MetaEdit+ Review

By Willer Travassos
260232775

Motivation behind its creation

- The principal driving motivation is to solve the common problems that plagues the development of software and systems. Namely:
  - Productivity affecting problems (e.g., unfair time schedules), and
  - Quality affecting problems

Motivation behind its creation

- The creators of MetaEdit+ looked for establishing a multi-method, multi-user, multi-tool environment
- Solving the problems that afflicted the CASE and CAME solutions that were existent at the time (early to mid 90's)
- Allowing developers to freely create tools and environments, while easily managing, re-using, integrating them

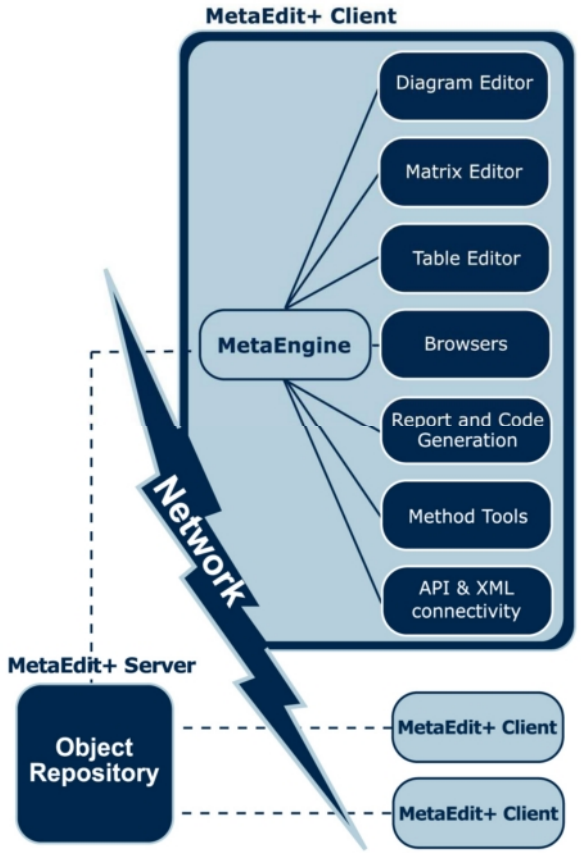
Software's Underlying Ideas

- MetaEdit+ covers five major problems that were present in similar tools:
  1. Lack of model integration and consistency check
  2. Lack of multi-user access to created objects, and models
  3. Lack of varied object representation (be it graphical, tabular, formulary, and etc)
  4. Being a true meta-modeling environment, where tools, systems and methods are developed using the same meta-modeling language. MetaEdit+ uses Graph-Object-Property-Port-Role-Relationship (GOPPRR)
  5. Lack of reporting techniques and limited information retrieval

Design Principles

- The architecture of MetaEdit+ is based on the following principles:
  - Object Orientation: enabling organization and re-use of software components in the environment, and interoperability between tools
  - Conceptual Modeling: objects contain aside from data, behavior and constraints on its relationships to other objects
  - Data Independence: tools in MetaEdit+ operate on data without knowing how it is stored
  - Representation Independence: models exist independently from their possible representations
  - Level independence: objects, methods, behaviors and other data are managed and manipulated in a similar way

General Architecture



- It centers around its MetaEngine
- The engine handles operations on the conceptual data, located at the Object Repository, according to the requests made by the tools that are part of the MetaEdit+ environment
- This architecture style allows:
  - easier integration of new tools in the environment, and
  - avoidance of duplicate manipulation code

General Architecture

- The Object Repository (OR) holds:
  - All data that is created using MetaEdit+.
  - Information about users and concurrent access (e.g., locking info, user passwords)
  - Information related to the specific visual representation of an object according to the tool invoking it
  - Output specifications of created languages
- Objects, behaviors, and general data are stored as GOPPRR concepts using the GOPPRR language (MetaEdit+'s modeling language)

MetaEdit+'s Environment

- The components/tools that form MetaEdit+ can be divided into five categories:
  1. Management tools: deals with basic software functionality
  2. Editing tools: used on the creation , manipulation, and deletion of models and its components. They can be form based or graphical based tools
  3. Retrieving tools: used to access models/objects in the OR and acquire its locks
  4. Annotation tools: used to add textual details, and/or and existing issues in a models,
  5. Method Management tools: used to retrieve and manipulate methods/behaviors of objects

The GOPPRR language

- The GOPPRR is the meta-metamodel of MetaEdit+. It stands for:
  - Graph: represents a model/container which includes a specific set of objects, relationships, roles, and properties
  - Object: represents an idea. Ex: a road segment
  - Property: defines attributes of an object or a relationship. Ex: a person's family name
  - Port: defines which types of connection are allowed between objects. Ex: in and out ports of a road segment
  - Role: represent how objects interact in a given relationship
  - Relationship: defines how objects should associate with each other.

