EclipseSource

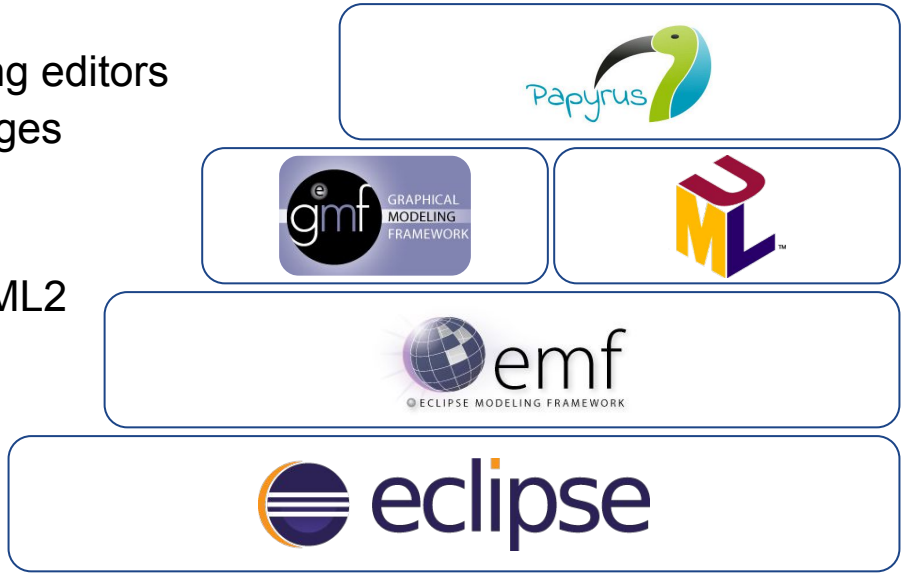
BridgePoint to Papyrus

Architectural Design Alternatives

Workshop, 24th of June, 2016

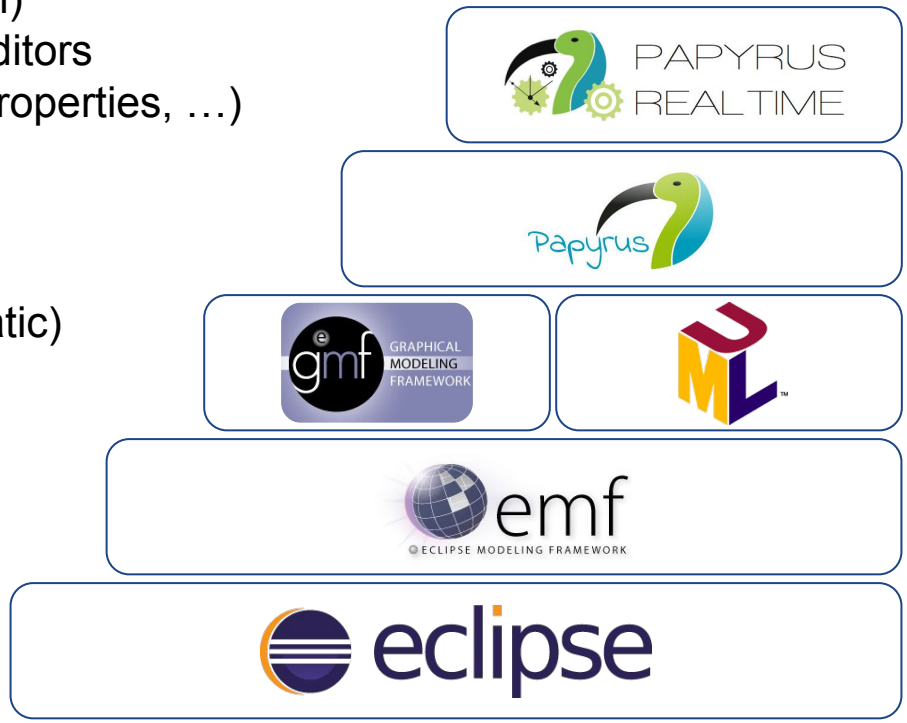
Papyrus Architecture Stack

- Papyrus
  - Built on UML2 and GMF
  - GMF-based diagrams for UML2
- GMF
  - FW for building graphical modeling editors
  - For EMF-based modeling languages
- UML2
  - EMF-based implementation of UML2
- UML2/GMF models
  - Plain EMF models
  - Any EMF technology works ~



Papyrus-RT Architecture Stack

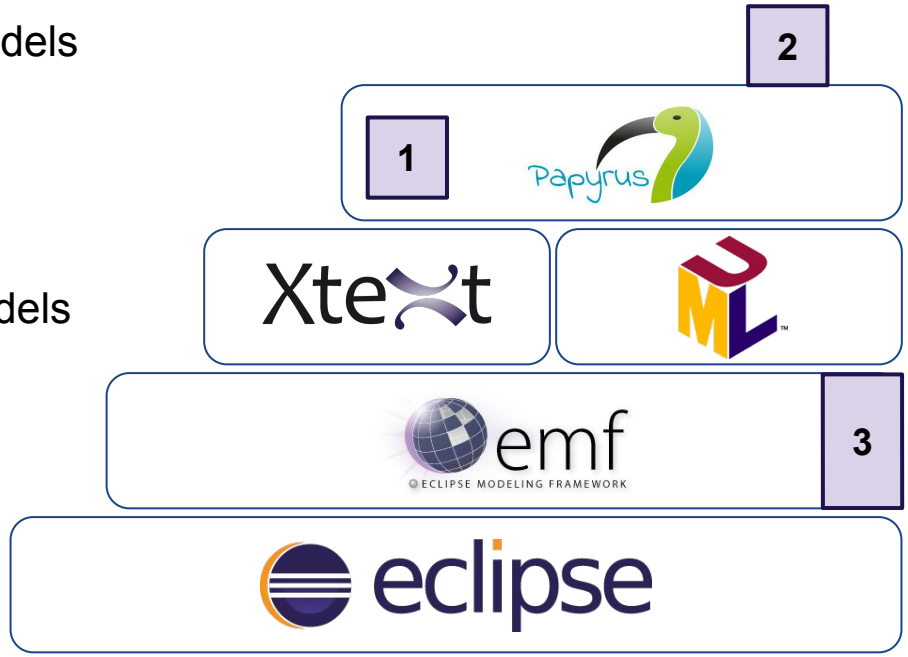
- Papyrus-RT
  - UML2-based profile (~metamodel)
  - Extension of Papyrus' diagram editors
  - Customization of tools (palette, properties, …)
  - Code generators, ...
- Papyrus/UML2-based profiles
  - Ecore metamodel generation (static)
  - EMF metamodel API generation



