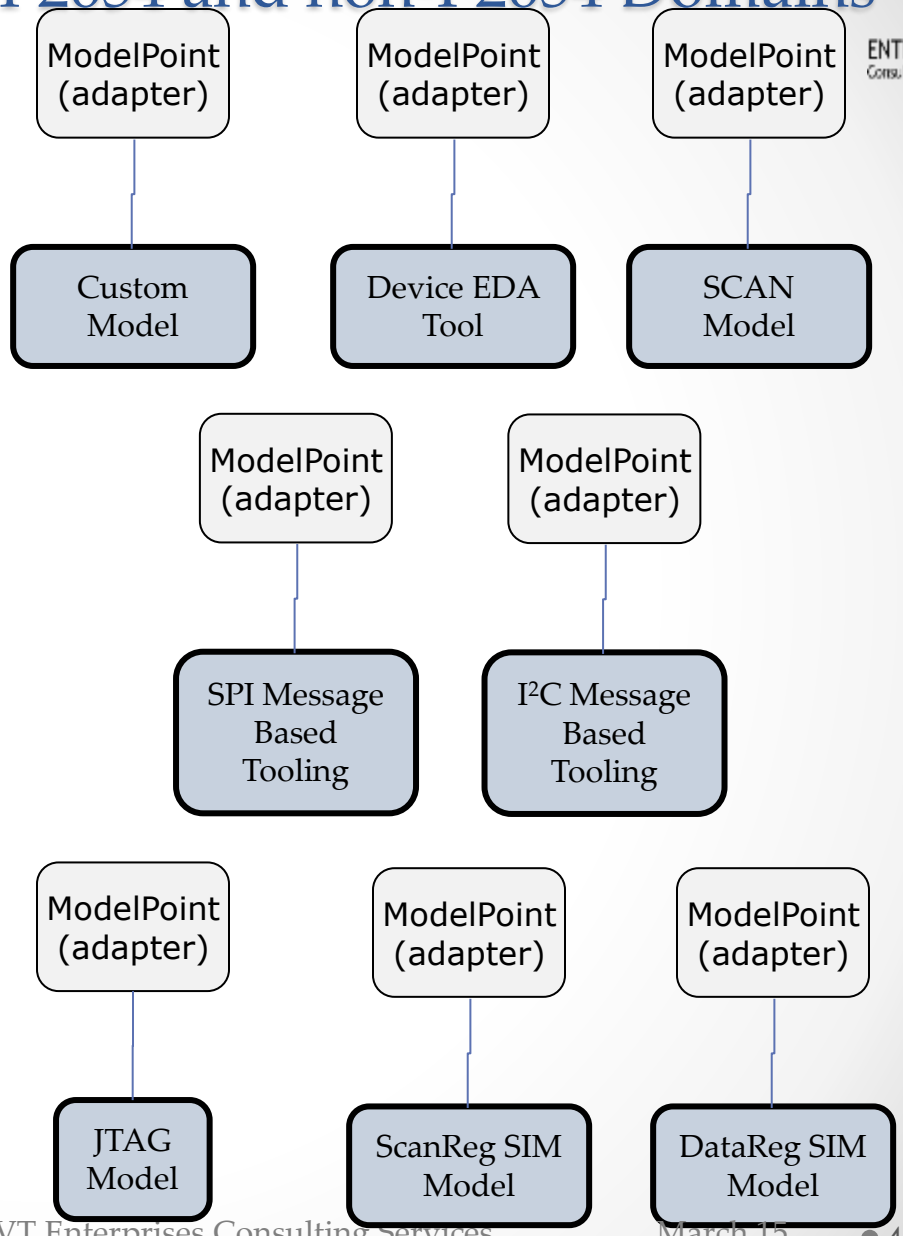


Generalized P1687.1 SW Model



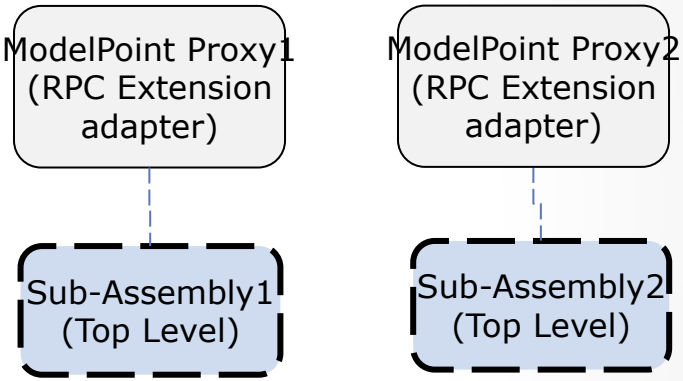
Strategy to Bridge Between P2654 and non-P2654 Domains