The GOPPRR language

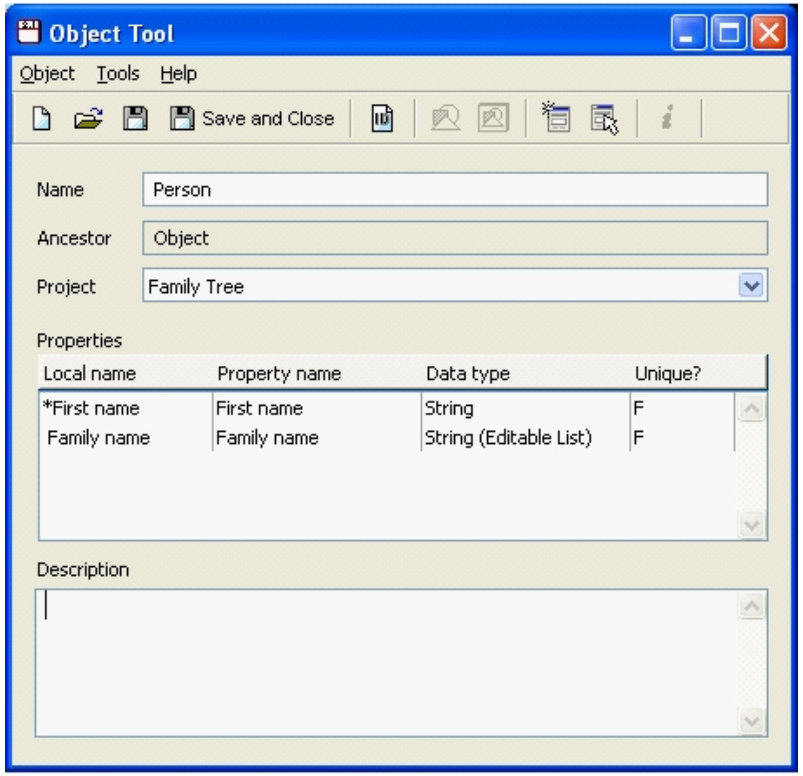
- The use of object orientation allows GOPPRR to use: generalization, specialization, and polymorphism, which lets a programmer create complex object behaviors in a fast and re-usable manner
- Integrity rules allows a programmer to add constraints on how properties should be used
- Method Integration is used to maintain a consistent representation of an object, or method, or property that is used on different graphs, i.e., a change in a property should be seen in all graphs, objects, and relationships that uses it

Creating a meta-model

- This example deals with generating a simple genealogical tree language, called Family tree, and a report
- Keep in mind that MetaEdit+ is capable of code generation, but this example is simple enough that we need it just to create our report
- In this example forms will be used to create our meta-model, instead of graphical representations

