User’s Guide to the GME Publications Database Paradigm: PubDB
Version 5.7.20

http://www.eecs.berkeley.edu/~sprinkle/useful/pubdb/

Jonathan Sprinkle, Ph.D.
sprinkle@acm.org

11 July 2005

License and Lawyertalk

Contents

1 Introduction

One of the most depressing things to see on a researchers webpage is that his or her last publication was about 3 years before today's date. Most of us realize that this is seldom a representative list of publications, but at the same time, we ask "how is it that this list can go so out of date?"

The answers vary, but mostly can be traced back to the issue of time constraints. That is, writing papers is much more important than keeping an updated webpage—especially for more advanced professors and researchers who have made their impact prior to web-based reporting, and view webpage updates as a occasional nuisance rather than a forum for announcing activities. Thus, these banal tasks are either occasionally done, or continuously put off.

The papers themselves (hopefully) do not fall victim to this classic blunder of disorganization and chaos. Most good researchers write good papers, and the standard paper-writing tool is definitively the LaTeX toolbox and its many free distributions and implementations. The astute reader will have, by now, realized that it would be wonderful to immediately harvest the technical information regarding recent publications into both a formal publishable form (technical journals/conferences, annual reports, *curriculum vitae*) as well as an informal one (webpages). That is exactly what the PubDB tool aims to do.

1.1 Overview

It's a bore to have to keep reworking your webpage and C.V. each time you publish a new paper. Furthermore, if you're a LaTeX user, you are already tired of maintaining your BibTeX database in such an unwieldy way, and have either given up on maintaining individual databases, or are frustrated with looking through your one huge one to get key names, and hesitant to give it out to other people.

The PubDB GME Paradigm is designed to alleviate some of these problems. It is a domain-specific modeling environment—based on the publications ontology of BibTeX—which provides a graphical interface to your publications collections. It allows you to build a publication database for each paper you write, and provides the ability to enter publication data once, and "point" to that publication arbitrarily many times for future papers. You can organize publications by area/author/etc., and keep "pointers" to publications anywhere. In the spirit of BibTeX, it has a special attribute listing for each kind of publication, relieving you of the need to remember which fields each kind of publication has, and visually reminding you that some publication parameters are optional, while others are required.

The most important piece of PubDB is its ability to generate artifacts which are useful to the publishing domain. You can create from your GME database the following kinds of artifacts:

- A BibTeX *.bib file
- A LaTeX source file that cites each entry in your *.bib (useful for generating text to insert into a LaTeX-based C.V. or annual report).
- An HTML file, configurable by a CSS class file

The trick to the PubDB tool is that it uses the philosophy of *domain-specific modeling* to restrict the input space of your database to what is useful for the domain of publications. The tool utilizes the GME meta-programmable modeling environment, and C++ output generators which produce the output artifacts useful for you. In the remainder of this section, we describe some of the basic concepts of domain-specific modeling and the GME tool itself.

1.2 Domain-Specific Modeling

1.2.1 The Generic Modeling Environment (GME)

The Generic Modeling Environment[1] is a configurable toolkit for creating domain-specific modeling and program synthesis environments. The configuration is accomplished through metamodels specifying the modeling paradigm (modeling language) of the application domain. The modeling paradigm contains all the syntactic, semantic, and

[1]This section taken largely from the GME website: http://www.isis.vanderbilt.edu/projects/gme/

presentation information regarding the domain; which concepts will be used to construct models, what relationships may exist among those concepts, how the concepts may be organized and viewed by the modeler, and rules governing the construction of models. The modeling paradigm defines the family of models that can be created using the resultant modeling environment.

The metamodeling language is based on the UML class diagram notation and OCL constraints. The metamodels specifying the modeling paradigm are used to automatically generate the target domain-specific environment. The generated domain-specific environment is then used to build domain models that are stored in a model database or in XML format. These models are used to automatically generate the applications or to synthesize input to different COTS analysis tools.

GME has a modular, extensible architecture that uses MS COM for integration. GME is easily extensible; external components can be written in any language that supports COM (C++, Visual Basic, C#, Python etc.). GME has many advanced features. A built-in constraint manager enforces all domain constraints during model building. GME supports multiple aspect modeling. It provides metamodel composition for reusing and combining existing modeling languages and language concepts. It supports model libraries for reuse at the model level. All GME modeling languages provide type inheritance. Model visualization is customizable through decorator interfaces.

Since GME is a meta-configurable environment, downloading and opening GME provides you only with the interface an unconfigured modeling environment. In order to take advantage of a *domain-specific* modeling environment you must create a metamodel; it defines the abstract syntax of the modeling language, and generates a configuration file for GME.

1.3 Importing models

Once you have registered your paradigm and the two components for generating BibTeX and HTML, you are ready to do one of two things: build your own models, or look to see how someone else has done so by importing examples.

2 Getting Started

In order to use the PubDB paradigm, you will need to have

1. Microsoft Windows XP, or newer/equivalent,
2. A working copy of GME, GME5, v.5.6.8, or take a chance that the files will work with your version (possible, if the major version is the same, but minor version is newer), and
3. The latest release of PubDB (5.7.20 for this manual) and its associated binary files or source code.

If you are wondering whether or not this works under an operating system other than Micrsoft Windows, it does not. At this time, GME is based heavily on Microsoft COM, and as such is not portable to other operating systems. If you have access to a Windows Terminal Server 2003 or equivalent, PubDB has been tested on that environment, and can be accessed via `rdesktop` under Linux to create a terminal session, and take advantage of the GME modeling tool and PubDB paradigm.

2.1 Installing GME

Please refer to the GME installation instructions. Please uninstall your version of GME if you are upgrading to the GME5/v.5.6.8 version for purposes of this paradigm. Once GME is installed, proceed to Sect. 2.2.

2.2 Installing PubDB

If you have not yet downloaded the binaries, they are available from `http://www.eecs.berkeley.edu/~sprinkle/useful/pubdb/`, available as a .zip file. Please extract the contents of this file somewhere on your computer, where they will be placed in their own directory. I recommend that you place them in the `$GME_ROOT/Contrib` directory, where they will appear as `$GME_ROOT/Contrib/PubDB_v5.7.20` . [2]

2.3 Registering your paradigm and components

In order to utilize the PubDB modeling environment, we must register the PubDB paradigm with GME so it will know the kinds of objects you can create.

1. Open GME, as shown in Fig. 1.
2. Choose `File`→`Register Paradigms...`, to reveal the dialog shown in Fig. 2.
3. Select `Register: For both` in the lower right corner. If you wish for the paradigm to be registered only to you as the user, then leave the selection as `Register: For user only`. However, it is recommended that you register the paradigm and components systemwide, in the event that other users later want to use the PubDB environment.
4. Choose `Add from File...`, select the paradigm, in file `$GME_ROOT/Contrib/PubDB_v5.7.20/meta/PublicationsDatabase.xmp`, and select `Open`.
5. While you have the PublicationsDatabase paradigm selected, choose `Components...`.
6. Select the `Register: Systemwide` radio button.
7. Choose `Install New...`, select the component, in file `$GME_ROOT/Contrib/PubDB_v5.7.20/bin/GenerateBibTeXDB.dll`

[2]Downloading the source: The source for the projects is not easily available for download at this time. However, the author is pleased to send it to you if you would like to look at his terribly hacked code. The source is not really necessary to run, only to recompile.

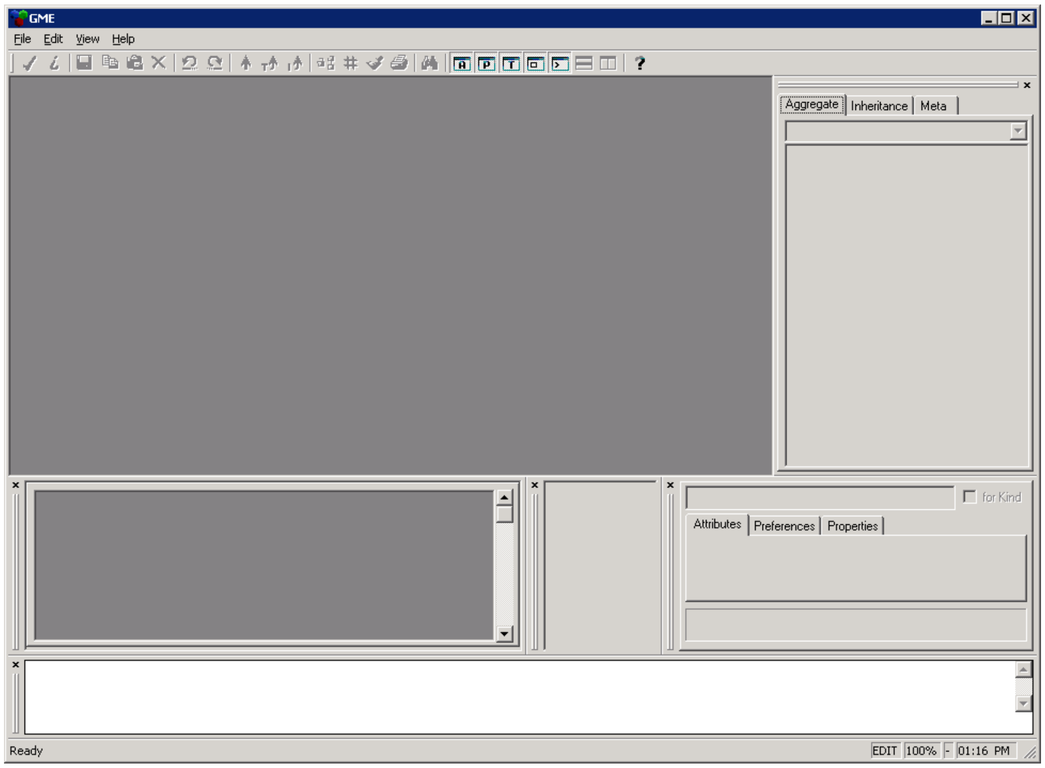
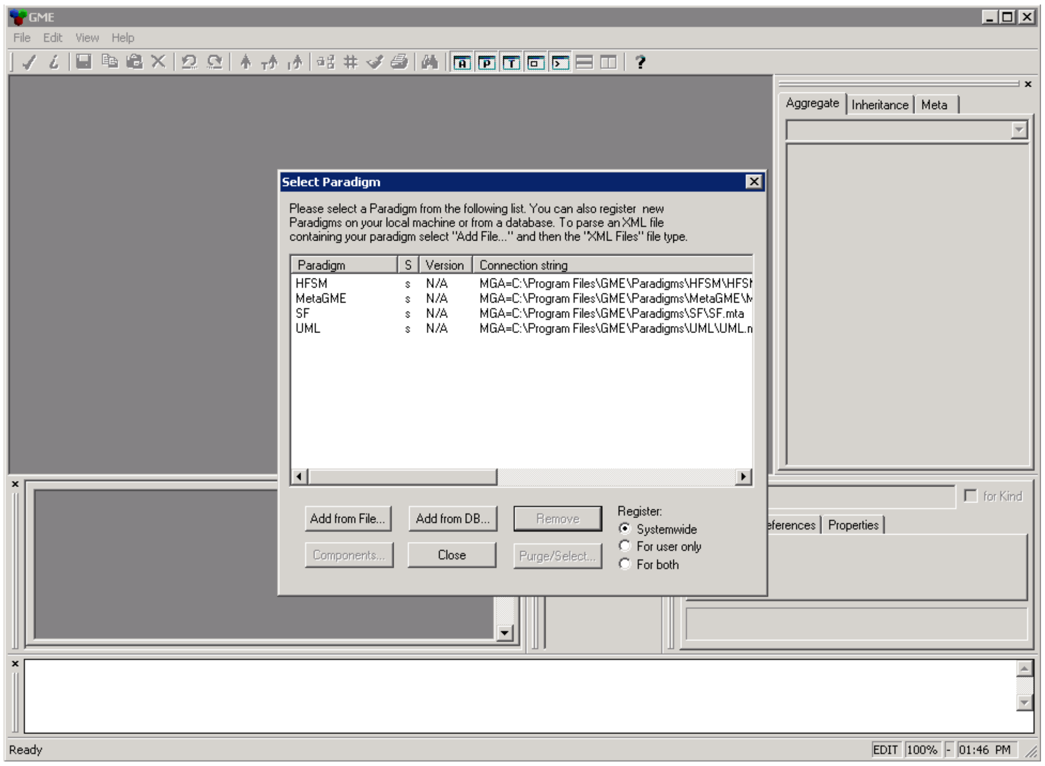