- ModelPoint (adapter) is a non-visible node in the topology path
- ModelPoint (adapter) translates non-P2654 grammars to P2654 grammar
- Translation interface between external tooling and the software model message scheme
- Ensures valid RVF messages to the software model
- Provides hook for external model tooling for registers or devices to interact with the model tree
- Provides hook for simulation tooling for registers or devices for verification comparison of simulation and hardware



Strategy to Bridge Between P2654 and non-P2654 Domains

- ModelPoint (adapter) provides a standardized interface to remote sub-systems that provide hooks to extend the modeling of a system to lower levels of control
  - This hook may be gRPC services handling valid P2654 RVF messages
  - This hook may be messages defining commands to be issued to a remote application interface as string messages passed via a Telnet, SSH, Raw RS-232, USB, or Ethernet port between system and sub-system
  - Dotted box may be located on sub-system or virtualized as a representation of sub-system on the system



Discussion 2/15/2021

Continue: How much transparency do we need at each level?

- To what extent does the boundary of an entity (level or node) need to expose detail of entities that are hierarchically below it?
  - What are the implications of not exposing those details?
    - Aren't all levels really black boxes to other levels?
    - We still need to define an API to/from the model for any software (external program or library) that is responsible for transforming the data stream from the lower level
    - There may be a call to an external program to transact the transformation (e.g., Michele's example of a Python program providing the encryption of data to the hardware)
    - Tool vendors may be providing the software model whereas test engineers for a system will provide the application program using the tool's model
    - Don't **need** to expose all the way down to a register. May expose only to a device interface where only SVF vectors are available to be applied to that interface that need to be retargeted by the model.

Discussion 2/15/2021

Continue: How much transparency do we need at each level?

- To what extent does the boundary of an entity (level or node) need to expose detail of entities that are hierarchically below it?
  - What are the implications of not exposing those details?
    - ModelPoint (slides 48 – 49) becomes the important piece of P2654 to provide the interface adapting external tooling with the P2654 model architecture
    - Code provided by P2654 would be exemplary in informative text/diagrams. These could demonstrate the primitives for each level. They should also demonstrate examples to diverse lower level standards and hardware interfaces needing to be coordinated by P2654 (informative).