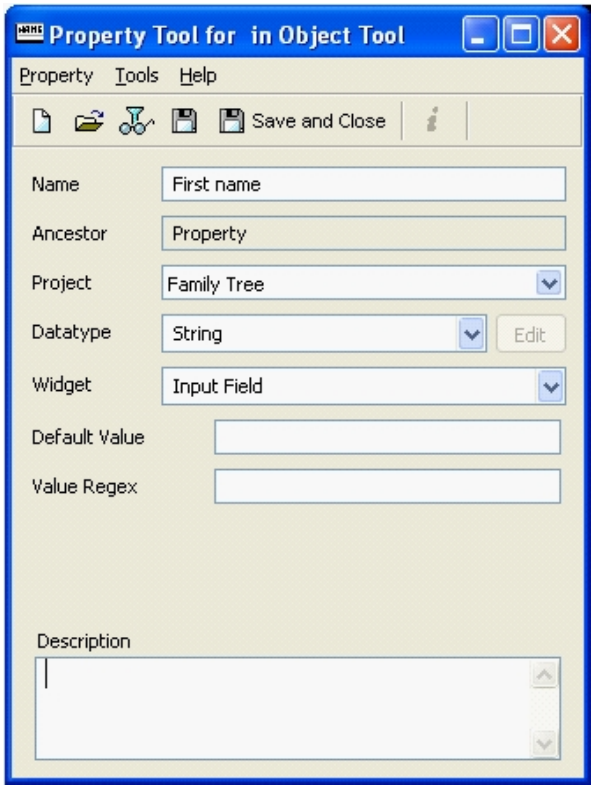
Creating a meta-model

- MetaEdit+ has 5 types of forms that are used to create models. They are: the Object tool, Relationship tool, Role tool, and Graph tool
- The first step is to create objects that are part of the model. Families are made of people, thus we need to represent them in our new language



Creating a meta-model

- The object tool provides means to create objects and add properties to it. In this case, people have names and last names, thus they become properties of the person object



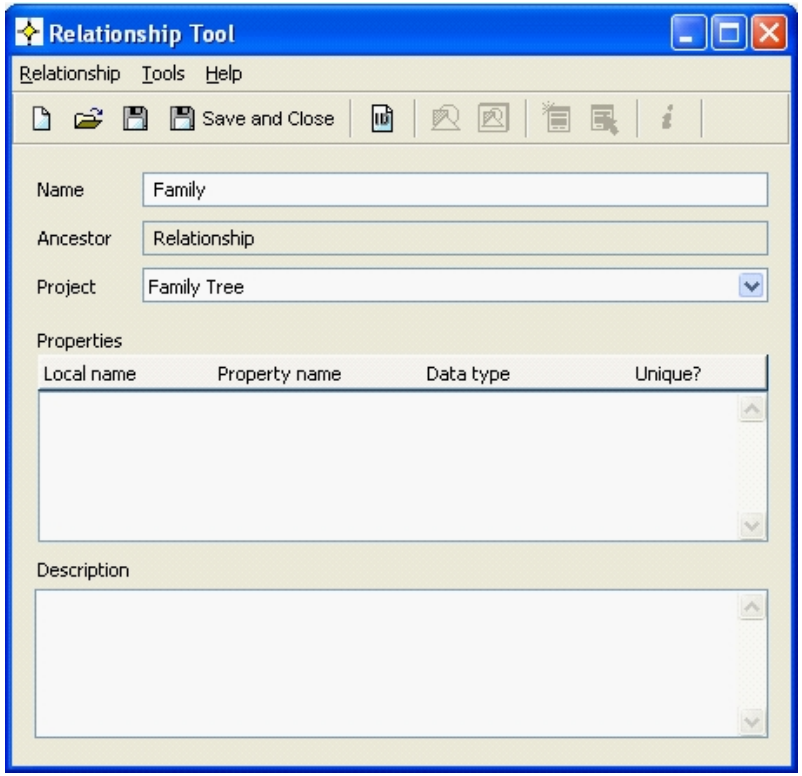
Creating a meta-model

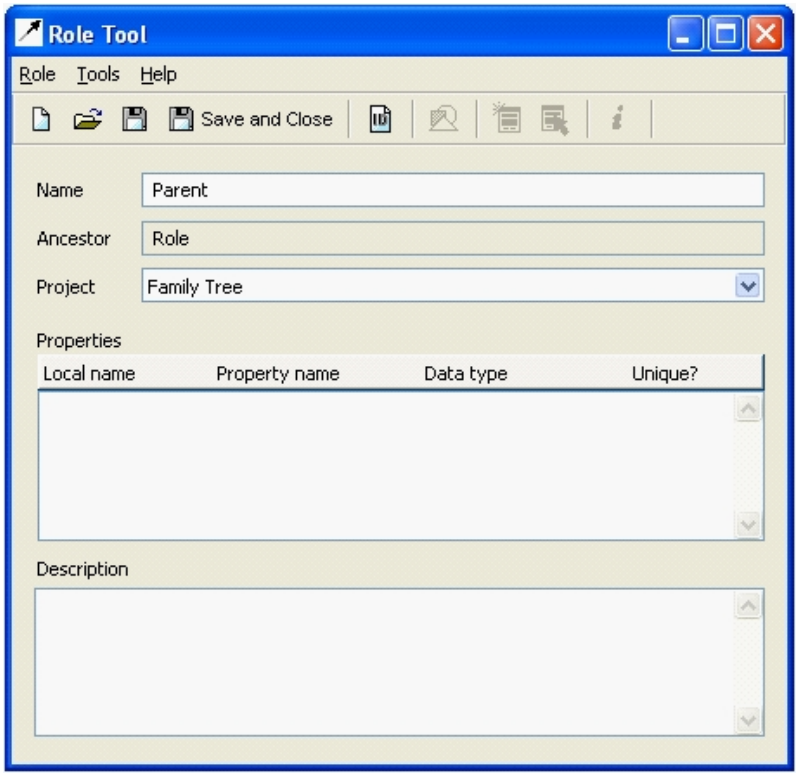
- The object tool also provides a Symbol Editor that allows a programmer to create representation of objects. They might contain text fields that display the contents of an object
- Here we define ports in an object, and determine its type.
- The Symbol Editor provides support for conditional representation, i.e., an object may contain display conditions that change the visual representation of an object. Ex: determining whether a person is deceased or not in or Family tree