Figure 1: The Generic Modeling Environment, unconfigured.

8. Choose `Install New...`, select the component, in file `$GME_ROOT/Contrib/PubDB_v5.7.20/bin/GenerateHTML.dll`
9. Choose `Close`
10. Choose `Close`

You should now see the blank GME window again (as in Fig. 1). Now, if you have used the paradigm before, are extremely anxious, or simply want to live on the edge, you are ready to start creating publications databases, which is explained in Sect. 4. However, if you would like to understand some background on the *kinds* of objects which are available for the PubDB paradigm, the next section explains the many types available in the modeling environment.



Please select a Paradigm from the following list. You can also register new Paradigms on your local machine or from a database. To parse an XML file containing your paradigm select "Add File..." and then the "XML Files" file type.

Paradigm	S	Version	Connection string
HFSM	s	N/A	MGA=C:\Program Files\GME\Paradigms\HFSM\HFSI
MetaGME	s	N/A	MGA=C:\Program Files\GME\Paradigms\MetaGME\M
SF	s	N/A	MGA=C:\Program Files\GME\Paradigms\SF\SF.mta
UML	s	N/A	MGA=C:\Program Files\GME\Paradigms\UML\UML.n

Register: [✓ Systemwide] For user only For both

Figure 2: Adding the PublicationsDatabase paradigm.

3 Understanding PubDB Kinds

The GME environment is an extremely versatile tool for domain-specific modeling. It is configured through a *metamodel*, which defines all of the possible constructs which can be created for a particular model (i.e., the *abstract syntax*). The nodes of the abstract syntax tree are made up of *kinds*, which are the types of objects that can be used in the model.

3.1 Domain-specific philosophy

In the domain-specific modeling philosophy, a particular modeling environment is considered *domain-specific* if it carries special relevance for an expert in a particular domain. In the engineering world, examples of domain-specific modeling environments are MATLAB Simulink (for the domain of systems simulation), SPICE (for the domain of circuit analysis), AutoCAD (for the domain of technical drawing), and—believe it or not—Microsoft PowerPoint (for audio/visual presentations). Each of these tools is made so that it is easier to use the tool/application for a particular purpose, and the way that this is made simple is through *kinds*.

For example, the MATLAB Simulink tool provides domain-specific concepts such as function generators, gain multipliers, and connections between them, to allow a user to visually create a model of a system for simulation. For an engineer who understands the concepts of a function generator and the other kinds of signal sources, the blocks make immediate sense, and almost require no explanation—the user simply modifies the attributes of the object and creates an electronic representation of what s/he expects to see in "real life." The goal of domain-specific modeling is to create such an environment for the necessary domains—in our case, the domain of publications.

3.2 Choosing domain concepts

Usually domain concepts are apparent based on the nomenclature, physical existence, and/or body of research for a particular domain. Once the optimal[3] set of domain concepts has been enumerated it is the job of the metamodeler to encode those domain concepts into a language.

The choice of the appropriate modeling concept for each domain concept is key to successfully encoding the domain concepts. Domain concepts that show properties of tree-like containment should utilize hierarchical structures. Domain concepts that show properties of graph-like interconnectedness should utilize (typed) associations between concepts. The method of entry for the domain user should be efficient, as well. For example, rather than have hundreds of types of similar objects from which to choose, it is more efficient to have one type with an enumerated list of attributes to choose the specifics for this type. This reduces the set from which a domain expert may choose domain concepts, providing a less cluttered beginning point (recall that we want a domain expert to use this environment without a manual).

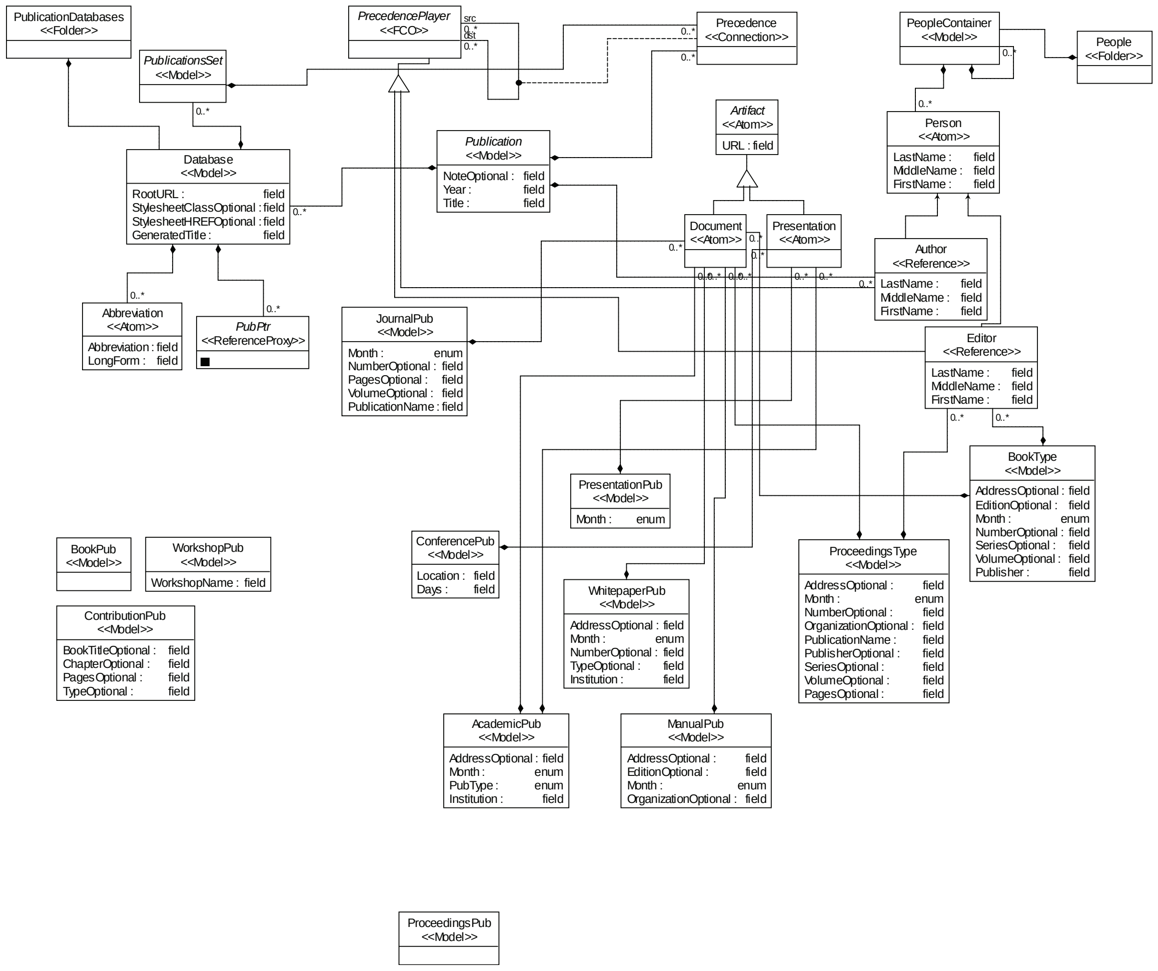
These examples of canonical abstraction methods are important, but are not so different from normal software abstraction when creating a class hierarchy and deciding whether to associate objects by containment, or by reference. What makes encoding domain concepts into a modeling language more interesting—and more useful when done correctly—is the ability to reflect the common notations and representations in the domain. This means that while abstraction using some modeling concept may actually be less efficient than a clever abstraction, the one that "looks" more like the notation of the domain expert is preferred.

Finally, metamodelers (like software architects) should keep in mind the possible future evolution of the domain when creating the design. Using inheritance concepts, it is possible to create abstractions of types in the metamodel which can be extended in future generations of the language (e.g., a generic type "network adapter" which can be specialized in future generations of the language by adding new types) while relying on common interfaces as defined by the generic type.

3.3 Drawing on experience: BibTeX

The purpose of this domain is to allow researchers/authors to create database artifacts for use with publishing and biographical tools. The main way in which the tool will be used is to generate a BibTeX database of selected pub-

[3]Here, optimal means that the domain-expert is satisfied with the level of abstraction of the domain concepts.