Xtext and how it is or can be used with Papyrus

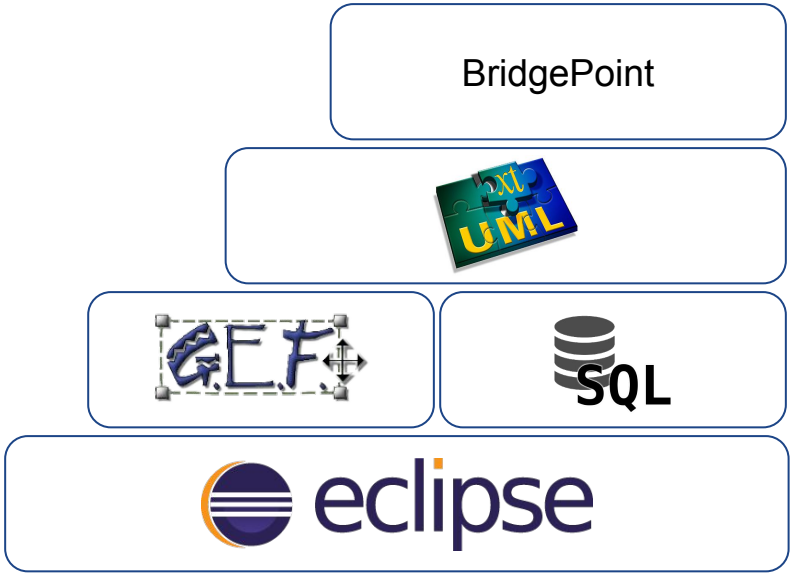
- Xtext
  - FW for building textual modeling languages
  - Editor for users
  - Parser and serializer for EMF models

- Is or can be used as a
  1. Textual editing inside diagrams
  2. Textual editor for UML models
  3. Persistence format for (UML) models



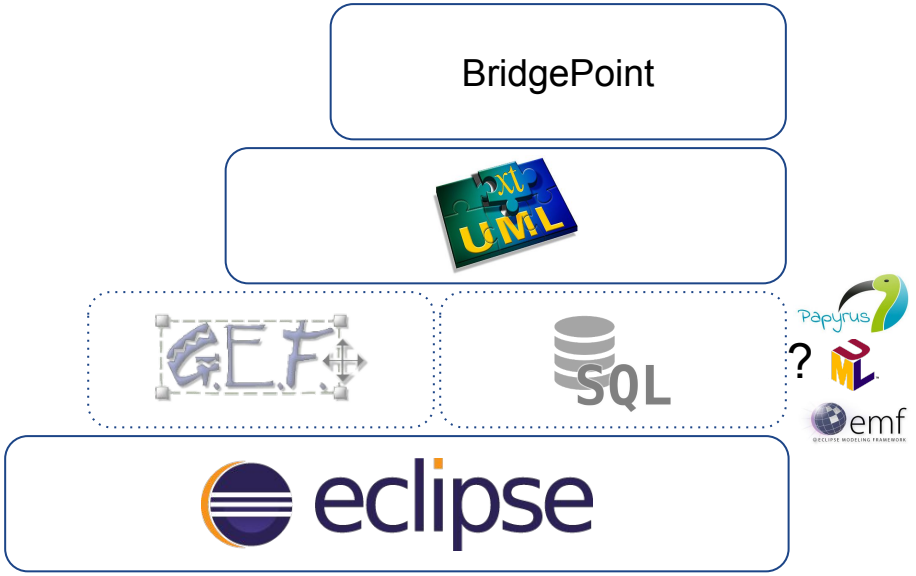
BridgePoint Status Quo

- Bridgepoint
  - GEF-based Modeling editors
  - Model compilers (rely on xtUML metamodel)
  - Model interpreters (rely on xtUML metamodel)
- xtUML
  - Metamodel of the language
  - API to interact with model
  - Implementation of the xtUML API
- SQL
  - Format to persist models
- GEF
  - Technology to build modeling editors



Goal of the Migration

- Bridgepoint
  - ~~GEF-based modeling editors~~ Extended existing modeling editors
  - Model compilers (rely on xtUML metamodel)
  - Model interpreters (rely on xtUML metamodel)
- xtUML
  - Metamodel of the language
  - API to interact with model
  - Implementation of the xtUML API
- ~~SQL~~
  - ~~Format to persist models~~
- ~~GEF~~
  - ~~Technology to build modeling editors~~



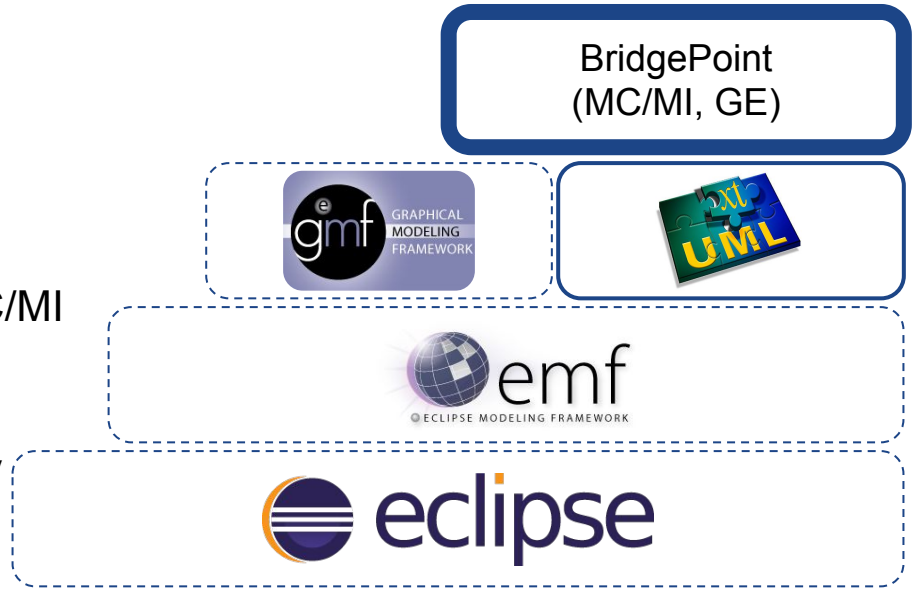
Architectural Design Alternatives

1. Ecore-based xtUML
2. UML2 emulation of xtUML
3. xtUML to UML2 transformation
4. UML2 and UML Profile and xtUML-compatible API (++)