Creating a meta-model

- Now that people are represented in our model, we need:
  - to determine relationship between them (with the Relationship tool) that will bind family members together, and
  - state the roles that each person takes in that family (using the Role Tool)
- To maintain simplicity, we just create Parent and Child roles

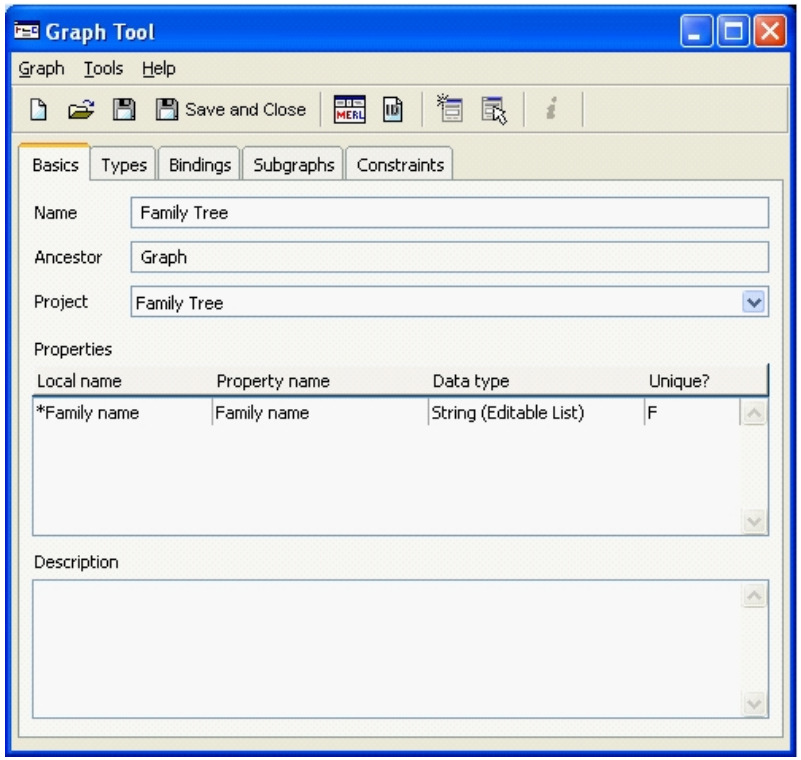
Creating a meta-model





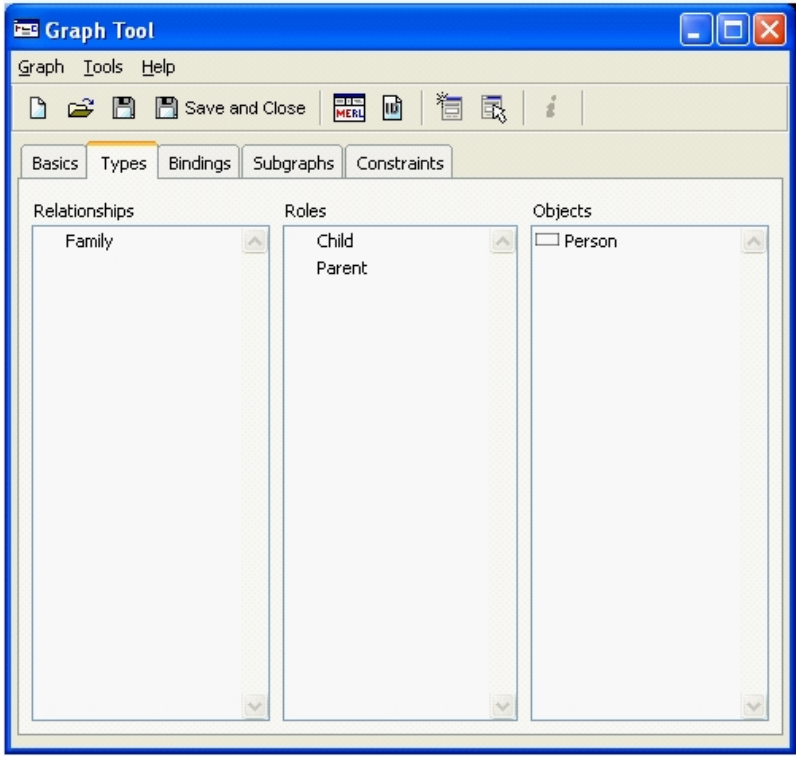
Creating a meta-model