PublicationDatabases <<Folder>>
PublicationsSet <<Model>>
PrecedencePlayer <<FCO>>
Precedence <<Connection>>
PeopleContainer <<Model>>
People <<Folder>>
Person <<Atom>> LastName : field MiddleName : field FirstName : field
Artifact <<Atom>> URL : field
Publication <<Model>> NoteOptional : field Year : field Title : field
Database <<Model>> RootURL : field StylesheetClassOptional : field StylesheetHREFOptional : field GeneratedTitle : field
Document <<Atom>>
Presentation <<Atom>>
Author <<Reference>> LastName : field MiddleName : field FirstName : field
Editor <<Reference>> LastName : field MiddleName : field FirstName : field
Abbreviation <<Atom>> Abbreviation : field LongForm : field
PubPtr <<ReferenceProxy>>
JournalPub <<Model>> Month : enum NumberOptional : field PagesOptional : field VolumeOptional : field PublicationName : field
BookType <<Model>> AddressOptional : field EditionOptional : field Month : enum NumberOptional : field SeriesOptional : field VolumeOptional : field Publisher : field
PresentationPub <<Model>> Month : enum
BookPub <<Model>>
WorkshopPub <<Model>> WorkshopName : field
ConferencePub <<Model>> Location : field Days : field
WhitepaperPub <<Model>> AddressOptional : field Month : enum NumberOptional : field TypeOptional : field Institution : field
ProceedingsType <<Model>> AddressOptional : field Month : enum NumberOptional : field OrganizationOptional : field PublicationName : field PublisherOptional : field SeriesOptional : field VolumeOptional : field PagesOptional : field
ContributionPub <<Model>> BookTitleOptional : field ChapterOptional : field PagesOptional : field TypeOptional : field
AcademicPub <<Model>> AddressOptional : field Month : enum PubType : enum Institution : field
ManualPub <<Model>> AddressOptional : field EditionOptional : field Month : enum OrganizationOptional : field
ProceedingsPub <<Model>>
src
dst
0..*

Figure 3: Structure of the kinds of objects in PubDB.

lications (i.e., a portion of—not the entire—database). Also, generation of HTML formatted text would enable a modeler to generate listings of items published by themselves (or another person). This means that the language should not be a model of BibTeX, but instead should be a model of *publications*; however, it is true that the BibTeX specification is an authoritative (and accepted) categorization of many kinds of publication classifications. Therefore, it is possible to take advantage of the work that hundreds of researchers have put in to the BibTeX domain to define the full set of classifications, while at the same time defining the PubDB language to be somewhat independent of BibTeX as an input format.

3.4 Kinds of objects in PubDB

The rest of this section describes textually the relationships and attributes available for the PubDB paradigm. This is explained visually in Figures 3, 4 and 5.

3.4.1 `Abbreviation`

The `Abbreviation` type is somewhat experimental—check your work when using it.

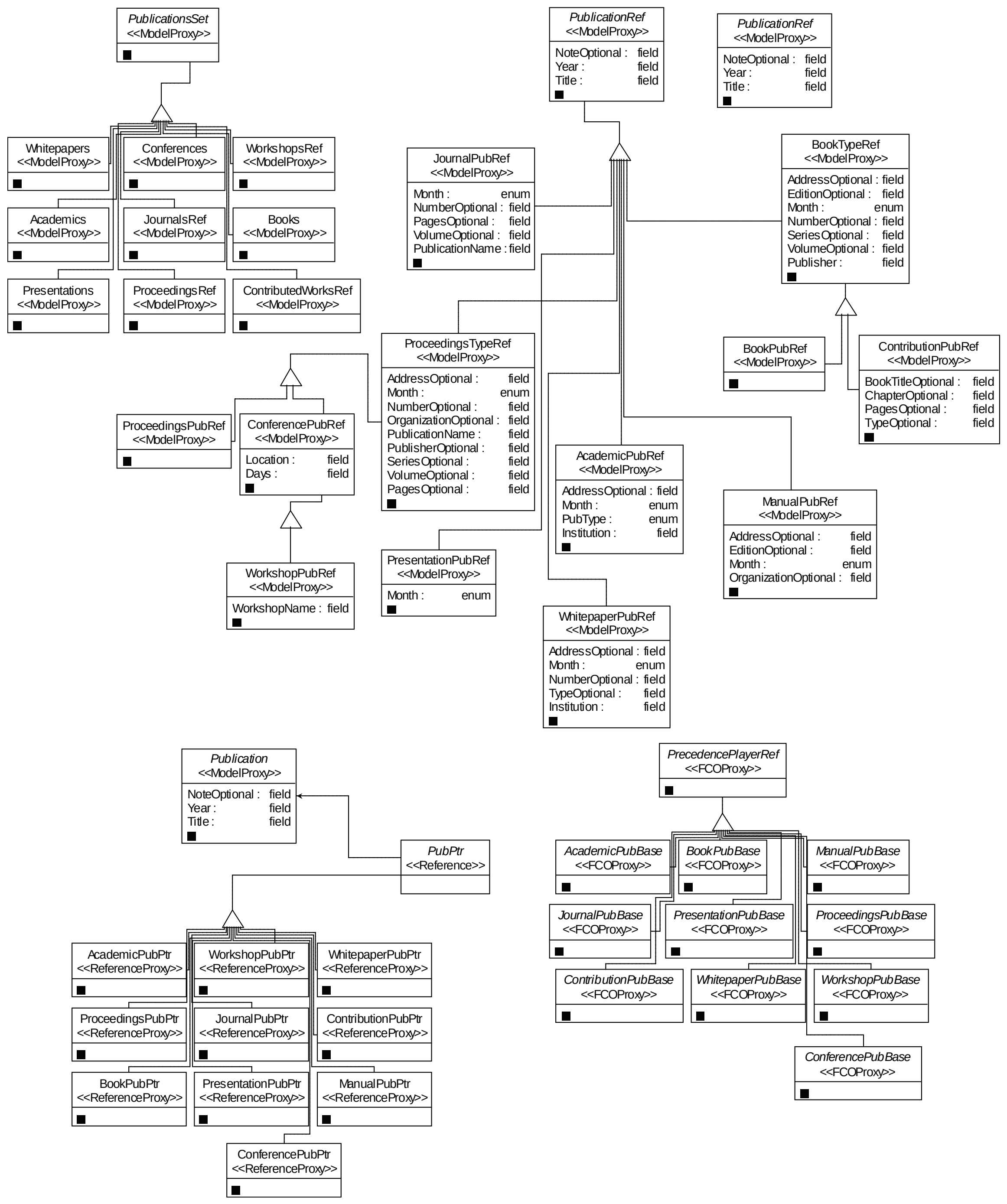

PublicationsSet <<ModelProxy>>
Whitepapers <<ModelProxy>>
Conferences <<ModelProxy>>
WorkshopsRef <<ModelProxy>>
Academics <<ModelProxy>>
JournalsRef <<ModelProxy>>
Books <<ModelProxy>>
Presentations <<ModelProxy>>
ProceedingsRef <<ModelProxy>>
ContributedWorksRef <<ModelProxy>>
PublicationRef <<ModelProxy>>
NoteOptional : field
Year : field
Title : field
PublicationRef <<ModelProxy>>
NoteOptional : field
Year : field
Title : field
JournalPubRef <<ModelProxy>>
Month : enum
NumberOptional : field
PagesOptional : field
VolumeOptional : field
PublicationName : field
BookTypeRef <<ModelProxy>>
AddressOptional : field
EditionOptional : field
Month : enum
NumberOptional : field
SeriesOptional : field
VolumeOptional : field
Publisher : field
ProceedingsTypeRef <<ModelProxy>>
AddressOptional : field
Month : enum
NumberOptional : field
OrganizationOptional : field
PublicationName : field
PublisherOptional : field
SeriesOptional : field
VolumeOptional : field
PagesOptional : field
BookPubRef <<ModelProxy>>
ContributionPubRef <<ModelProxy>>
BookTitleOptional : field
ChapterOptional : field
PagesOptional : field
TypeOptional : field
ProceedingsPubRef <<ModelProxy>>
ConferencePubRef <<ModelProxy>>
Location : field
Days : field
AcademicPubRef <<ModelProxy>>
AddressOptional : field
Month : enum
PubType : enum
Institution : field
ManualPubRef <<ModelProxy>>
AddressOptional : field
EditionOptional : field
Month : enum
OrganizationOptional : field
WorkshopPubRef <<ModelProxy>>
WorkshopName : field
PresentationPubRef <<ModelProxy>>
Month : enum
WhitepaperPubRef <<ModelProxy>>
AddressOptional : field
Month : enum
NumberOptional : field
TypeOptional : field
Institution : field
Publication <<ModelProxy>>
NoteOptional : field
Year : field
Title : field
PubPtr <<Reference>>
AcademicPubPtr <<ReferenceProxy>>
WorkshopPubPtr <<ReferenceProxy>>
WhitepaperPubPtr <<ReferenceProxy>>
ProceedingsPubPtr <<ReferenceProxy>>
JournalPubPtr <<ReferenceProxy>>
ContributionPubPtr <<ReferenceProxy>>
BookPubPtr <<ReferenceProxy>>
PresentationPubPtr <<ReferenceProxy>>
ManualPubPtr <<ReferenceProxy>>
ConferencePubPtr <<ReferenceProxy>>
PrecedencePlayerRef <<FCOProxy>>
AcademicPubBase <<FCOProxy>>
BookPubBase <<FCOProxy>>
ManualPubBase <<FCOProxy>>
JournalPubBase <<FCOProxy>>
PresentationPubBase <<FCOProxy>>
ProceedingsPubBase <<FCOProxy>>
ContributionPubBase <<FCOProxy>>
WhitepaperPubBase <<FCOProxy>>
WorkshopPubBase <<FCOProxy>>
ConferencePubBase <<FCOProxy>>

Figure 4: The definition of objects in PubDB draws heavily upon the object-oriented philosophy of inheritance, in order to reduce the complexity of the paradigm, and also to reduce the possibility of error during metamodel design.