Ecore-based xtUML

- BridgePoint
  - Model compilers and interpreters (MC/MI)
- Graphical editors (GE)
  - Based on GMF (or Sirius, Graphiti)
- xtUML
  - Implementation with Ecore

- Advantages
  - Flexibility regarding metamodel
  - Easy integration with existing MC/MI
- Disadvantages
  - Implementation of own editors (!)
  - No reuse of Papyrus functionality



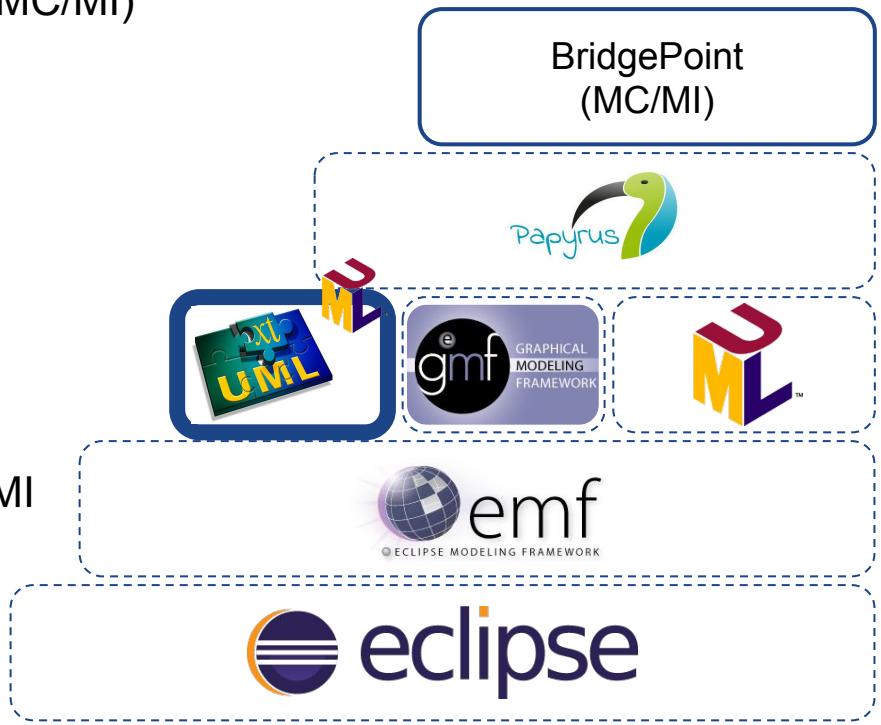
Ecore-based xtUML vs Papyrus

Reuse of Papyrus requires (compatibility with) UML2!

UML2 emulation of xtUML

- BridgePoint
  - Model compilers and interpreters (MC/MI)
- Graphical editors
  - Reused from Papyrus
- xtUML
  - Implementation with Ecore
  - Emulation of UML2 MM API

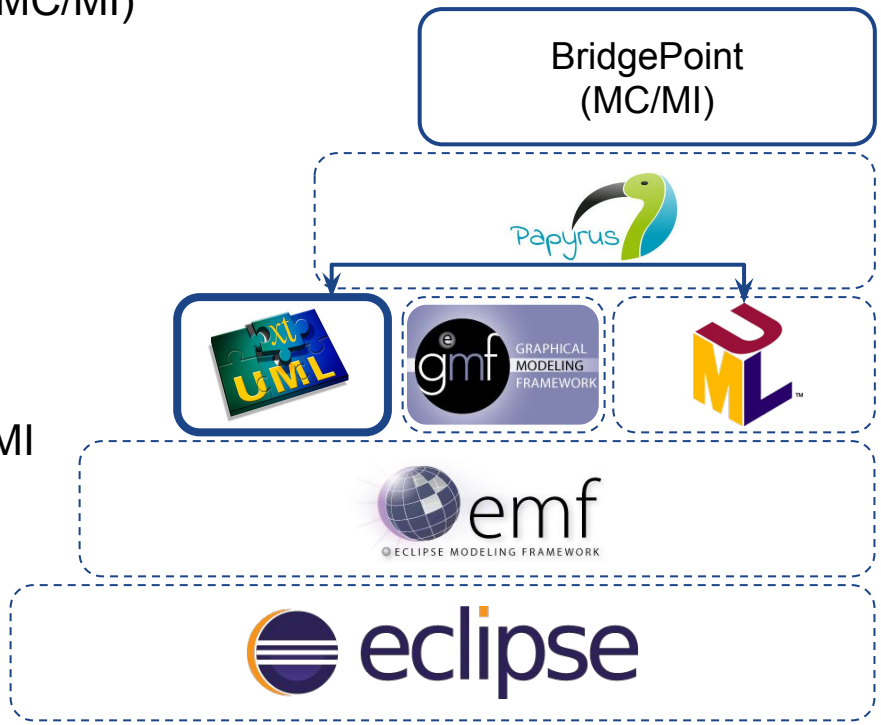
- Advantages
  - Papyrus UML2 editor reuse
  - Easy integration with existing MC/MI
- Disadvantages
  - UML2 emulation → high effort (!)
  - Only commonalities with UML2 can be be made visible to Papyrus



xtUML to UML2 Transformation

- BridgePoint
  - Model compilers and interpreters (MC/MI)
- Graphical editors
  - Reused from Papyrus
- xtUML
  - Implementation with Ecore