- Having created fragments of our family model set, we just need to group them together. This is where the graph part of MetaEdit+ marches in
- The Graph tool will set how objects can be connected, and what types an object can become



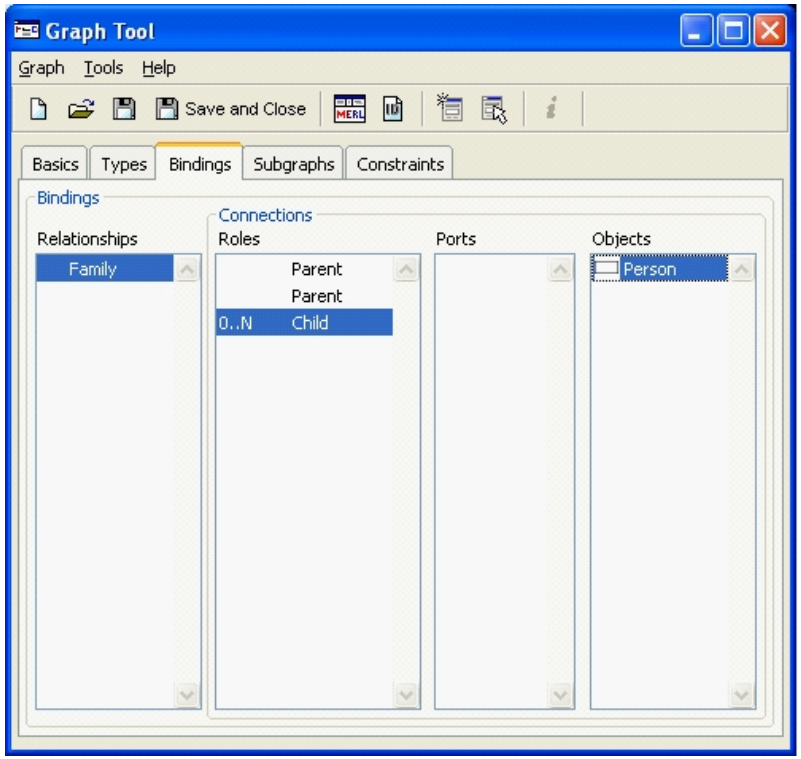
Creating a meta-model

- To define object types we click on the Types tab in the graph tool
- Then we add objects that we want to take part in this model and the roles/types the object have take