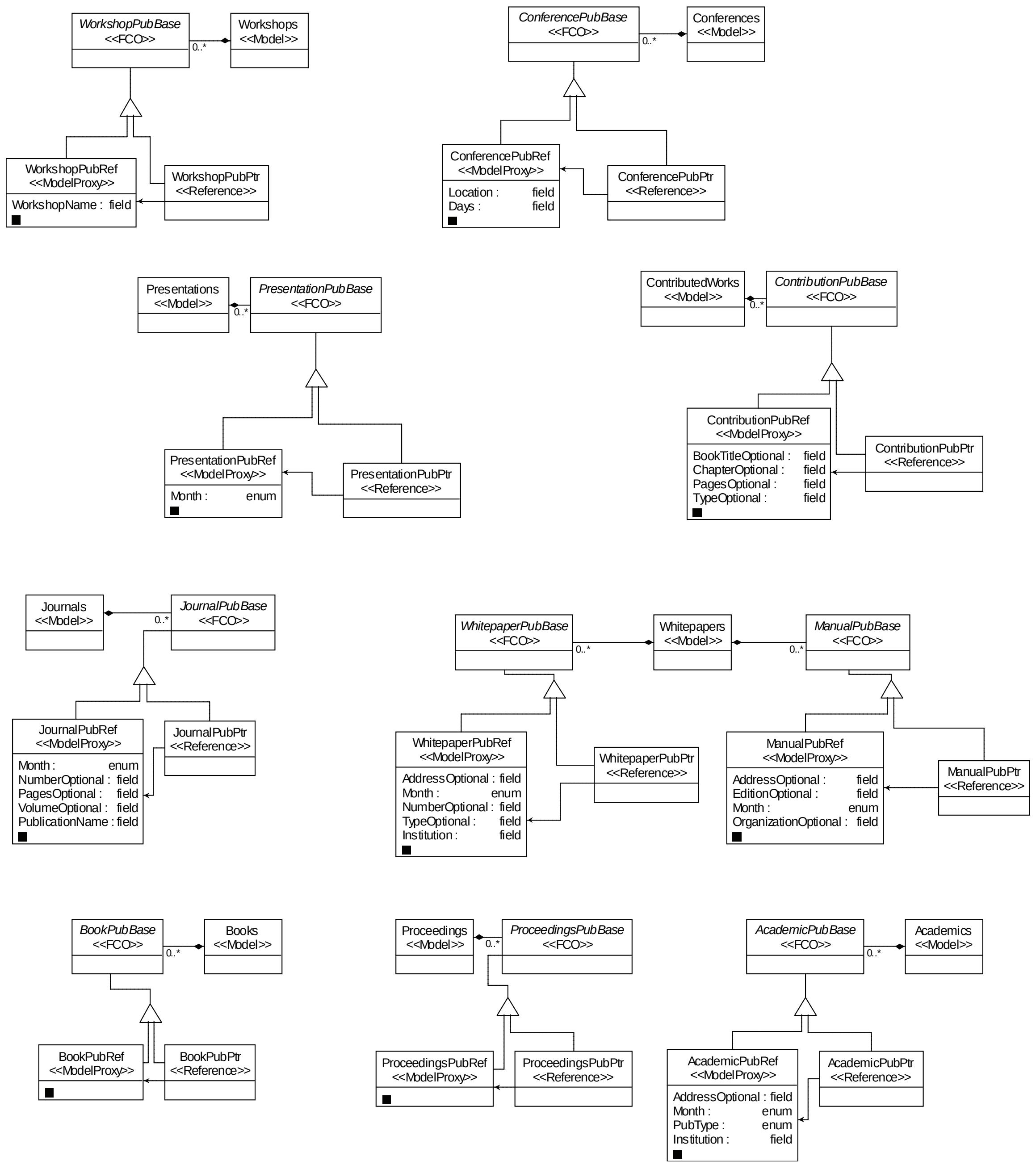


Figure 5: Objects in PubDB can refer to one another, to ease the difficulty of maintaining updated copies of multiply referred papers, or to create container `Databases` which can hold record of certain kinds of publication (i.e., a year's worth of publications, useful for an annual report).

`Abbreviation` is an `Atom` which is created in the `Abbreviations` aspect of a `Database`. When viewing the `TotalView` aspect, `Abbreviations` will not be visible.

If an `Abbreviation` exists for a `Database`, then any use of the abbreviated form in an attribute will be inserted as a string abbreviation in the BibTeX database. In HTML generation, the string is simply replaced by the long form attribute value.

Attributes:

`Long form`:	*field*
`Abbreviation`:	*field*

3.4.2 `Academics`

`Academics` is a `Model` which can be created in a `Database`. It may contain objects of kind `AcademicPub`, and `AcademicPubPtr`.

3.4.3 `AcademicPub`

figures/metamodel/icons/Publications `AcademicPub` is a `Model` which can be created inside a model of kind `Academics`. It's prefix during BibTeX generation is `a:`.

Supertype:

`Publication`

Attributes:

`Title`:	*field*, inherited from `Publication`
`Type of publication`:	*enum* Area Paper *(default)* Thesis Dissertation Report
`Institution`:	*field*
`Address`:	*optional field*

`Month (if applicable)`:	*enum* (optional) *(default)* January Febrary March April May June July August September October November December
`Year`:	*field*, inherited from `Publication`
`Additional Information`:	*optional field*, inherited from `Publication`

3.4.4 `AcademicPubPtr`

figures/metamodel/icons/PublicationRef `AcademicPubPtr` is a `Reference` to an `AcademicPub` object. I can be created inside an object of kind `Academics`, and can be sequenced to predict its order in generated artifacts.

Like all objects of kind `PubPtr`, it can be created in a `Database` for convenient reuse of a bibliographic entry across multiple databases, without needing to create the appropriate container model. It depends solely on its referred object for attribute values, and thus is useless unless it is pointing to an existing object. It uses only its GME name during generation. The GME name serves as the key name, following the prefix.

Supertype:

PubPtr

3.4.5 `Artifact (Abstract)`

`Artifact` is an abstract type (meaning that it may not be created on its own—only its subtypes may be created).

Subtypes:

`Document`, `Presentation`

Attributes:

`URL`:	*field*

3.4.6 `Author`

figures/metamodel/icons/author `Author` is a `Reference` to a `Person`. It can be created inside objects of kind `Publication`. Also, an `Author` need not point to a `Person` to be valid; in fact, null-pointers are useful for specifying authors who may not be frequently used, thus reducing the need to create a `Person` to which to refer. If an `Author` exists as a null-pointer, then it must have values for the attributes in order to retrieve the names. If an `Author` is given values for *any* of its attributes, then the code generator will not check to see whether or not it refers to a `Person`, and therefore will not use any attribute values from a `Person`, should there be a link.

When `Authors` are created inside a model, they are sequenced via connections to specify their order in the publication. Omitting these connections may result in unexpected behavior.

Attributes:

`First name`:	*field*
`Middle name(s)/initials`:	*field*
`Last name`:	*field*

As with the `Person`, when editing the name of an `Author`, the use of standard LaTeX accents and diacritical marks is permitted. To create a person who stands for the common bibliographic abbreviation *et al.* (short for the Latin *et alli*, "and others"), add as one of the fields the value `others`, and leave the other fields blank.

3.4.7 `Books`

`Books` is a `Model` which can be created in a `Database`. It may contain objects of kind `BookPub`, and `BookPubPtr`.

3.4.8 `BookPub`

figures/metamodel/icons/Publications `BookPub` is a `Model` which can be created inside a model of kind `Books`. It's prefix during BibTeX generation is `b:`.

Supertype:

`Publication`, `BookType`

Attributes:

`Title`:	*field*, inherited from `Publication`
`Edition`:	*optional field*, inherited from `BookType`
`Series`:	*optional field*, inherited from `BookType`
`Volume`:	*optional field*, inherited from `BookType`
`Number`:	*optional field*, inherited from `BookType`
`Publisher`:	*field*, inherited from `BookType`
`Address`:	*optional field*, inherited from `BookType`

`Month (if applicable)`:	*enum*
	(optional) *(default)*
	January
	Febrary
	March
	April
	May
	June
	July
	August
	September
	October
	November
	December
`Year`:	*field*, inherited from `Publication`
`Additional Information`:	*optional field*, inherited from `Publication`

3.4.9 `BookPubPtr`

figures/metamodel/icons/PublicationRef `BookPubPtr` is a `Reference` to an `BookPub` object. I can be created inside an object of kind `Books`, and can be sequenced to predict its order in generated artifacts.

Like all objects of kind `PubPtr`, it can be created in a `Database` for convenient reuse of a bibliographic entry across multiple databases, without needing to create the appropriate container model. It depends solely on its referred object for attribute values, and thus is useless unless it is pointing to an existing object. It uses only its GME name during generation. The GME name serves as the key name, following the prefix.

Supertype:

PubPtr

3.4.10 `Conferences`

`Conferences` is a `Model` which can be created in a `Database`. It may contain objects of kind `ConferencePub`, `ConferencePubPtr`, and `WorkshopPub` (though this is not encouraged, since workshop papers are generally grouped under a separate heading).

3.4.11 `ConferencePub`

figures/metamodel/icons/Publications `ConferencePub` is a `Model` which can be created inside a model of kind `Conferences`. It's prefix during BibTeX generation is `c:`.

Supertype:

`Publication`, `ProceedingsType`

Subtypes:

`WorkshopPub`

Attributes:

`Title`:	*field*, inherited from `Publication`
`Publication name`:	*optional field*, inherited from `ProceedingsType`
`Organization`:	*optional field*, inherited from `ProceedingsType`
`Series`:	*optional field*, inherited from `ProceedingsType`
`Volume`:	*optional field*, inherited from `ProceedingsType`
`Number`:	*optional field*, inherited from `ProceedingsType`
`Publisher`:	*field*, inherited from `ProceedingsType`
`Address`:	*optional field*, inherited from `ProceedingsType`
`Page(s) [p1--pn]`:	*optional field*, inherited from `ProceedingsType`
`Location`:	*optional field*
`Days`:	*optional field*
`Month (if applicable)`:	*enum* (optional) *(default)* January Febrary March April May June July August September October November December
`Year`:	*field*, inherited from `Publication`
`Additional Information`:	*optional field*, inherited from `Publication`

3.4.12 `ConferencePubPtr`

figures/metamodel/icons/PublicationRef `ConferencePubPtr` is a `Reference` to an `ConferencePub` object. I can be created inside an object of kind `Conferences`, and can be sequenced to predict its order in generated artifacts.

Like all objects of kind `PubPtr`, it can be created in a `Database` for convenient reuse of a bibliographic entry across multiple databases, without needing to create the appropriate container model. It depends solely on its referred object for attribute values, and thus is useless unless it is pointing to an existing object. It uses only its GME name during generation. The GME name serves as the key name, following the prefix.

Supertype:

PubPtr

3.4.13 `ContributedWorks`

`ContributedWorks` is a `Model` which can be created in a `Database`. It may contain objects of kind `ContributionPub`, and `ContributionPubPtr`.

3.4.14 `ContributionPub`

figures/metamodel/icons/Publications `ContributionPub` is a `Model` which can be created inside a model of kind `ContributedWorks`. It's prefix during BibTeX generation is `cont:`.

Supertype:

`Publication`, `BookType`

Attributes:

`Title`:	*field*, inherited from `Publication`
`Edition`:	*optional field*, inherited from `BookType`
`Book title`:	*optional field*
`Series`:	*optional field*, inherited from `BookType`
`Chapter(s) [c1--cn]`:	*optional field*
`Volume`:	*optional field*, inherited from `BookType`
`Number`:	*optional field*, inherited from `BookType`
`Publisher`:	*field*, inherited from `BookType`
`Address`:	*optional field*, inherited from `BookType`
`Type`:	*optional field*
`Page(s) [p1--pn]`:	*optional field*
`Month (if applicable)`:	*enum* (optional) *(default)* January Febrary March April May June July August September October November December
`Year`:	*field*, inherited from `Publication`
`Additional Information`:	*optional field*, inherited from `Publication`

3.4.15 `ContributionPubPtr`

figures/metamodel/icons/PublicationRef `ContributionPubPtr` is a `Reference` to an `ContributionPub` object. I can be created inside an object of kind `ContributedWorks`, and can be sequenced to predict its order in generated artifacts.