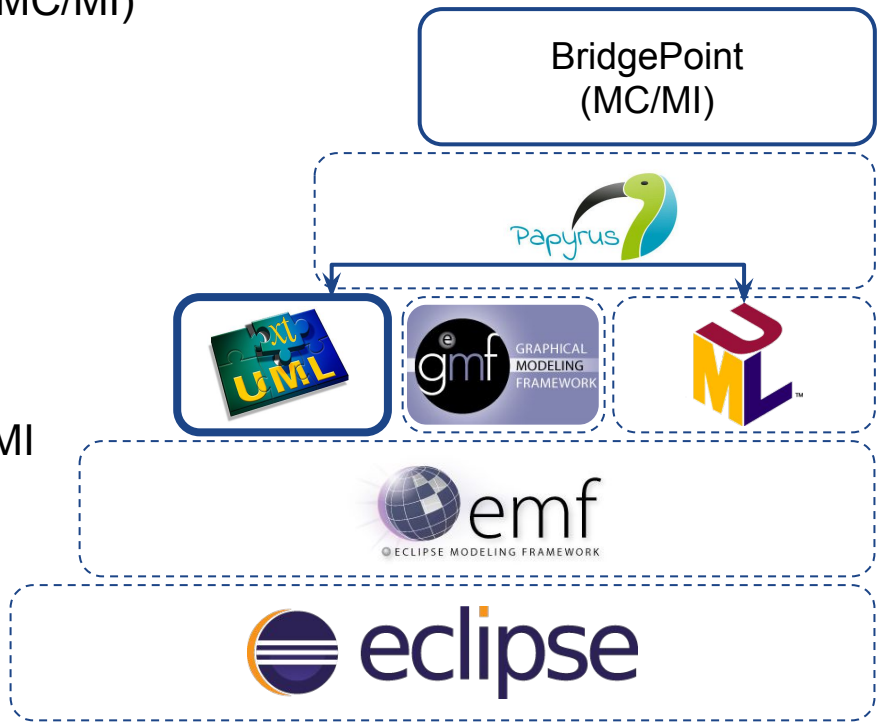
- Advantages
  - Papyrus UML2 editor reuse
  - Easy integration with existing MC/MI
- Disadvantages
  - ~~UML2 emulation → high effort (!)~~
  - Bi-directional transformation
  - Transformation performance risk
  - Only commonalities with UML2 can be be made visible to Papyrus (!)



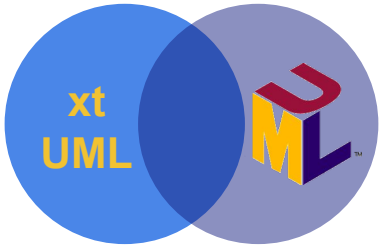
xtUML to UML2 Transformation

- BridgePoint
  - Model compilers and interpreters (MC/MI)
- Graphical editors
  - Reused from Papyrus
- xtUML
  - Implementation with Ecore

- Advantages
  - Papyrus UML2 editor reuse
  - Easy integration with existing MC/MI
- Disadvantages
  - ~~UML2 emulation ⟶ high effort (!)~~
  - Bi-directional transformation
  - Transformation performance risk
  - **Only commonalities with UML2 can be be made visible to Papyrus (!)**

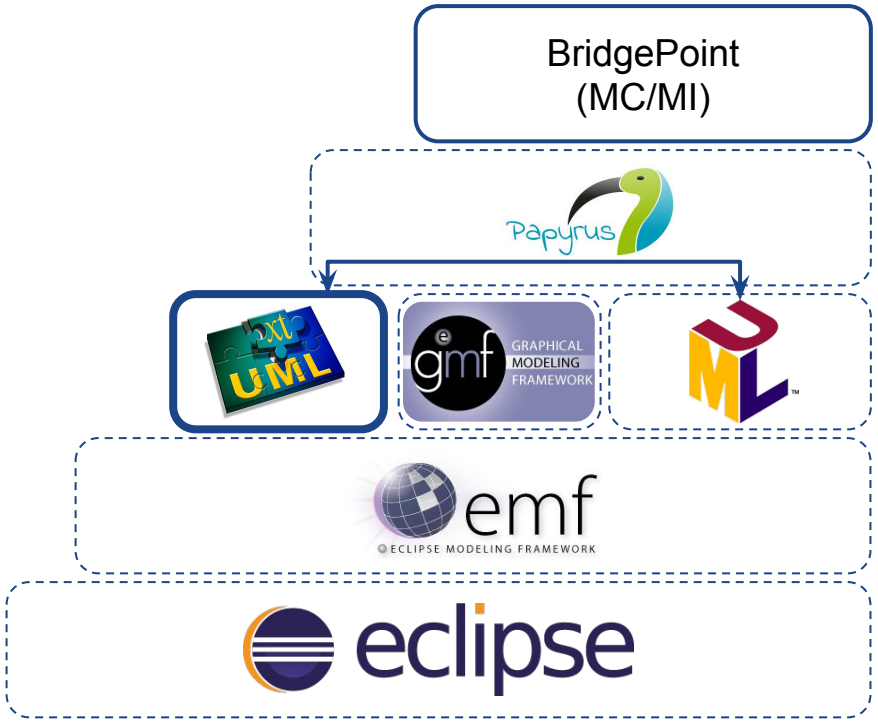


xtUML vs UML2



- Papyrus supports UML only
  - Intersection of xtUML and UML
  - Rest is invisible to Papyrus
  - Custom extension of Papyrus

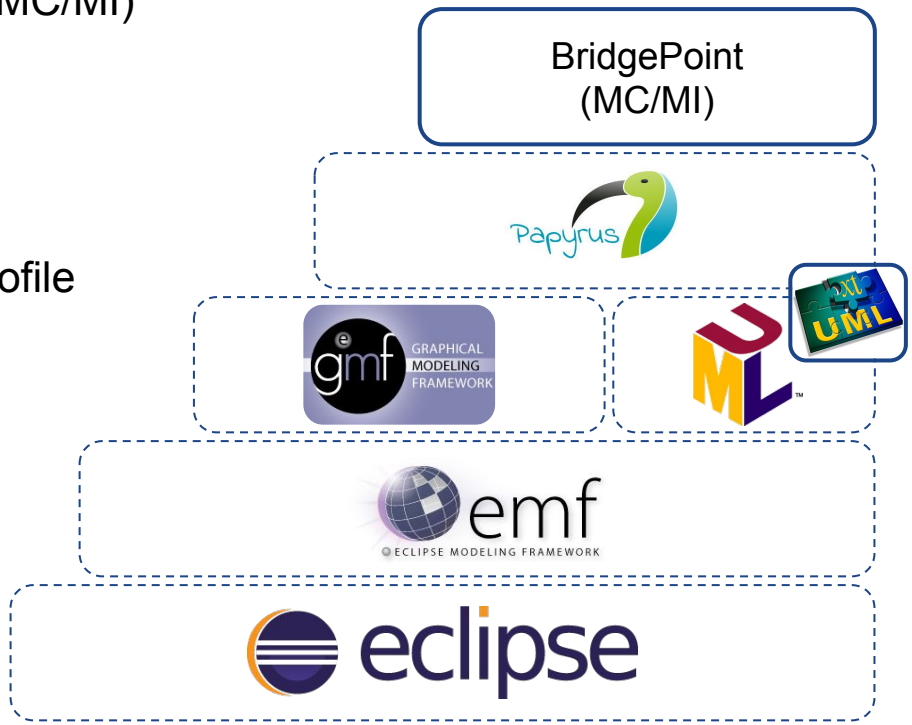
  → **UML2 profile**



UML2-based xtUML

- BridgePoint
  - Model compilers and interpreters (MC/MI)
- Graphical editors
  - Reused from Papyrus
  - Extended for xtUML
- xtUML
  - Implementation as native UML2 profile

