An overview of MDDI Bus

Why?

Integration of MDD tools

MDD tool integration is a specialization of generic tool integration.

Generic tool integration requirements

- service integration
    - through service interfaces
        - registry-driven interaction
    - requirements on tool
        - a tool should provide and publish service APIs (provided and possibly required)
    - requirements on bus
        - service registry, matching of provided/required interfaces
- event-driven interaction
    - event subscription and notification service
    - requirements on tool
        - event interfaces (events it generates and consumes) should be published
    - requirements on bus
        - event management services

MDD tool integration requirements

MDD tool categories

- model editors (visual, textual, etc)
    - need to transform between concrete and abstract syntaxes
- model compilers and executers
    - operate mostly at the abstract syntax level
- model analyzers
    - operate mostly at the abstract syntax level

MDD tool characteristics

- An MDD tool processes models of a given meta model
- Meta models of different tools can overlap
    - syntactic overlap (i.e. same meta classes exist in both)
    - semantic overlap (i.e. meta classes may be different but objects of one are derivable from the objects of the other)

- There can be a unified meta model of which tool specific meta models are views

MDD tool integration requirements

- All the generic tool integration requirements described above and the following:
    - Model integration
        - each tool publishes the MOF meta model of the models it manages
        - producer-consumer (of models) relationships between tools
        - push/pull interfaces;
            - batch push/pull
                - entire model is pushed/pulled
                - in XMI form
            - incremental push/pull
                - delta models
            - push/pull may be initiated by the participating tools themselves, or by an external agent (such as host IDE or a process execution engine, etc)

    - requirements on bus
        - setup producer consumer relationships between tools
            - meta model maps, in the form of transfromation specs such as QVT
                - default model map (copy) that can be overridden by extensions where transformations can be plugged in
                    - transformations may be simple copiers when the source and target meta models are identical
        - batch push
            - export source model from source tool in XMI form
            - run transformation, transform it to models of the consumer tools
            - import target models into target tools
        - delta push
            - export delta model from the source tool
            - run transformation which produces the target model in XMI, and delta model of the target model as a side effect.
            - import delta models into target tools

          Alternately when a tool does not support delta models,
            - tools provide notification mechanisms for change notification
            - bus builds delta models from these notifications
            - Upon push, transformations are run as before

        - Bus updates the target models by invoking model update APIs of the target tools
      - pull is similar to push, it is just that it is initiated by a consumer and models are pulled from all the registered producers.

  - requirements on tool
    - batch push/pull
      - XMI export/import of abstract models
      - XMI export/import of view models optionally
    - delta push/pull
      - delta model support

      or
      - model change notification support and APIs to update models

- View model integration
  Producers and consumers of view models are expected to have the same meta models and same visual notations to render them. A view model just needs to be copied from producer to the consumer without needing any transformation.

- Inter-model consistency checking between models produced by different tools
  - a third pre-defined generic constraint validation tool acts as a consumer of the respective tools
    - consistency constraints can be specified using OCL or QVT

A possible eclipse-based realization of the above

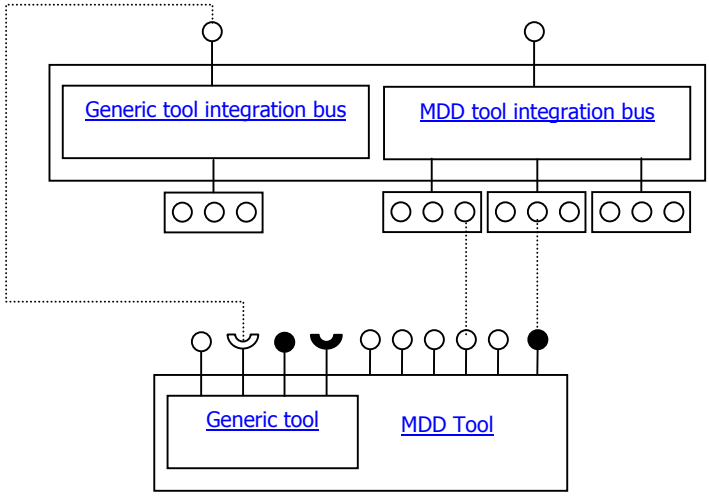
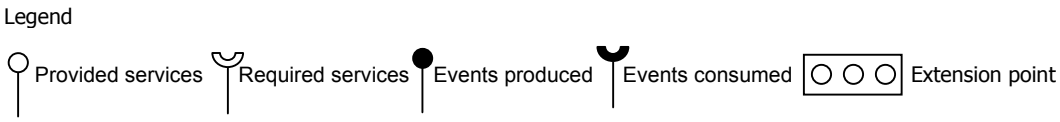


Fig. 1. Architecture of MDDI bus



Generic tool:
- Service interface
  - provided interface
  - required interface

- Event interface
  - produced events
    - <event_name> <data>*
  - consumed events
    - <event_name> <event_handler> <data>*

MDDTool extends Generic tool:
- XMI model export interface
- XMI model import interface
- XMI view model export interface
- XMI view model import interface
- Model change event notification interface
- Reflexive API to update models

Generic tool integration bus:
// Provides service registry and event management services
  - Tool extension point:
    - <Tool_id>

// service registry related
- <startup_interface>
- (<interfrace_name> <interface_class>)*
- <shutdown_interface>

// event subscription inerface
- (<event_id> <event_handler>)*

- APIs:

// service lookup
- getInterface(<interface_name>):Interface

// event publication interface
- publishEvent(<event_id>, <data>)

MDD tool integration bus extends Generic tool integration bus:
- Tool extension point:
  - <Tool_id>
  - <Meta_model_spec> // meta model XMI file
  - <XMI_export_interface>
  - <XMI_import_interface>
  - [<XMI_view_export_interface>]
  - [<XMI_view_import_interface>]
  - MOF-reflexive-api = (yes|no)
  - ChangeNotification = (yes|no)
- Producer-consumer extension point:
  - Producer-tool=<tool_id>
  - Consumer-tool=<tool_id>
  - [transformation-spec=<xyz>.qvt]
  - Transfer-mode=(batch|delta)
- Inter-model consistency extension point:
  - Producer-tools=(<tool_id_1>, <tool_id_2>,...<tool_id_n>)
  - Consistency-spec=(<xyz>.ocl | <xyz>.qvt)
- APIs:

  // push/pull interface
  - modelPull(<into_tool_id>)
  - modelPush(<from_tool_id>)
  - viewPull(<into_tool_id>)
  - viewPush(<from_tool_id>)

  // change notification interface
  - notifyChange(<from_tool_id>, <change-kind>, <value>, [<oldValue>])