Like all objects of kind `PubPtr`, it can be created in a `Database` for convenient reuse of a bibliographic entry across multiple databases, without needing to create the appropriate container model. It depends solely on its referred object for attribute values, and thus is useless unless it is pointing to an existing object. It uses only its GME name during generation. The GME name serves as the key name, following the prefix.

Supertype:

PubPtr

3.4.16 `Database`

`Database` is a `Model` which collects publication-related items for organizational purposes. The `Database` is the fundamental unit which is generated during the interpretation phase. Its name forms the beginning of the generated filename (e.g., a `Database` object with GME name `MyPublications` would produce `MyPublications.bib` after running the GenerateBibTeXDB component, and `MyPublications.html` after running the GenerateHTML component.

Objects of kind `Database` can contain objects of kind `PublicationsSet`[4], `PubPtr`[5], and `Abbreviation`.

A `Database` may be created inside a `PublicationsDatabase` folder, or inside a `Publication` model or its subtype. The former is useful when grouping together publications which belong to a person/organization, etc. The latter is more useful when creating a database for a publication which is currently being edited, which will then store the bibliographic database inside the bibliographic information for that paper. This is extremely convenient when you remember that you cited a paper in a document you created, or someone else created, and you can locate that original bibliographic database easily.

Attributes:

`Title for HTML page`:	*field*
`Stylsheet location`:	*optional field*
`Output location`:	*field*
`Name of style`:	*optional field*

3.4.17 `Document`

figures/metamodel/icons/artifact `Document` is an `Atom`, which inherits from `Artifact`. `Documents` can be created in `Models` of kind `JournalPub`, `AcademicPub`, `WhitepaperPub`, `ManualPub`, `ProceedingsType`[6], and `BookType`[7].

`Documents` are used during HTML generation to produce links to where the document file may be found on the web. The document is then placed over to the left of the bibliographic entry of the HTML page. During this execution, the file type is analyzed to see if an appropriate icon can be used in place of the standard HTML icon (represented by a Microsoft Internet Explorer thumbnail). The file types that are currently analyzed and used are: `.doc`, `.htm`, `.html`, `.pdf`, `.pps`, `.ppt`, and `.zip`. If no `Document` exists in the publication, the space is left blank.

Supertype:

`Artifact`

[4]Notably its subtypes `Whitepapers`, `Conferences`, `Workshops`, `Academics`, `Journals`, `Books`, `Presentations`, `Proceedings`, and `ContributedWorks`

[5]Notably its subtypes `WhitepaperPubPtr`, `ConferencePubPtr`, `WorkshopPubPtr`, `AcademicPubPtr`, `JournalPubPtr`, `BookPubPtr`, `PresentationPubPtr`, `ProceedingsPubPtr`, and `ContributionPubPtr`

[6]Notably its subtypes `ProceedingsPub`, `ConferencePub`, and `WorkshopPub`

[7]Notably its subtypes `BookPub`, and `ContributionPub`

Attributes:

`URL`:	*field*, inherited from `Artifact`

3.4.18 `Editor`

figures/metamodel/icons/editor `Editor` is a `Reference` to a `Person`. It can be created inside objects of kind `BookType` and `ProceedingsType`. As with the kind `Author`, an `Editor` need not point to a `Person` to be valid; in fact, null-pointers are useful for specifying authors who may not be frequently used, thus reducing the need to create a `Person` to which to refer. If an `Editor` exists as a null-pointer, then it must have values for the attributes in order to retrieve the names. If an `Editor` is given values for *any* of its attributes, then the code generator will not check to see whether or not it refers to a `Person`, and therefore will not use any attribute values from a `Person`, should there be a link.

When `Editors` are created inside a model, they are sequenced via connections to specify their order in the publication. Omitting these connections may result in unexpected behavior.

Attributes:

`First name`:	*field*
`Middle name(s)/initials`:	*field*
`Last name`:	*field*

As with the `Person`, when editing the name of an `Editor`, the use of standard LaTeX accents and diacritical marks is permitted. To create a person who stands for the common bibliographic abbreviation *et al.* (short for the Latin *et alli*, "and others"), add as one of the fields the value `others`, and leave the other fields blank.

3.4.19 `Folders`

Folder are containers used to organize models. There are two kinds of folders in the PubDB paradigm: `People` and `PublicationsDatabase`. The `People` folder can only hold other `People` folders, and objects of kind `PeopleContainer`. The `PublicationsDatabase` folder can only hold objects of kind `Database`.

3.4.20 `Journals`

`Journals` is a `Model` which can be created in a `Database`. It may contain objects of kind `JournalPub`, and `JournalPubPtr`.

3.4.21 `JournalPub`

figures/metamodel/icons/Publications `JournalPub` is a `Model` which can be created inside a model of kind `Journals`. It's prefix during BibTeX generation is `j:`.

Supertype:

`Publication`

Attributes:

`Title`:	*field*, inherited from `Publication`

`Publication name`:	*field*
`Volume`:	*optional field*
`Number`:	*optional field*
`Page(s) [p1--pn]`:	*optional field*
`Month (if applicable)`:	*enum*

(optional) *(default)*
January
Febrary
March
April
May
June
July
August
September
October
November
December

`Year`:	*field*, inherited from `Publication`
`Additional Information`:	*optional field*, inherited from `Publication`

3.4.22 `JournalPubPtr`

figures/metamodel/icons/PublicationRef `JournalPubPtr` is a `Reference` to an `JournalPub` object. I can be created inside an object of kind `Journals`, and can be sequenced to predict its order in generated artifacts.

Like all objects of kind `PubPtr`, it can be created in a `Database` for convenient reuse of a bibliographic entry across multiple databases, without needing to create the appropriate container model. It depends solely on its referred object for attribute values, and thus is useless unless it is pointing to an existing object. It uses only its GME name during generation. The GME name serves as the key name, following the prefix.

Supertype:

PubPtr

3.4.23 `ManualPub`

figures/metamodel/icons/Publications `ManualPub` is a `Model` which can be created inside a model of kind `Whitepapers`. It's prefix during BibTeX generation is `m:`.

Supertype:

`Publication`

Attributes:

`Title`:	*field*, inherited from `Publication`
`Organization`:	*optional field*

`Edition`:	*optional field*
`Address`:	*optional field*
`Month (if applicable)`:	*enum*
	(optional) *(default)*
	January
	Febrary
	March
	April
	May
	June
	July
	August
	September
	October
	November
	December
`Year`:	*field*, inherited from `Publication`
`Additional Information`:	*optional field*, inherited from `Publication`

3.4.24 `ManualPubPtr`

figures/metamodel/icons/PublicationRef `ManualPubPtr` is a `Reference` to an `ManualPub` object. I can be created inside an object of kind `Whitepapers`, and can be sequenced to predict its order in generated artifacts.

Like all objects of kind `PubPtr`, it can be created in a `Database` for convenient reuse of a bibliographic entry across multiple databases, without needing to create the appropriate container model. It depends solely on its referred object for attribute values, and thus is useless unless it is pointing to an existing object. It uses only its GME name during generation. The GME name serves as the key name, following the prefix.

Supertype:

PubPtr

3.4.25 `Presentation`

figures/metamodel/icons/artifact

3.4.26 `PeopleContainer`

`PeopleContainer` is a `Model`. It can contain zero or more `Persons`, and zero or more `PeopleContainers`. It can also be created inside a `Folder` of kind `People`.

3.4.27 `Person`

figures/metamodel/icons/person `Person` is an `Atom`. It cannot contain any objects. It may be created in a `PeopleContainer` model. It may be created as a reference, through kinds `Author` and `Editor`.

Attributes:

`First name`:	*field*
`Middle name(s)/initials`:	*field*
`Last name`:	*field*

When editing the name of a `Person`, the use of standard LaTeX accents and diacritical marks is permitted. To create a `Person` who stands for the common bibliographic abbreviation *et al.* (short for the Latin *et alli*, "and others"), add as one of the fields the value `others`, and leave the other fields blank[8].

`Presentation` is an `Atom`, which inherits from `Artifact`. Objects of kind `Presentation` can be created in `Models` of kind `PresentationPub`, `AcademicPub`, and `ConferencePub`[9].

Objects of kind `Presentation` are used during HTML generation to produce links to where a presentation of the bibliographic entry may be found on the web. The link to the online presentation is then placed to the right of the bibliographic entry of the HTML page. During this execution, the file type is analyzed to see if an appropriate icon can be used in place of the standard HTML icon (represented by a Microsoft Internet Explorer thumbnail). The file types that are currently analyzed and used are: `.doc`, `.htm`, `.html`, `.pdf`, `.pps`, `.ppt`, and `.zip`. If no `Presentation` exists in the publication, the space is left blank.

Supertype:

`Artifact`

Attributes:

`URL`:	*field*, inherited from `Artifact`

3.4.28 `Presentations`

`Presentations` is a `Model` which can be created in a `Database`. It may contain objects of kind `PresentationPub`, and `PresentationPubPtr`.

3.4.29 `PresentationPub`

figures/metamodel/icons/Publications `PresentationPub` is a `Model` which can be created inside a model of kind `Presentations`. It's prefix during BibTeX generation is `p:`.

Supertype:

`Publication`

Attributes:

`Title`:	*field*, inherited from `Publication`

[8]Note that in order for this to function properly with BibTeX, there must be at least one additional author, listed *before* `others`.

[9]As well as its subtype `WorkshopPub`

`Month (if applicable)`:	*enum*
	(optional) *(default)*
	January
	Febrary
	March
	April
	May
	June
	July
	August
	September
	October
	November
	December
`Year`:	*field*, inherited from `Publication`
`Additional Information`:	*optional field*, inherited from `Publication`

3.4.30 `PresentationPubPtr`

figures/metamodel/icons/PublicationRef `PresentationPubPtr` is a `Reference` to an `PresentationPub` object. I can be created inside an object of kind `Presentations`, and can be sequenced to predict its order in generated artifacts.

Like all objects of kind `PubPtr`, it can be created in a `Database` for convenient reuse of a bibliographic entry across multiple databases, without needing to create the appropriate container model. It depends solely on its referred object for attribute values, and thus is useless unless it is pointing to an existing object. It uses only its GME name during generation. The GME name serves as the key name, following the prefix.

Supertype:

PubPtr

3.4.31 `Proceedings`

`Proceedings` is a `Model` which can be created in a `Database`. It may contain objects of kind `ProceedingsPub`, and `ProceedingsPubPtr`.

3.4.32 `ProceedingsPub`

figures/metamodel/icons/Publications `ProceedingsPub` is a `Model` which can be created inside a model of kind `Proceedings`. It's prefix during BibTeX generation is `p:`.