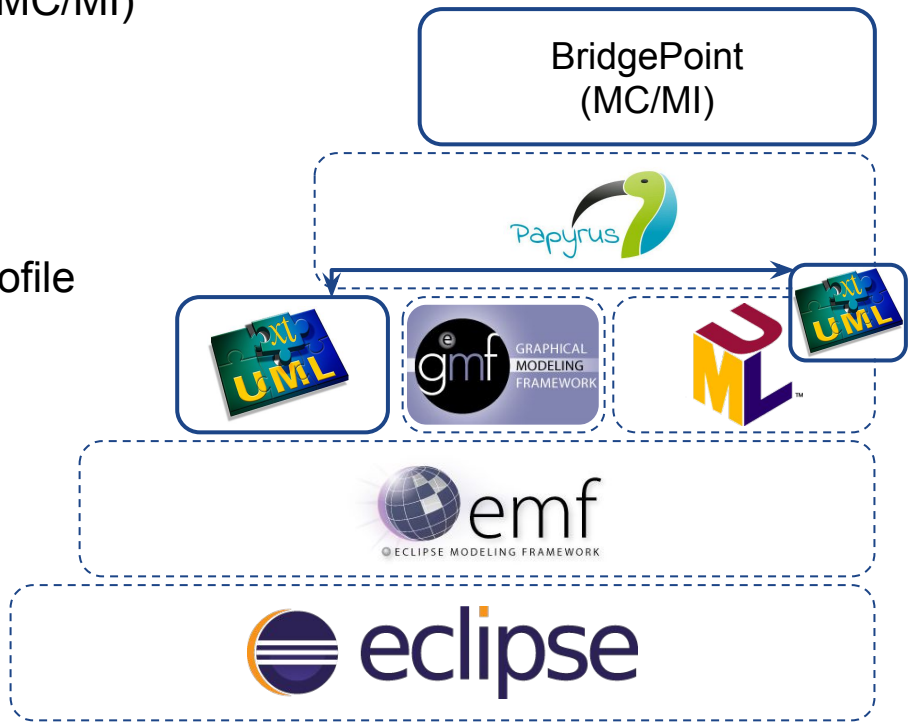
- Advantages
  - Papyrus UML2 editor reuse
  - Native Papyrus extension support
- Disadvantages
  - MC/MI have to be reimplemented for UML2 & profile API



UML2-based xtUML with transformation to native xtUML

- BridgePoint
  - Model compilers and interpreters (MC/MI)
- Graphical editors
  - Reused from Papyrus
  - Extended for xtUML
- xtUML
  - Implementation as native UML2 profile

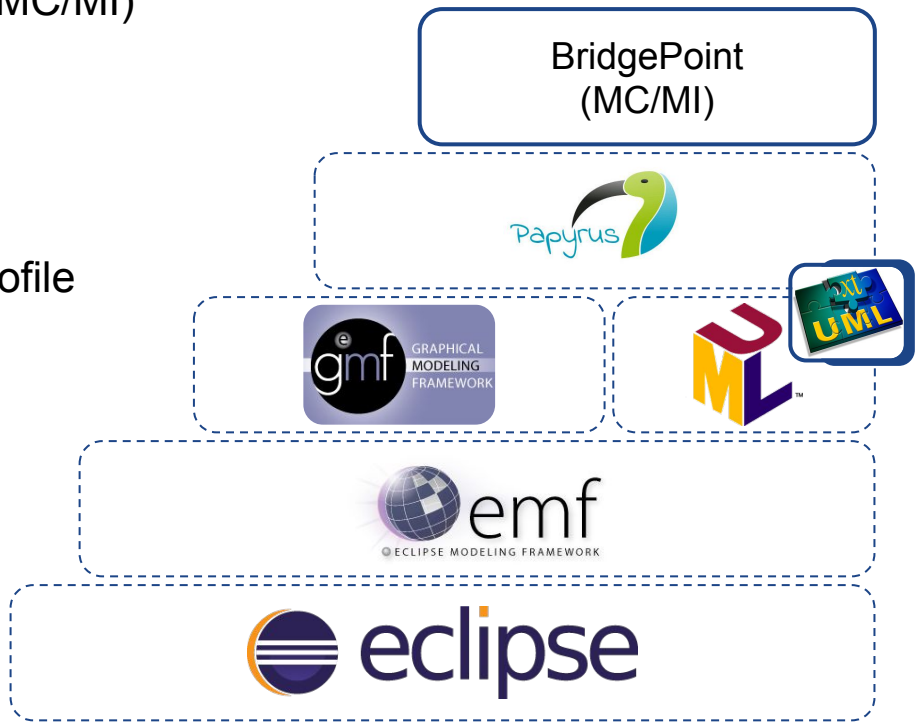
- Advantages
  - Papyrus UML2 editor reuse
  - Native Papyrus extension support
  - MC/MI can be reused
  - Support for partial migration
- Disadvantages
  - Bi-directional transformation
  - Transformation performance risk



UML2-based xtUML with native xtUML compatible API

- BridgePoint
  - Model compilers and interpreters (MC/MI)
- Graphical editors
  - Reused from Papyrus
  - Extended for xtUML
- xtUML
  - Implementation as native UML2 profile