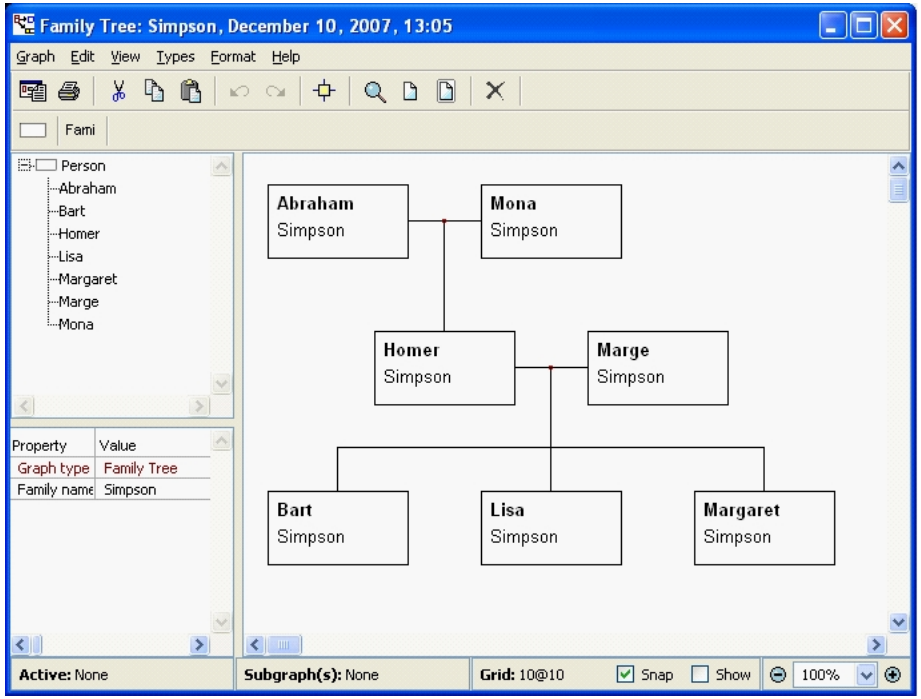
Creating a meta-model

- Finally we can we determine how different roles are related to one another by setting a simple hierarchical structure
- Parents have children and children may become parents and have children of their own
- In this model we part from the principle that a child can only have 2 parents, and parent can have zero or more children (this is set in the cardinality menu)



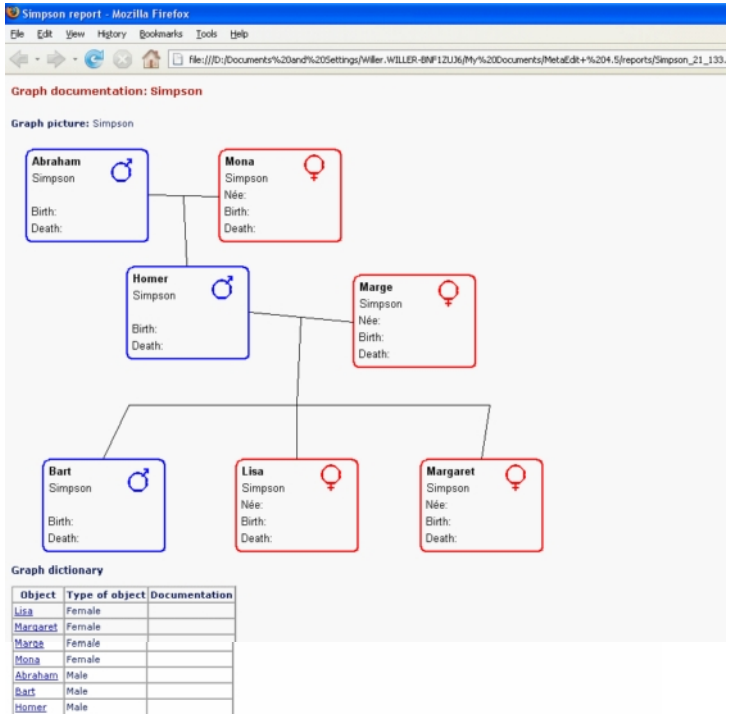
Using the meta-model

- After these steps our simple meta-model is created
- We can use the Create Graph Editor to generate our model



Producing output

- We can produce generators/reports with MetaEdit+ to represent data in models
- Generators are created using MetaEdit+'s script language called MERL



Pros and Cons

- Pros:
  - Documentation provides great support with diversified examples
  - Simple to install and use
  - Well integrated, supports all main platforms
- Cons:
  - Closed source, not enough papers about it
  - Very Expensive, the Workbench tool, which just allows meta-model creation, costs 9500 euros!, plus a 20% current price annual fee to maintain a license. The price of the full solution reaches 17000 euros.

References

- Kelly, S., Lyytinen, K., and Rossi, M., "MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment", Proceedings of CAiSE'96, 8th Intl. Conference on Advanced Information Systems Engineering, Lecture Notes in Computer Science 1080, Springer-Verlag, pp. 1-21, 1996
- Juha-Pekka Tolvanen, Risto Pohjonen, Steven Kelly, "Advanced Tooling for Domain-Specific Modeling: MetaEdit+", 7th OOPSLA workshop on Domain-Specific Modeling, pp. 1-8, 2007
- MetaEdit+ 4.5 User's Manuals, MetaCase, March 2008 (http://www.metacase.com/support/45/manuals/index.html)

Finally…

- Questions?