Supertype:

`Publication`,`ProceedingsType`

Attributes:

`Title`:	*field*, inherited from `Publication`
`Publication name`:	*optional field*, inherited from `ProceedingsType`
`Organization`:	*optional field*, inherited from `ProceedingsType`
`Series`:	*optional field*, inherited from `ProceedingsType`
`Volume`:	*optional field*, inherited from `ProceedingsType`
`Number`:	*optional field*, inherited from `ProceedingsType`
`Publisher`:	*field*, inherited from `ProceedingsType`
`Address`:	*optional field*, inherited from `ProceedingsType`
`Page(s) [p1--pn]`:	*optional field*, inherited from `ProceedingsType`
`Month (if applicable)`:	*enum*
	(optional) *(default)*
	January
	Febrary
	March
	April
	May
	June
	July
	August
	September
	October
	November
	December
`Year`:	*field*, inherited from `Publication`
`Additional Information`:	*optional field*, inherited from `Publication`

3.4.33 `ProceedingsPubPtr`

figures/metamodel/icons/PublicationRef `ProceedingsPubPtr` is a `Reference` to an `ProceedingsPub` object. I can be created inside an object of kind `Proceedings`, and can be sequenced to predict its order in generated artifacts.

Like all objects of kind `PubPtr`, it can be created in a `Database` for convenient reuse of a bibliographic entry across multiple databases, without needing to create the appropriate container model. It depends solely on its referred object for attribute values, and thus is useless unless it is pointing to an existing object. It uses only its GME name during generation. The GME name serves as the key name, following the prefix.

Supertype:

PubPtr

3.4.34 `ProceedingsType (Abstract)`

`ProceedingsType` is an abstract type (meaning that it may not be created on its own—only its subtypes may be created).

Supertype:

`Publication`

Subtypes:

`ConferencePub`, `ProceedingsPub`, `WorkshopPub`

Attributes:

`Title`:	*field*, inherited from `Publication`
`Publication name`:	*optional field*
`Organization`:	*optional field*
`Series`:	*optional field*
`Volume`:	*optional field*
`Number`:	*optional field*
`Publisher`:	*optional field*
`Address`:	*optional field*
`Page(s) [p1--pn]`:	*optional field*
`Month (if applicable)`:	*enum*
	(optional) *(default)*
	January
	Febrary
	March
	April
	May
	June
	July
	August
	September
	October
	November
	December
`Year`:	*field*, inherited from `Publication`
`Additional Information`:	*optional field*, inherited from `Publication`

3.4.35 `Publication (Abstract)`

`Publication` is an abstract type, which is the supertype of all kinds of publications which may be created. It has as its attributes the values common to all kinds of publications considered in this bibliographic domain. It can contain objects of kind `Database`, and `Author`, and it can sort objects of kind `Author` through the `Precedence` connection.

Subtypes:

`AcademicPub`, `BookPub`, `ConferencePub`, `ContributionPub`, `JournalPub`, `ManualPub`, `PresentationPub`, `ProceedingsPub`, `WhitepaperPub`, and `WorkshopPub`

Attributes:

`Title`:	*field*
`Year`:	*field*
`Additional Information`:	*optional field*

3.4.36 `PublicationsSet (Abstract)`

`PublicationsSet` is the supertype of several kinds of objects, used to speak generally about containment and connection relationships. Thus, it cannot be created explicitly (only its subtypes can).

Subtypes:

> `Whitepapers`, `Conferences`, `Workshops`, `Academics`, `Journals`, `Books`, `Presentations`, `Proceedings`, and `ContributedWorks`

3.4.37 `Whitepapers`

`Whitepapers` is a `Model` which can be created in a `Database`. It may contain objects of kind `WhitepaperPub`, `WhitepaperPubPtr`, `ManualPub`, and `ManualPubPtr`. The GME name of the object determines the heading title when generating HTML.

3.4.38 `WhitepaperPub`

figures/metamodel/icons/Publications `WhitepaperPub` is a `Model` which can be created inside a model of kind `Whitepapers`. It's prefix during BibTeX generation is `tr:` (for `techreport`).

Supertype:

> `Publication`

Attributes:

`Title`:	*field*, inherited from `Publication`
`Institution`:	*field*
`Number`:	*optional field*
`Address`:	*optional field*
`Type`:	*optional field*

`Month (if applicable)`:	*enum* (optional) *(default)* January Febrary March April May June July August September October November December
`Year`:	*field*, inherited from `Publication`
`Additional Information`:	*optional field*, inherited from `Publication`

3.4.39 `WhitepaperPubPtr`

figures/metamodel/icons/PublicationRef `WhitepaperPubPtr` is a `Reference` to an `WhitepaperPub` object. I can be created inside an object of kind `Whitepapers`, and can be sequenced to predict its order in generated artifacts.

Like all objects of kind `PubPtr`, it can be created in a `Database` for convenient reuse of a bibliographic entry across multiple databases, without needing to create the appropriate container model. It depends solely on its referred object for attribute values, and thus is useless unless it is pointing to an existing object. It uses only its GME name during generation. The GME name serves as the key name, following the prefix.

Supertype:

PubPtr

3.4.40 `Workshops`

`Workshops` is a `Model` which can be created in a `Database`. It may contain objects of kind `WorkshopPub`, and `WorkshopPubPtr`.

3.4.41 `WorkshopPub`

figures/metamodel/icons/Publications `WorkshopPub` is a `Model` which can be created inside a model of kind `Workshops`. It's prefix during BibTeX generation is `w:`.

Supertype:

`Publication`, `ConferencePub`, `ProceedingsType`

Attributes:

`Title`:	*field*, inherited from `Publication`
`Publication name`:	*optional field*, inherited from `ProceedingsType`
`Organization`:	*optional field*, inherited from `ProceedingsType`
`Series`:	*optional field*, inherited from `ProceedingsType`
`Volume`:	*optional field*, inherited from `ProceedingsType`
`Number`:	*optional field*, inherited from `ProceedingsType`
`Publisher`:	*field*, inherited from `ProceedingsType`
`Address`:	*optional field*, inherited from `ProceedingsType`
`Page(s) [p1--pn]`:	*optional field*, inherited from `ProceedingsType`
`Location`:	*optional field*, inherited from `ConferencePub`
`Days`:	*optional field*, inherited from `ConferencePub`

`Month (if applicable)`: *enum*

(optional) *(default)*
January
Febrary
March
April
May
June
July
August
September
October
November
December

`Year`:	*field*, inherited from `Publication`
`Additional Information`:	*optional field*, inherited from `Publication`

3.4.42 `WorkshopPubPtr`

figures/metamodel/icons/PublicationRef `WorkshopPubPtr` is a `Reference` to an `WorkshopPub` object. I can be created inside an object of kind `Workshops`, and can be sequenced to predict its order in generated artifacts.

Like all objects of kind `PubPtr`, it can be created in a `Database` for convenient reuse of a bibliographic entry across multiple databases, without needing to create the appropriate container model. It depends solely on its referred object for attribute values, and thus is useless unless it is pointing to an existing object. It uses only its GME name during generation. The GME name serves as the key name, following the prefix.

Supertype:

PubPtr

4 Creating a Database

5 Managing Persons

6 Creating New Publications

7 Best Practices

8 Troubleshooting

9 Quick Start

This section is useful for users already familiar with the GME tool [1]. Please refer to the GME Manual for specific instructions which are not the purpose of this document. Please install GME on your system prior to reading this section. The version of GME for which this User's Manual was tested was GME5, v.5.6.8.

9.1 Downloading the binaries

If you have not yet downloaded the binaries, they are available from `http://www.eecs.berkeley.edu/~sprinkle/useful/pubdb/`, available as a .zip file. Please extract the contents of this file somewhere on your computer, where they will be placed in their own directory. I recommend that you place them in the `$GME_ROOT/Contrib` directory, where they will appear as `$GME_ROOT/Contrib/PubDB_v5.7.20` . [10]

9.2 Registering your paradigm and components

In order to utilize the PubDB modeling environment, we must register the PubDB paradigm with GME so it will know the kinds of objects you can create.

1. Open GME, as shown in Fig. 6.

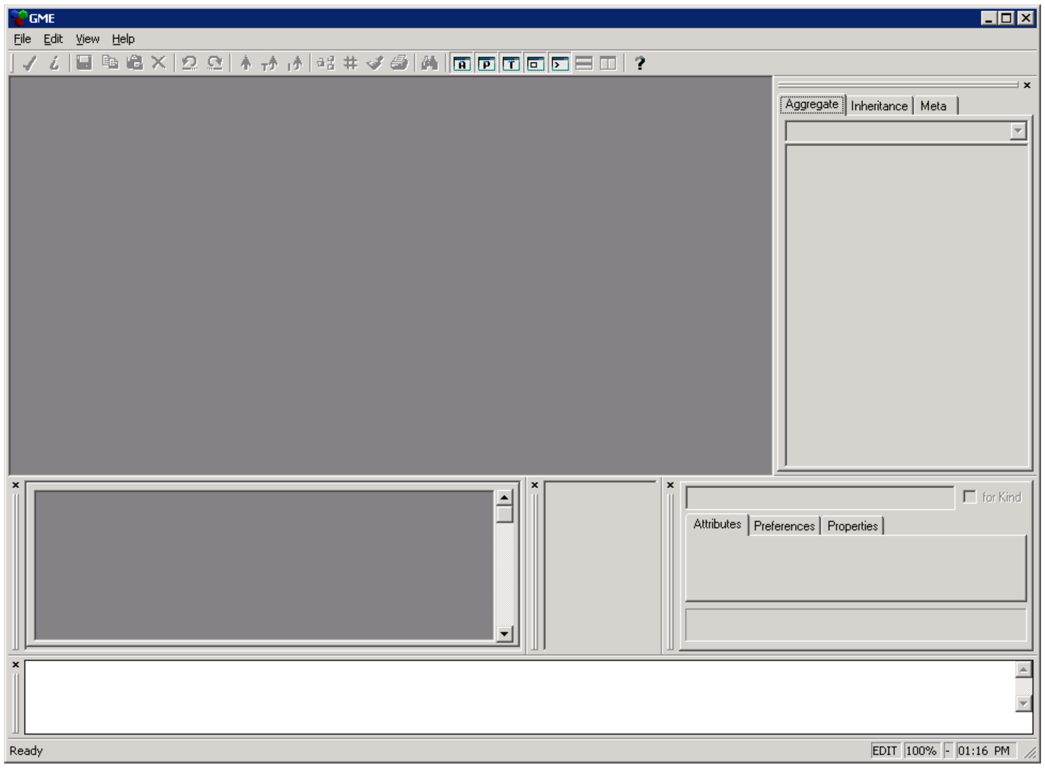
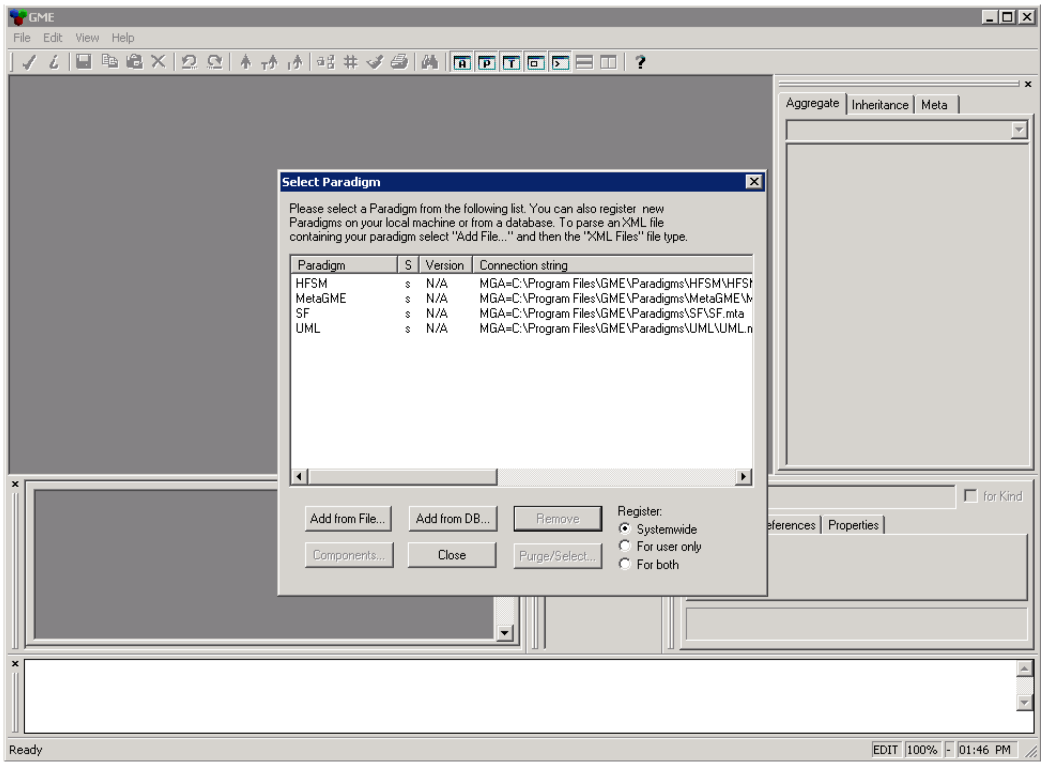