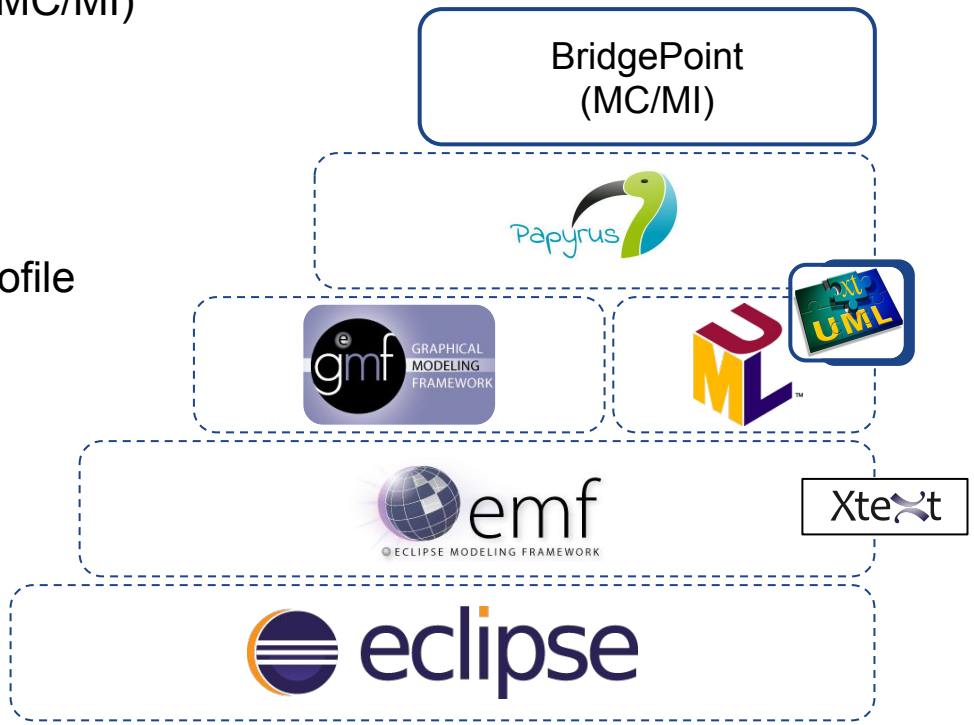
- Advantages
  - Papyrus UML2 editor reuse
  - Native Papyrus extension support
  - MC/MI can be reused
  - Support for partial migration
- Disadvantages
  - xtUML compatible API



UML2-based xtUML with native xtUML compatible API

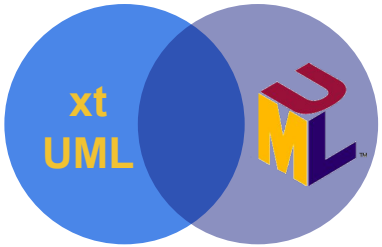
- BridgePoint
  - Model compilers and interpreters (MC/MI)
- Graphical editors
  - Reused from Papyrus
  - Extended for xtUML
- xtUML
  - Implementation as native UML2 profile

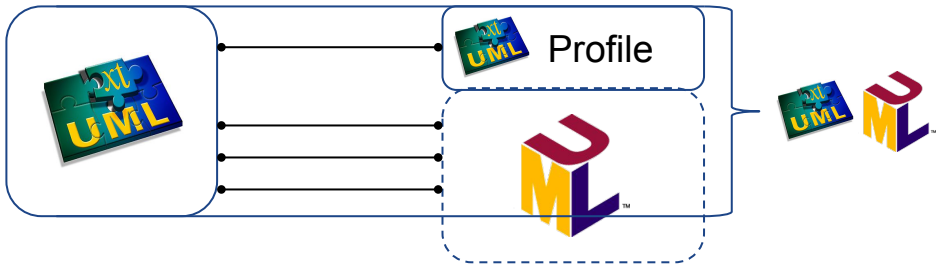
- Advantages
  - Papyrus UML2 editor reuse
  - Native Papyrus extension support
  - MC/MI can be reused
  - Support for partial migration
- Disadvantages
  - xtUML compatible API



UML2-based xtUML with native xtUML compatible API

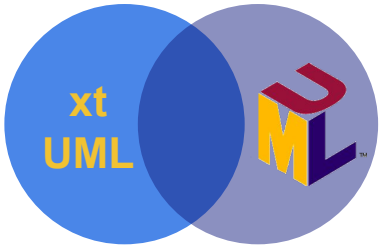
- UML2-based xtUML
  - Mapping of xtUML to UML2 → UML2 subset
  - Implementing the remainder of xtUML as a UML profile

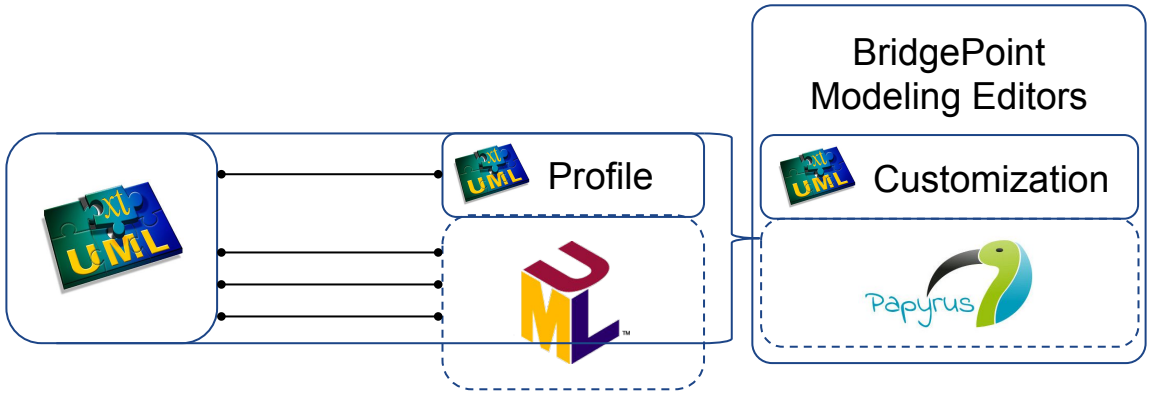




UML2-based xtUML with native xtUML compatible API

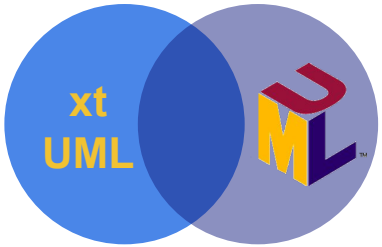
- UML2-based xtUML for BridgePoint Modeling editors
  - Papyrus modeling editors
  - Customization for UML2-based xtUML