Discussion 2/15/2021

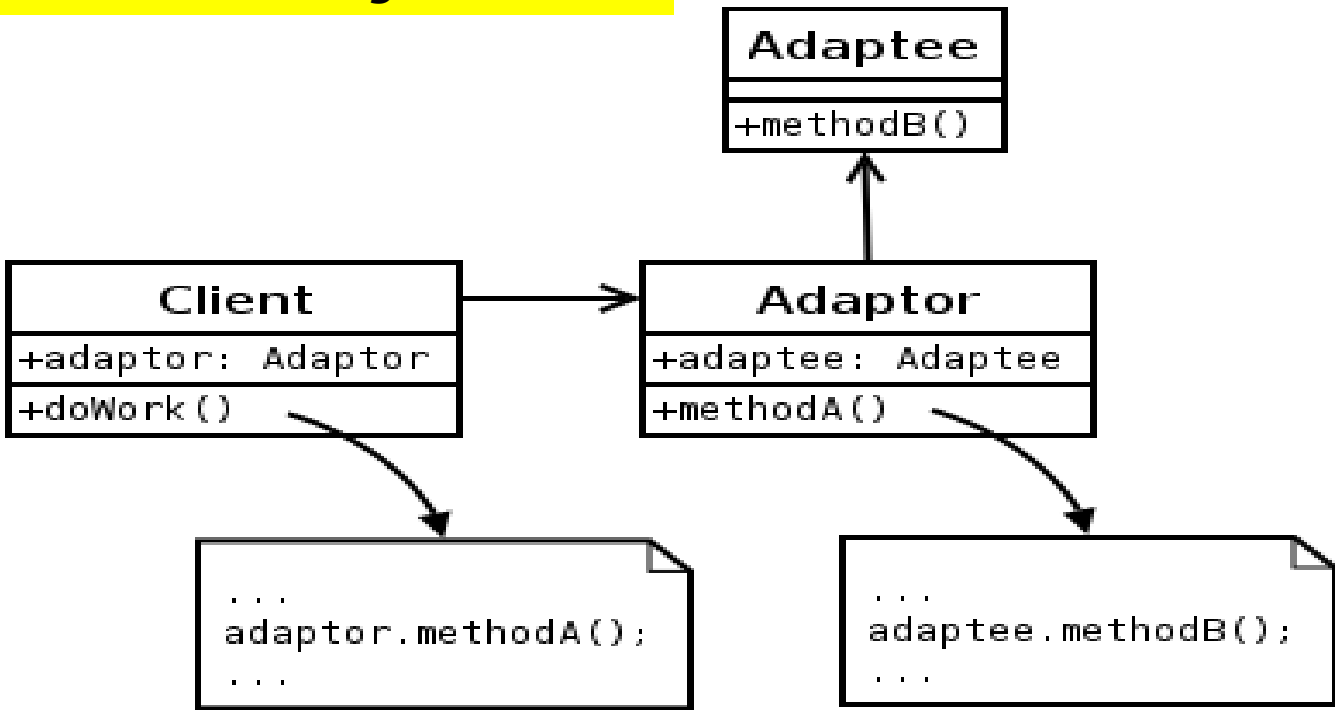
What does a Transform Strategy need to perform?

- Are there mandatory behaviours?
  - Needs to support a particular message protocol format for input and output
  - Needs to be able to synchronize with applications software when certain aspects of flow need to take place to coordinate with other nodes in the tree

- What does the standard need to say about how would we use the descriptions?

Discussion 2/15/2021

ModelPoint is implemented as an Adapter Software Design Pattern



Adapter Pattern
https://en.wikipedia.org/wiki/Adapter_pattern

Discussion 02/22/2021

What does a Transform Strategy need to perform?

- Mandatory aspects?
  - Minimum essential feature set
    - Must understand the grammar of the input messages and mapping to what has to be written as the output grammar to satisfy the requirements of the input messages
    - Synchronization of the message events with the updates of state for the other entities in the system, where there are dependencies
    - Who owns grammar? Client, Host, or both?
- Optional aspects?
  - To what extend would these to be addressed in the standard?