Figure 6: The Generic Modeling Environment, unconfigured.

2. Choose `File`→`Register Paradigms...`, to reveal the dialog shown in Fig. 7.

[10] Downloading the source: The source for the projects is not easily available for download at this time. However, the author is pleased to send it to you if you would like to look at his terribly hacked code. The source is not really necessary to run, only to recompile.



Please select a Paradigm from the following list. You can also register new Paradigms on your local machine or from a database. To parse an XML file containing your paradigm select "Add File..." and then the "XML Files" file type.

Paradigm	S	Version	Connection string
HFSM	s	N/A	MGA=C:\Program Files\GME\Paradigms\HFSM\HFSI
MetaGME	s	N/A	MGA=C:\Program Files\GME\Paradigms\MetaGME\M
SF	s	N/A	MGA=C:\Program Files\GME\Paradigms\SF\SF.mta
UML	s	N/A	MGA=C:\Program Files\GME\Paradigms\UML\UML.n

Register: [✓ Systemwide] For user only For both

Figure 7: Adding the PublicationsDatabase paradigm.

3. Select `Register: For both` in the lower right corner. If you wish for the paradigm to be registered only to you as the user, then leave the selection as `Register: For user only`. However, it is recommended that you register the paradigm and components systemwide, in the event that other users later want to use the PubDB environment.
4. Choose `Add from File...`, select the paradigm, in file `$GME_ROOT/Contrib/PubDB_v5.7.20/meta/PublicationsDatabase.xmp`, and select `Open`.
5. While you have the PublicationsDatabase paradigm selected, choose `Components...`.
6. Select the `Register: Systemwide` radio button.
7. Choose `Install New...`, select the component, in file `$GME_ROOT/Contrib/PubDB_v5.7.20/bin/GenerateBibTeXDB.dll`
8. Choose `Install New...`, select the component, in file `$GME_ROOT/Contrib/PubDB_v5.7.20/bin/GenerateHTML.dll`
9. Choose `Install New...`, select the component, in file `$GME_ROOT/Contrib/PubDB_v5.7.20/bin/GenerateLaTeXSummaries.dll`
10. Choose `Close`
11. Choose `Close`

You should now see the blank GME window again (as in Fig. 6).

9.3 Example models

Several examples are included with the release. In this subsection, the details of importing and viewing the database which was used to create this document are given.

9.3.1 Importing the `.xme` file

The textual version of GME models is called the `.xme` file, and it is in XML format. This format is preferred to the `.mga` format, since this format is binary storage, and not guaranteed to load between versions of the GME tool (i.e., although the paradigm does not change, and is still valid for newer versions of GME, the models created may not open).

1. Open GME, as shown in Fig. 6.
2. Choose `File`→`Import XML...`
3. Choose the exported model, in file `$GME_ROOT/Contrib/PubDB_v5.7.20/examples/doc/UsingPublicationsDatabase-exported.xme`
4. Choose `Open`
5. Select `Create project file` (the default)
6. Choose `Next`
7. Choose a name for the model, such as `UsingPublicationsDatabase.mga`, and place it in the folder `$GME_ROOT/Contrib/PubDB_v5.7.20/examples/doc/`
8. You should get a dialog which says "The XML file was successfully imported."

You should now see the blank GME windows, except that there should be two component buttons, and the browser should be populated by an item called `Root Folder`. This is shown in Fig. 8.

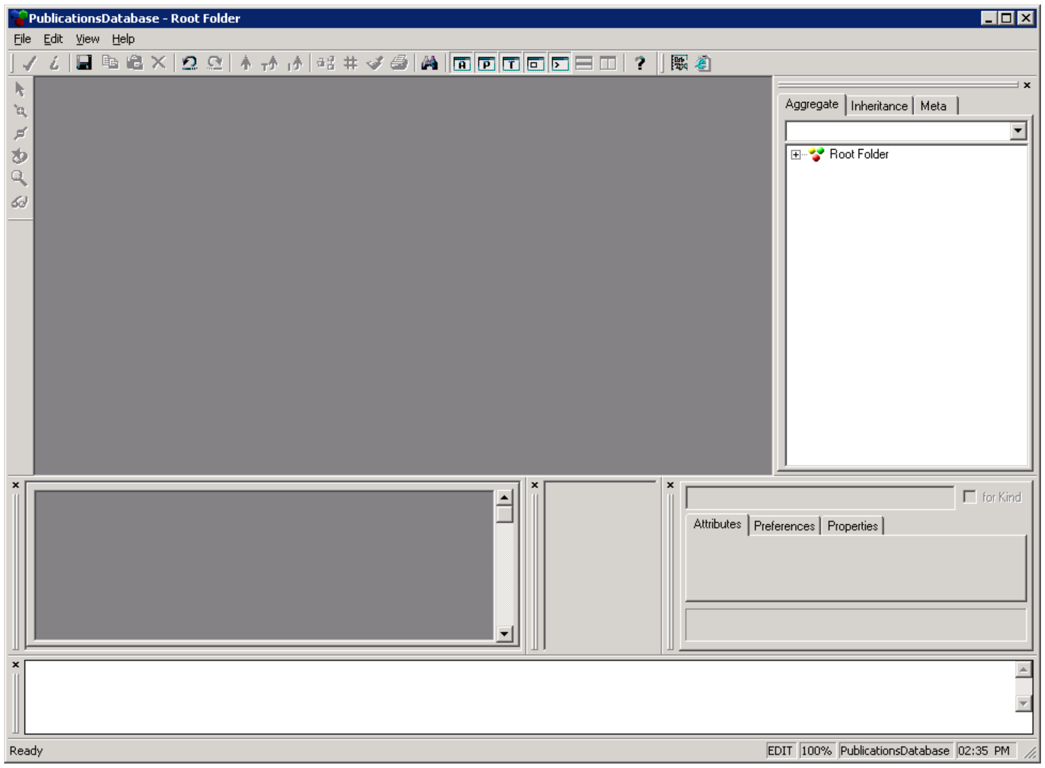
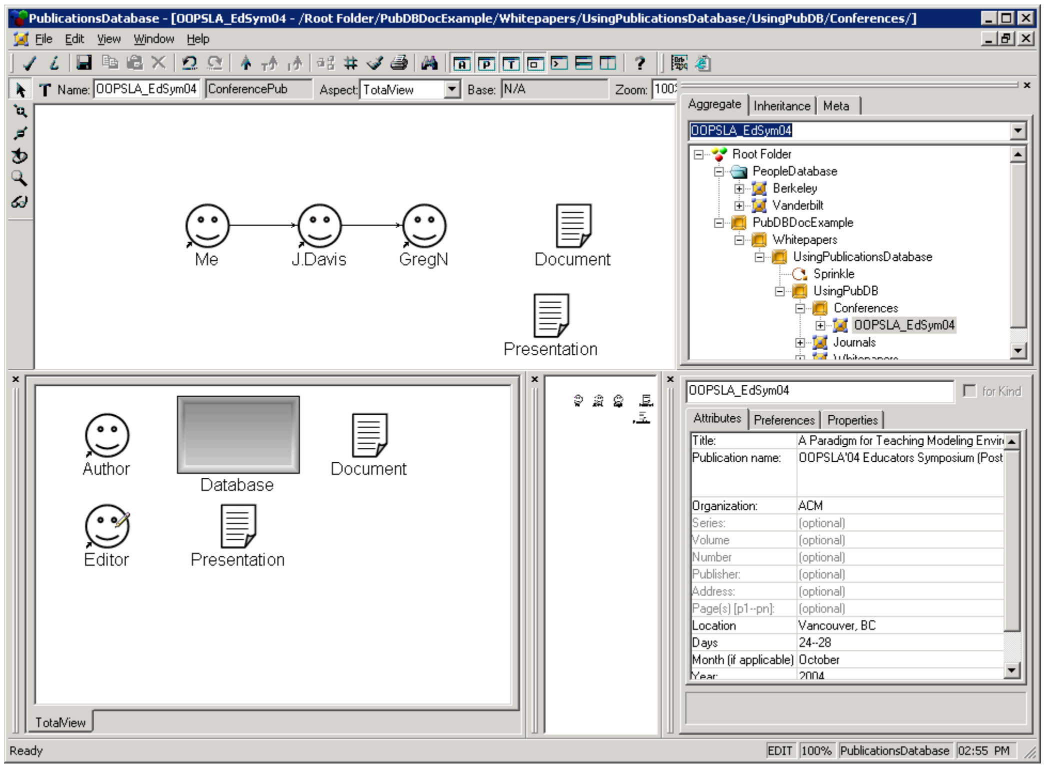

PublicationsDatabase - Root Folder
File Edit View Help
Aggregate Inheritance Meta
Root Folder
for Kind
Attributes Preferences Properties
Ready
EDIT 100% PublicationsDatabase 02:35 PM

Figure 8: The Generic Modeling Environment, configured for PubDB, having recently imported a model via the XML format, `.xme`



Title:	A Paradigm for Teaching Modeling Envir
Publication name:	OOPSLA'04 Educators Symposium (Post
Organization:	ACM
Series:	(optional)
Volume	(optional)
Number	(optional)
Publisher:	(optional)
Address:	(optional)
Page(s) [p1--pn]:	(optional)
Location	Vancouver, BC
Days	24--28
Month (if applicable)	October
Year:	2004

Figure 9: An example publication entry in the publications database. Note the ordering of the authors, as well as the presence of where you can find the presentation and document online.