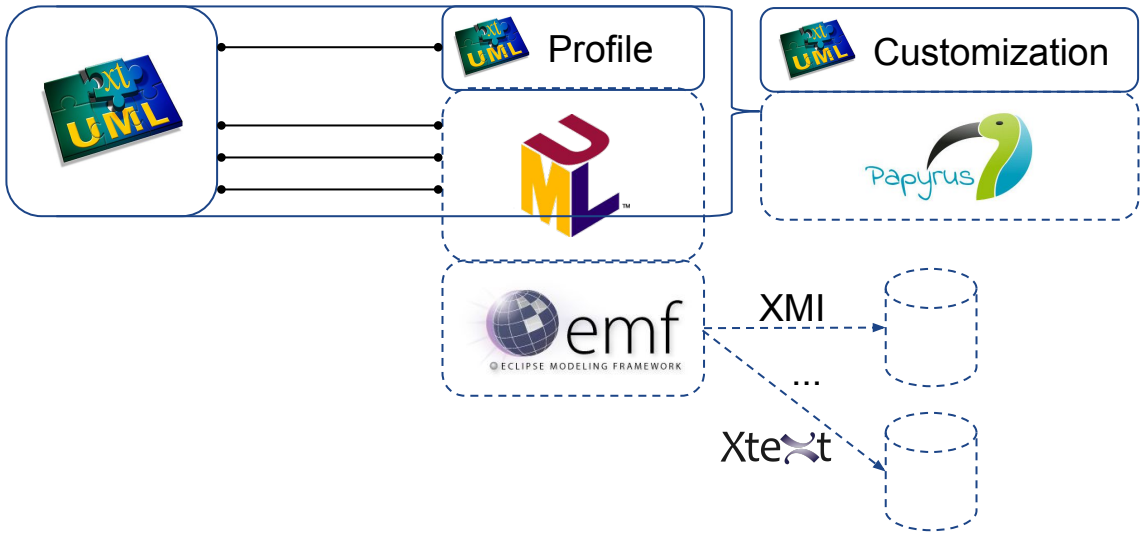




UML2-based xtUML with native xtUML compatible API

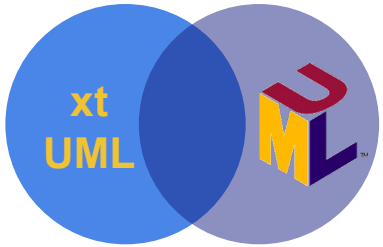
- UML2-based xtUML for persistence
  - EMF default format: XMI
  - Other existing persistence formats (json, CDO, …)
  - Optional custom formats possible (e.g., Xtext based)

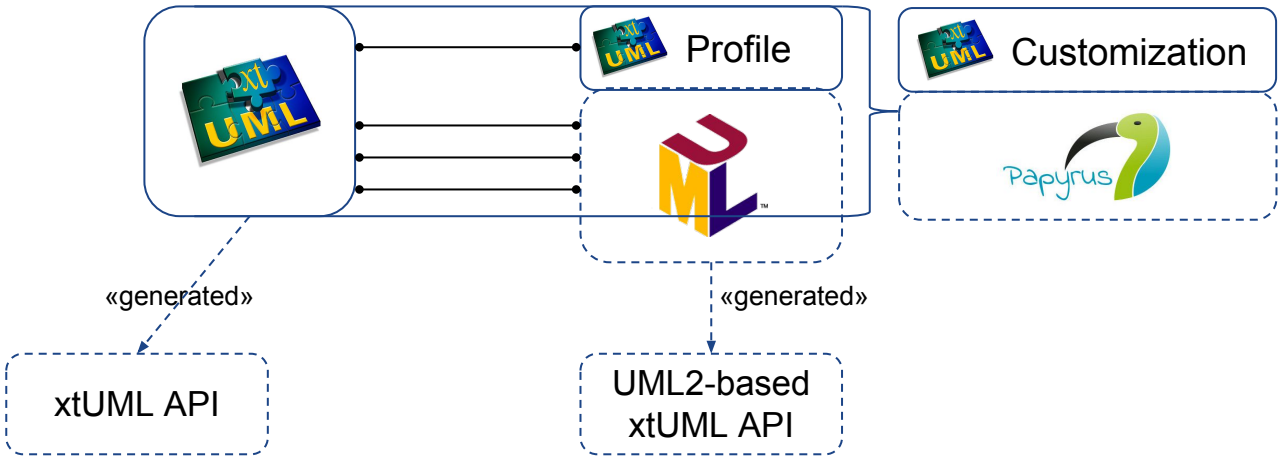




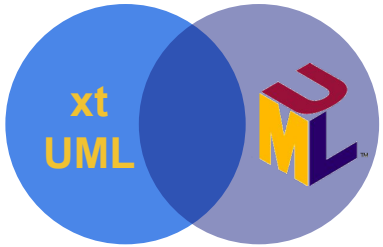
UML2-based xtUML with native xtUML compatible API

- Native xtUML API for BridgePoint MC/MI
  - Ecore metamodel can be generated from xtUML profile
  - Builds UML2-based metamodel API together with UML2 API
  - xtUML API can be generated from xtUML metamodel

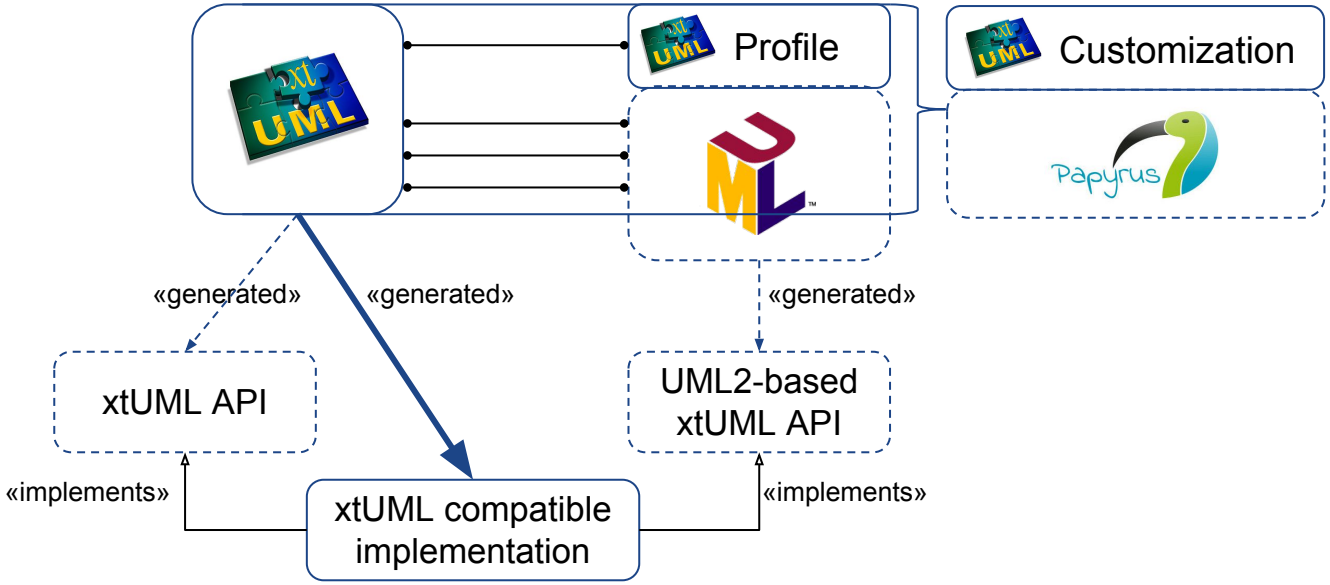




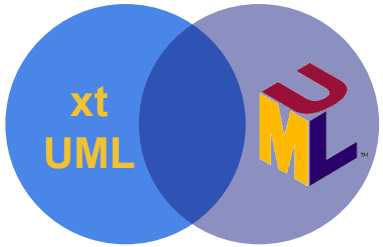
UML2-based xtUML with native xtUML compatible API