ModelPoint – flesh out details of scope and aspect

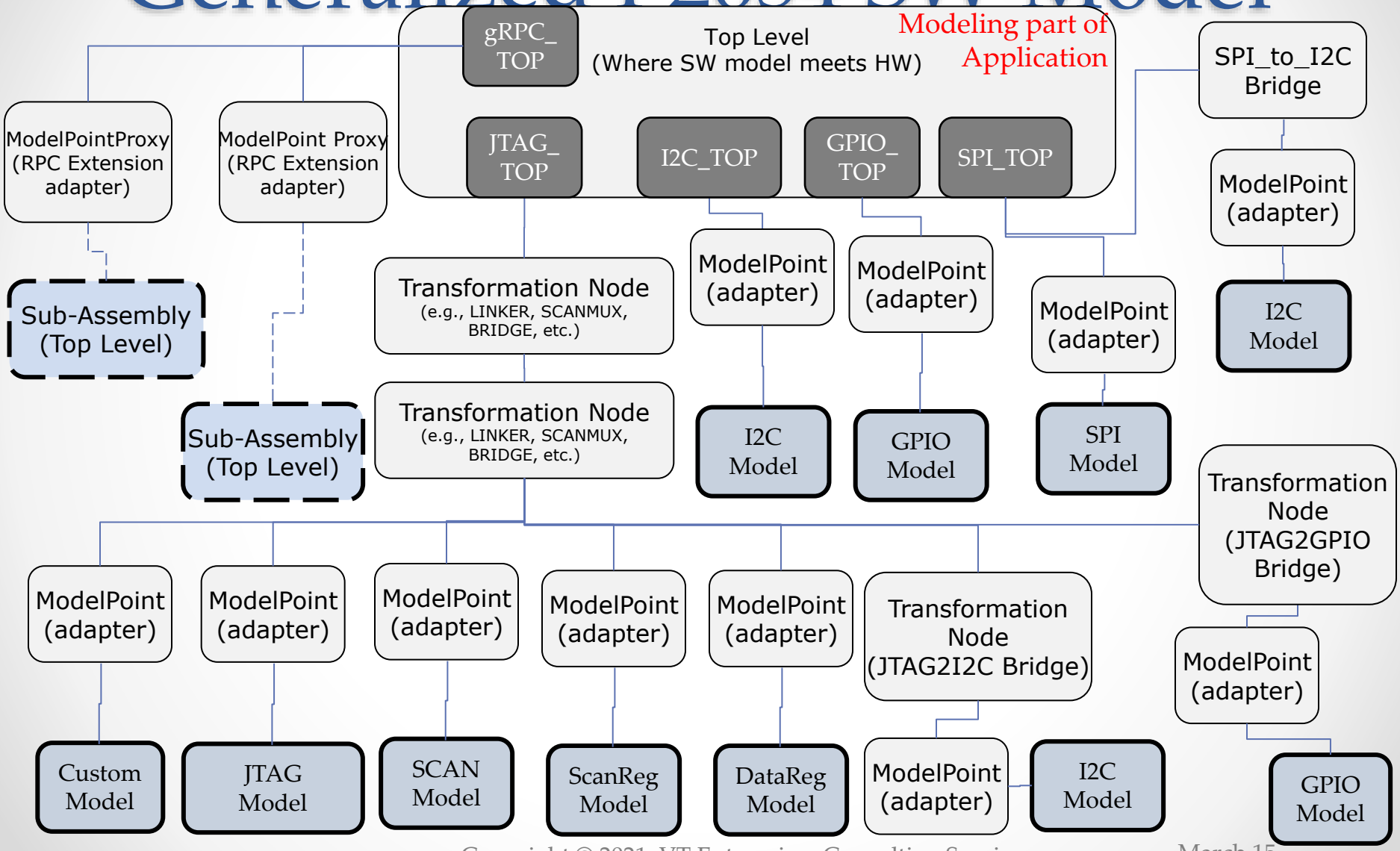
- Common point of hand-off from non-P2654 entities (interfaces, protocols, standards) into P2654.

Discussion 3/8/2021

Primitives: What is needed for a generic group?

- Are the currently proposed base keywords/types the best ones to use?
  - Won't know until applied to all of our use case test cases
- What does the generic case look like (for ad hoc extensions)?
  - Given the third bullet points, how useful is a Generic case and what it looks like in terms of future proofing the standard?
  - The Custom description proposed tries to become a superset of all the descriptions to achieve a catch-all for parameterization to a transform strategy in some generic way.
  - Is it possible to describe all nodes using this notation with specifying certain keywords as optional and useful for only certain types of use cases?
    - Becomes more of an education problem than a technical problem
    - Special case generalized descriptions for certain node use case behaviours (e.g., REGISTER, CHAIN, LINKER, MODELPOINT) may allow simplifying education of use and conditions in implementation
- What are the benefits of having pre-defined keywords/types over simply using the generic?
  - IEEE 1687 tried using a Generic AccessLink and found it needed to specify the IEEE1149.1 AccessLink to more accurately describe the connection between the 1687 network and the device pin interface. Thus, a single Generic AccessLink proved to be insufficient for their case as it opened up the standard to all sorts of ad hoc implementations creating the lack of portability of implementations from the standard outside of 1149.1, but allowable for user defined specification.