9.3.2 Browsing the model

Use the tree browser on the right hand side of the screen to look through the database of publications which were used during the production of this manual. Double-click on models to view them in the main window (see Fig. 9). Please use these model examples to gain expertise on how to

9.4 Generating Output

Output is created by a phase called *interpretation* in GME. There are two kinds of output to generate: BibTeX `.bib` files, and HTML files. A default location of `./` is chosen upon creation of a new model file for the output of these generators. You can modify this to be a more useful directory, and/or choose to select the output directory at interpretation time.

Databases are generated from the context of the *current selected object*, regardless of how the interpreter is evoked.

9.4.1 Generating a BibTeX database

?? Select the database "UsingPubDB", found in the following path of the GME project imported in Sect. 9.3. There are four ways to invoke the interpreter—once it is invoked, the process is identical to complete.

1. Choose the BibTeX icon from the component bar in GME (see Fig. 10(b)).
2. Choose the *i* button from the component navigation bar. Then, select `GenerateBibTeXDB`, and choose `Interpret` (as shown in Fig. 10(a)).
3. Choose `File→Run Interpreter→GenerateBibTeXDB Interpreter`

4. Right click in the correct model context[11], and select **`Interpret`**, which invokes the dialog shown in Fig. 10(a).

Once you have invoked the interpreter, the steps are:

1. Answer `Yes` to the question "Use default folder (from "OutputLocation" in Database model)?"
2. Answer appropriately to the question "Would you like to generate a CiteFile (in the same directory) to cite each entry in the generated BibTeX database?"
   (a) `Yes`, if you want to create a file which will automatically cite all of these entries, and produce an output document which will list all of the items. Use this example when producing output that you will copy into another document which may not be using BibTeX, but you want to create BibTeX-like entries (e.g., a resume).
   (b) `No`, if you are using the **`.bib`** output as part of an existing project, and you will decide when/where they are cited. For most applications, this is the best option.
3. You should now see a dialog displaying the output location and name of your **`.bib`** file. For this example, the output location should read as follows:
   `./\UsingPubDB.bib`

9.4.2 Generating HTML

Select the database "UsingPubDB", found in the following path of the GME project imported in Sect. 9.3. There are four ways to invoke the interpreter—once it is invoked, the process is identical to complete.

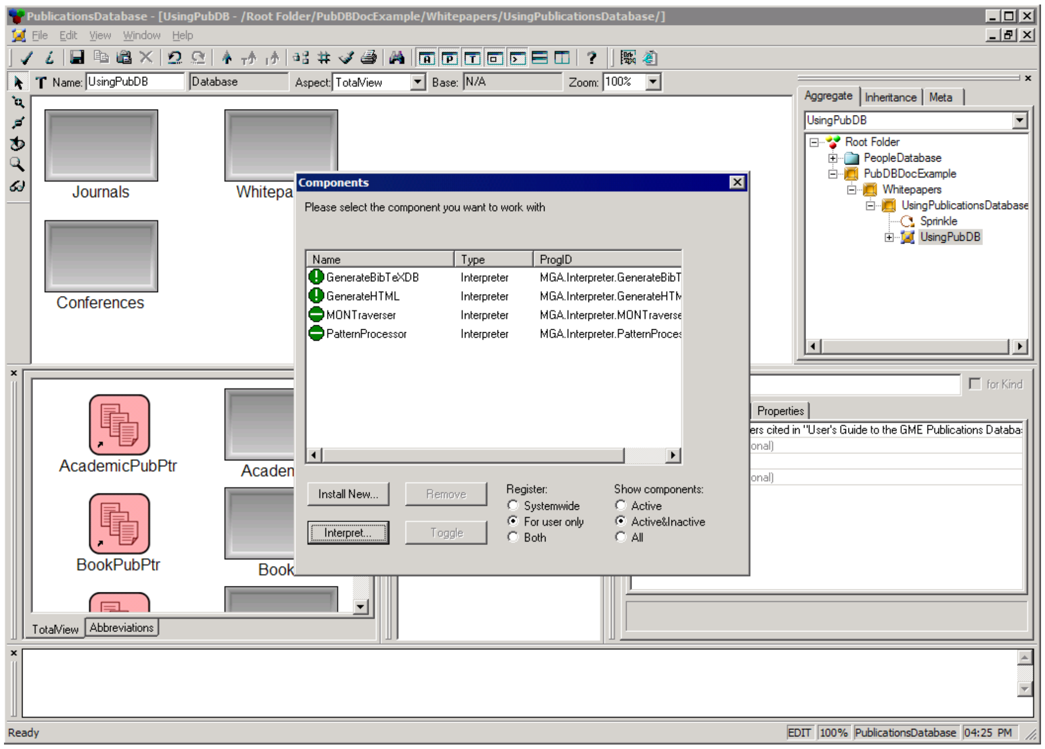
1. Choose the HTML icon from the component bar in GME (see Fig. 10(c)).
2. Choose the *i* button from the component navigation bar. Then, select `GenerateHTML`, and choose `Interpret` (as shown in Fig. 10(a)).
3. Choose **`File→Run Interpreter→GenerateHTML Interpreter`**
4. Right click in the correct model context[12], and select **`Interpret`**, which invokes the dialog shown in Fig. 10(a).

Once you have invoked the interpreter, the steps are:

1. Answer `Yes` to the question "Use default folder (from "OutputLocation" in Database model)?"
2. You should now see a dialog displaying the output location and name of your **`.html`** file. For this example, the output location should read as follows:
   `./\UsingPubDB.html`
3. In order to see the icons correctly, you must use the files which are found in
   `$GME_ROOT/Contrib/PubDB_v5.7.20/web`
   in the same directory as the output **`.html`** file. If you copy the output file to the **`/web`** directory, or the icons to the **`/examples/doc`** directory, you should see the icons correctly.

[11]The right click is used to provide a single model as the point of execution for the interpreter. You must click on the model itself while viewing the interior of a container. The generative interpreters for the PubDB paradigm operate only on objects of kind **`Database`**. If another object is selected, the interpreter traces back in the ownership hierarchy to find the nearest—i.e., lowest—database container.

[12]The right click is used to provide a single model as the point of execution for the interpreter. You must click on the model itself while viewing the interior of a container. The generative interpreters for the PubDB paradigm operate only on objects of kind **`Database`**. If another object is selected, the interpreter traces back in the ownership hierarchy to find the nearest—i.e., lowest—database container.



PublicationsDatabase - [UsingPubDB - /Root Folder/PubDBDocExample/Whitepapers/UsingPublicationsDatabase/]

Name: UsingPubDB Database Aspect: TotalView Base: N/A Zoom: 100%

Components

Please select the component you want to work with

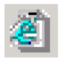
Name	Type	ProgID
GenerateBibTeXDB	Interpreter	MGA.Interpreter.GenerateBibT
GenerateHTML	Interpreter	MGA.Interpreter.GenerateHTM
MONTraverser	Interpreter	MGA.Interpreter.MONTraverse
PatternProcessor	Interpreter	MGA.Interpreter.PatternProces

Register: ( ) Systemwide [✓ For user only] ( ) Both

Show components: ( ) Active [✓ Active&Inactive] ( ) All

(a) GME Interpreter Choice Dialog. This dialog is invoked, since there is more than one possible interpreter for this paradigm.

(b) The GenerateBibTeXDB interpreter icon.



(c) The GenerateHTML interpreter icon.

Figure 10: Methods for interpreter invocation

If you specify a stylesheet and title, these are appropriately inserted into the HTML code such that you can take advantage of the color/link/font schemes you desire. For example, see the
`$GME_ROOT/Contrib/PubDB_v5.7.20/examples/html/index.html`
for a version which uses frames and stylesheets alongside generated BibTeX and HTML files for a comprehensive HTML display that also yields BibTeX entries for other researchers to use when citing your work.

9.4.3 Generating LaTeX Summaries

Select the database "SampleSummaryCollection", found in the following path of the GME project imported in Sect. 9.3. There are four ways to invoke the interpreter—once it is invoked, the process is identical to complete.

1. Choose the Summaries icon from the component bar in GME (see Fig. ??).
2. Choose the *i* button from the component navigation bar. Then, select `GenerateLaTeXSummaries`, and choose `Interpret` (as shown in Fig. 10(a)).
3. Choose `File`→`Run Interpreter`→`GenerateLaTeXSummaries Interpreter`
4. Right click in the correct model context[13], and select `Interpret`, which invokes the dialog shown in Fig. 10(a).

Once you have invoked the interpreter, the steps are:

1. Answer `Yes` to the question "Use default folder (from "OutputLocation" in Database model)?"
2. You should now see a dialog displaying the output location and name of your `.tex` file. For this example, the output location should read as follows:
   `./\SampleSummaryCollection.tex`
3. Generate the BibTeX database for this GME model, as discussed in Sect. ?? (this is necessary to compile the file correctly).
4. In order to see the output PDF file, compile the main `.tex` file and run BibTeX appropriately, and you should see a full document.

If you want to customize the headers and footers of the summary document, then follow the directions in the output files.

[13]The right click is used to provide a single model as the point of execution for the interpreter. You must click on the model itself while viewing the interior of a container. The generative interpreters for the PubDB paradigm operate only on objects of kind `Database`. If another object is selected, the interpreter traces back in the ownership hierarchy to find the nearest—i.e., lowest—database container.

References

[1] A. Lédeczi *et al.*, *The Generic Modeling Environment*, Vanderbilt University, http://www.isis.vanderbilt.edu/, 2005, GME5.