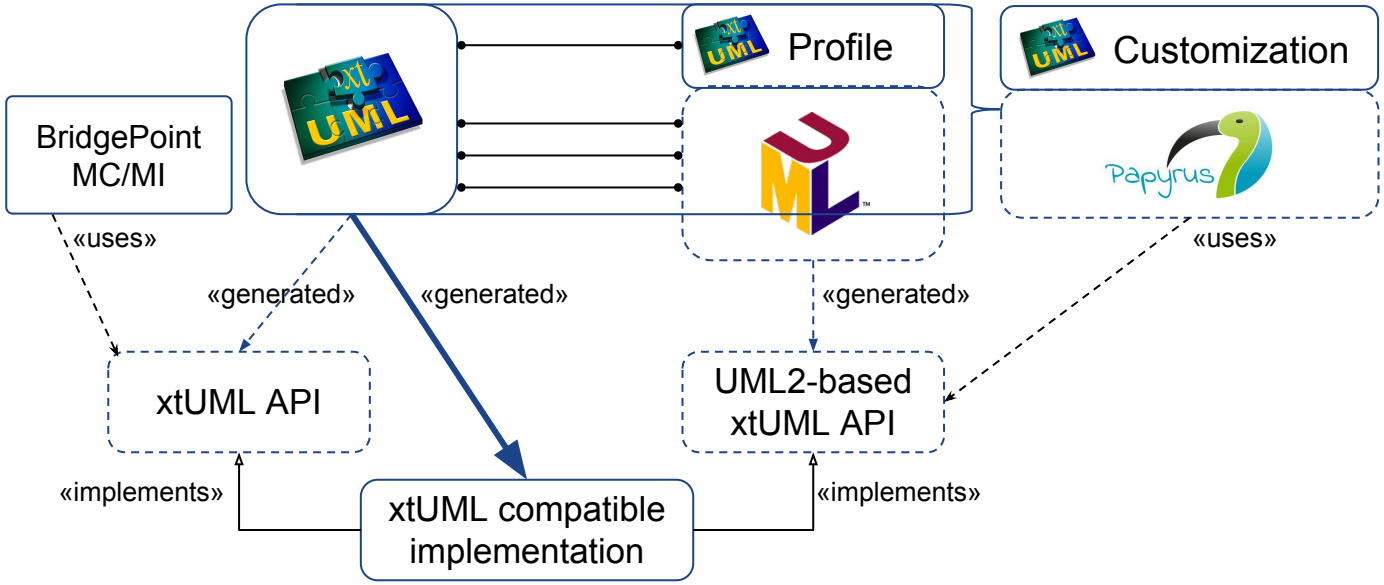
- Native xtUML API for BridgePoint MC/MI
  - Ecore metamodel can be generated from xtUML profile
  - Builds UML2-based metamodel API together with UML2 API
  - xtUML API can be generated from xtUML metamodel
  - Custom generator for xtUML compatible implementation of UML2 API



UML2-based xtUML with native xtUML compatible API

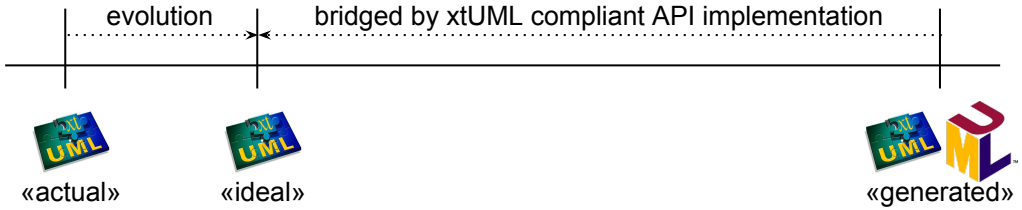


- Native xtUML API for BridgePoint MC/MI
  - Ecore metamodel can be generated from xtUML profile
  - Builds UML2-based metamodel API together with UML2 API
  - xtUML API can be generated from xtUML metamodel
  - Custom generator for xtUML compatible implementation of UML2 API



From the current xtUML API to the ideal xtUML API

- Optional transformation of xtUML metamodel to Ecore-based metamodel
  - Not necessary, but may be beneficial for maintenance in the long run
- Generation of xtUML API from xtUML metamodel (exists probably)
- Identifying the gap between xtUML API and UML2-based xtUML API
  - “Technical gap” → multiple occurrences of a few patterns (e.g. list access, …)
  - “Language gap” → can be derived from the mapping of xtUML to UML&profile
- Custom generator for generating the xtUML compliant implementation
- Distinguish between evolution and bridging
  - Slight API may be necessary
    → improve API
    → ease bridging



Next steps: Proof-of-concept implementations

Proof-of-concept implementation: xtUML-compatible implementation of two APIs

- Select an adequate metamodel partition from xtUML
- Map the model partition to UML2 and define UML profile for what could not be mapped
- Generate UML2-based xtUML API
- Manual xtUML-compatible implementation of UML2-based xtUML API

- *Analysis of technical gap and language gap*
- *Analysis of what can be automated in a generator and derived from mapping*

Proof-of-concept of partially migrated model

- *Load the model partition of an existing xtUML model*
- *Switch deserializer to instantiate xtUML-compliant implementation*
- *Save model with EMF XMI serializer*
- *Open serialized model with Papyrus*

Proof-of-concept: Identify metamodel partitioning

Migration Roadmap

Before migration

1. Define model partitions (risk: do they exist and how big are they?)
2. Define order of partitions to work on (risk: prevent late blocking issues)
3. Optional: migrate to Ecore-based xtUML metamodel

Iterate for each model partition

1. Extend xtUML profile
2. Extend xtUML-compatible implementation of UML2-based xtUML API
   a. Write tests with existing models
   b. Run generator
   c. Identify issues in the implementation
   d. Extend generator or add manual extensions
   e. Re-generate xtUML-compliant API implementation
3. Potentially adapt tool
   (e.g. switch from existing GEF-based editor to Papyrus; customize Papyrus)
4. Deploy implementation for model partition