Revisited
Generalized P2654 SW Model



Discussion 3/8/2021

- Entities [Proposed categories to consider by BGVT]
  - Registers (Leaf node)
    - Safe value (similar to what BSDL defines for cells)
    - Size (number of cells making up the register storage)
  - ModelPoint (connection to external tooling for the model)(Leaf node)
  - Chain (e.g., jtag path, 1687 network)
  - Linker (e.g., TAP, BSCAN2, Gateway)
    - Superset of Chain providing additional information for managing children
  - Instance (reference to another description file)
  - Root (top level where communication between application and software model is made)
  - Custom (user defined or black box)
  - Controller (representing access point to each UUT test bus hardware driver for Root node)
- ➢ The following slides are used with copyright permission by Bradford G. Van Treuren

A leaf node' that represents the 1687 or JTAG TDR register modeling node.

```
{
  "REGISTER" : {
    "name" : "<node name>",
    "cproto" : "<RVF protocol>",
    "size"  : <size>,
    "safe" : "<safe value>",
    "transform" : "<tstrategy>",
    "inject" : "<istrategy>",
    "injectors" : [
      {<injector>},
      {<injector>},
    ],
    "debug" : "<dstrategy>",
    "sticky" :  false,
  }
}
```

REGISTER

<node name>: instance name for the Model Node

<RVF protocol>: Message callback protocol sent by the REGISTER out of cproto Client

<size>: number of bits composing the register in decimal notation

<safe value>: the default value for the model considered a safe value for the model

<tstrategy>: The tstrategy to use (e.g., SCAN, DIGITAL, TDR). This name corresponds to the library containing the transformation callbacks defining the strategy to use.

<istrategy>: The istrategy corresponds to the library containing callbacks defining the injection strategy to use. State changes are done via injection.

<injector>: The injector corresponds to the library containing the interface to the injection source and corresponding parameters.

<dstrategy>: Optional debug process to use. The dstrategy corresponds to the library containing the debug callbacks for this node.

sticky: Used to specify the safe value corresponds to the last value it has been assigned, if set to true. Otherwise, the safe value is always used. If model is reset, the specified <safe value> is reloaded as the safe value.

INSTANCE

A leaf node' that represents the instantiation of a module at the point of insertion (as for any other child nodes).

```
{
  "INSTANCE" : {
    "name" : "<node name>",
    "sit" : "<SIT file>"
  },
  "INSTANCE" : {
    "name" : "<node name>",
    "factory" : "<factory name>"
  }
}
```

<node name>: instance name for the Model Node

<SIT file>: The name or path of a SIT file with .sit extension. It must be enclosed between double quotes if filename contains spaces or reserved keywords.

<factory name>: identify a model factory registered by a plugin

A hierarchical node' that represents a chained hierarchy. This node models a structure of multiple segments as a hierarchy of sibling children nodes of the same order. Cardinality order of the modules in the list is important. Modules are specified with the children keyword.

```
{
  "CHAIN" : {
    "name" : "<node name>",
    "cproto" : "<RVF protocol>",
    "hproto" : "<RVF protocol>",
    "transform" : "<tstrategy>",
     "inject" : "<istrategy>",
    "injectors" : [
     {<injector>},
     {<injector>},
    ],   "debug" : "<dstrategy>",
    "visible" : false,
    "children" : [
     {
       <child node>,
       <child node>
     }]
  }
}
```

CHAIN

<node name>: instance name for the Model Node

<RVF protocol>: Message callback protocol used by the corresponding Client and Host Interfaces

<tstrategy>: Transformation strategy to use (e.g., SCAN, JTAG, I2C, SPI). This name corresponds to the library containing the transformation callbacks defining the strategy to use.

<istrategy>: The istrategy corresponds to the library containing callbacks defining the injection strategy to use. State changes are done via injection.

<injector>: The injector corresponds to the library containing the interface to the injection source and corresponding parameters.

<dstrategy>: Optional debug process to use. The strategy corresponds to the library containing the debug callbacks for this node.

<visible>: Attribute to specify whether node is visible in the path specification or not. Default is false, not visible.

<child node>: One or more nodes defining a hierarchical sub-tree of this CHAIN.

LINKER

A hierarchical node' that represents a selectable chained hierarchy. This node models a structure of multiple segments as a hierarchy of sibling children nodes that may be present or missing from the path. Order of the modules in the list is important. Modules are specified with the children keyword.

```
{
  "LINKER" : {
    "name" : "<node name>",
    "cproto" : "<RVF protocol>",
    "hproto" : "<RVF protocol>",
    "transform" : "<tstrategy>",
    "inject" : "<istrategy>",
    "injectors" : [
      {<injector>},
      {<injector>},
    ],
    "debug" : "<dstrategy>",
    "visible" : false,
    "selector" : "<selector>",
    "control" : [ "<control node>", "<control node>"],
    "derivations" : <count>,
    "parameters" : "<parameters>",
    "children" : [ {
        <child node>,
        <child node>
      } ]
  }
}
```

<node name>: instance name for the Model Node

<RVF protocol>: Message callback protocol used by the corresponding Client and Host Interfaces

<tstrategy>: Transformation strategy to use (e.g., SCAN, JTAG, I2C, SPI). This name corresponds to the library containing the transformation callbacks defining the strategy to use.

<istrategy>: The istrategy corresponds to the library containing callbacks defining the injection strategy to use. State changes are done via injection.

<injector>: The injector corresponds to the library containing the interface to the injection source and corresponding parameters.

<dstrategy>: Optional debug process to use. The strategy corresponds to the library containing the debug callbacks for this node.

<visible>: Attribute to specify whether node is visible in the path specification or not. Default is false, not visible.

<selector>: A string identifying the callback set for Linker (e.g., "Binary", "Binary_noidle", "One_Hot", "One_Hot_noidle", "N_Hot", "N_Hot_noidle", {"Table": [<ordered list of values>]}, or {"Custom" : "<custom strategy>"}), where <custom strategy names a library containing the customized selection callbacks.

<control node>: A list of register or slice of a register used to define the bits used to control the selection. <control>s are concatenated to obtain the Virtual Register selecting the linker.

<child node>: One or more nodes defining a hierarchical sub-tree of this LINKER.

<parameters>: String containing <selector> specific parameters.

A leaf node' that represents a translator between the software model and external tools or protocol formats. This node references the strategy to be used to bridge the tooling with the model tree.

```
{
  "MODELPOINT" : {
    "name" : "<node name>",
    "cproto" : "<RVF protocol>",
    "transform" : "<tstrategy>",
    "debug" : "<dstrategy>",
    "visible" : false,
    "transformParams" : "<parameters>"
  }
}
```

MODELPOINT

<node name>: instance name for the Model Node

<RVF protocol>: Message callback protocol sent by the MODELPOINT out of cproto Client

<tstrategy>: Translation strategy to use . This name corresponds to the library containing the translation methods defining the strategy to use for bridging external resources with the model tree.

<dstrategy>: Optional debug process to use. The strategy corresponds to the library containing the debug callbacks for this node.

<visible>: Attribute to specify whether node is visible in the path specification or not. Default is false, not visible.

<parameters>: String containing <transform strategy> specific parameters.

Discussion 3/15/2021

- Entities [Proposed categories to consider by BGVT]
  - Registers (Leaf node)
    - Safe value (similar to what BSDL defines for cells)
    - Size (number of cells making up the register storage)
  - ModelPoint (connection to external tooling for the model)(Leaf node)
  - Chain (e.g., jtag path, 1687 network)
  - Linker (e.g., TAP, BSCAN2, Gateway)
    - Superset of Chain providing additional information for managing children
  - Instance (reference to another description file)
  - Root (top level where communication between application and software model is made)
  - Custom (user defined or black box)
  - Controller (representing access point to each UUT test bus hardware driver for Root node)
- ➢ The following slides are used with copyright permission by Bradford G. Van Treuren

CUSTOM

A model node' that represents the customizable node of a tree where none of the primitive node types describe the behavior of the node. The children represent a list of dissimilar sub-trees coordinated for a test.

```
{
  "CUSTOM" : {
    "name" : "<node name>",
     "cproto" : "<RVF protocol>",
    "hproto" : "<RVF protocol>",
    "transform" : "<tstrategy>",
    "inject" : "<istrategy>",
    "injectors" : [
     {<injector>},
     {<injector>},
    ],    "debug" : "<dstrategy>",
    "visible" : true,
    "selector" : "<selector>",
    "control" : [ "<control node>", "<control node>"],
    "parameters" : "<parameters>",
    "children" : [
     {
       <child node>,
       <child node>
     }]
  }
}
```

<node name>: instance name for the Model Node

<RVF protocol>: Message callback protocol used by the corresponding Client and Host Interfaces

<tstrategy>: Transformation strategy to use (e.g., SCAN, JTAG, I2C, SPI). This name corresponds to the library containing the transformation callbacks defining the strategy to use.

<istrategy>: The istrategy corresponds to the library containing callbacks defining the injection strategy to use. State changes are done via injection.

<injector>: The injector corresponds to the library containing the interface to the injection source and corresponding parameters.

<dstrategy>: Optional debug process to use. The strategy corresponds to the library containing the debug callbacks for this node.

<visible>: Attribute to specify whether node is visible in the path specification or not. Default is false, not visible.

<selector>: A string identifying the callback set for Linker (e.g., "Binary", "Binary_noidle", "One_Hot", "One_Hot_noidle", "N_Hot", "N_Hot_noidle", {"Table": [<ordered list of values>]}, or {"Custom" : "<custom strategy>"}), where <custom strategy names a library containing the customized selection callbacks.

<control node>: A list of register or slice of a register used to define the bits used to control the selection. <control>s are concatenated to obtain the Virtual Register selecting the customized linker.

<parameters>: String containing node specific parameters.

<child node>: One or more nodes defining a hierarchical sub-tree of this CUSTOM node.

Discussion 3/15/2021

- Brad started out with CUSTOM type and realized there were cases that showed up over and over with a common footprint (e.g., MODELPOINT, CHAIN, LINKER). Should the user be allowed to define their own specialized model types? Will that break standard or portability?
- These descriptions provide a handle or reference to the real software (transform strategy, inject strategy, debug strategy, injector) that defines the actual behavior of the node. A standard should only provide reference to defined software and not describe the behavior if going to be flexible and portable.
- Order of children is dictated by the tstrategy software. For example, a TAP uses a LINKER description that contains an IR register and a data MUX/LINKER for the TDR management as the children. The data LINKER has children that align to the selector table indices defined in the description.

➢ The following slides are used with copyright permission by Bradford G. Van Treuren

A model node' that represents the top node of a tree where CONTROLLER nodes are coordinated as a single unit. The children represent a list of dissimilar sub-trees, as CONTROLLERS, coordinated for a test.

```
{
  "ROOT" : {
    "name" : "<node name>",
    "hproto" : "<RVF protocol>",
    "transform" : "<tstrategy>",
    "debug" : "<dstrategy>",
    "visible" : true,
    "children" : [
      {
        <child node>,
        <child node>
      }]
  }
}
```

ROOT

<node name>: instance name for the Model Node

<RVF protocol>: Message callback protocol used by the corresponding Client and Host Interfaces

<tstrategy>: Transformation strategy to use (e.g., SCAN, JTAG, I2C, SPI). This name corresponds to the library containing the transformation callbacks defining the strategy to use.

<dstrategy>: Optional debug process to use. The strategy corresponds to the library containing the debug callbacks for this node.

<visible>: Attribute to specify whether node is visible in the path specification or not. Default is false, not visible.

<child node>: One or more nodes defining a hierarchical sub-tree of this ROOT node.

CONTROLLER

A model node' that represents the top node of a tree where event messages are folded into hardware device driver calls. The children represent a list of similar sub-trees coordinated for a test and controlled by the CONTROLLER.

```
{
  "CONTROLLER" : {
    "name" : "<node name>",
    "hproto" : "<RVF protocol>",
    "transform" : "<tstrategy>",
    "debug" : "<dstrategy>",
    "visible" : true,
    "children" : [
      {
        <child node>,
        <child node>
      }]
  }
}
```

<node name>: instance name for the Model Node

<RVF protocol>: Message callback protocol used by the corresponding Client and Host Interfaces

<tstrategy>: Transformation strategy to use (e.g., SCAN, JTAG, I2C, SPI). This name corresponds to the library containing the transformation callbacks defining the strategy to use.

<dstrategy>: Optional debug process to use. The strategy corresponds to the library containing the debug callbacks for this node.

<visible>: Attribute to specify whether node is visible in the path specification or not. Default is false, not visible.

<child node>: One or more nodes defining a hierarchical sub-tree of this CONTROLLER node.