

Yablona

Dynamic Data Modelling

by
Peter Gürtler
pguertler@gmx.net

1 Table of Contents

1	Table of Contents	2
2	Introduction.....	2
3	Features	3
3.1	Dynamic Search.....	3
3.2	Dynamic Attributes	4
3.3	Object References.....	5
3.4	Historization.....	5
4	Dynamic Types.....	6
4.1	Types and Subtypes.....	6
4.2	Attributes	8
5	Update Points.....	9
6	Architecture and Environment	10
6.1	Overview	10
6.2	Persistence	10
6.3	Application Server	10
6.4	Clients and Client Environment.....	10
7	Future of Yablona.....	11
7.1	Events	11
7.2	Database.....	11
7.3	File System.....	11
8	References.....	11

2 Introduction

This document gives a short introduction to Yablona. It is aimed at readers who are already familiar with basic concepts of data modelling (e.g. in SQL or Java) and object oriented programming (OOP).

In theory, Yablona is an object oriented, dynamically typed data modelling approach. The type of a Yablona object is not determined when it is instantiated, the type is rather determined at runtime by evaluating the attribute values of the object. This leads to a very dynamic environment where object data and metadata highly interact.

In practice, Yablona is a generic web application implemented in Enterprise Java Beans (EJB) technology with a Java Server Faces (JSF) web client. The application can be configured to be e.g. an 'Issue tracker / Bug tracker' software or a media file library. The data modelling for these applications happens at runtime in the web application (and not at development time like in many traditional systems).

You can see the configurations work at <http://www.yablona.org/issuetracker> and <http://www.yablona.org/cyrilov>.

Yablona is open source. All sources are available at www.yablona.org.

Yablona is designed and developed by Peter Gürtler (pguertler@gmx.net). I would like to thank Marco Terzer for many very inspiring discussions that have significantly pushed Yablona forward.

3 Features

The following examples are taken from the “Media File Library” configuration of Yablona. They show the basic concepts of Yablona in an end-user perspective.

3.1 Dynamic Search

On the left side of the screen, the user can filter the items to be shown on the righthand side. In figure 1, only items of ‘Media Type’ ‘Text’ are displayed. If the ‘Media Type’ filter is changed to ‘Sound’, the set of filter attributes changes (figure 2): The attributes ‘Mix’ and ‘Instruments’ are applicable for sound files, but have not been applicable in the ‘Text’ file filter. The columns in the item list have also dynamically adapted to the attributes that are available.

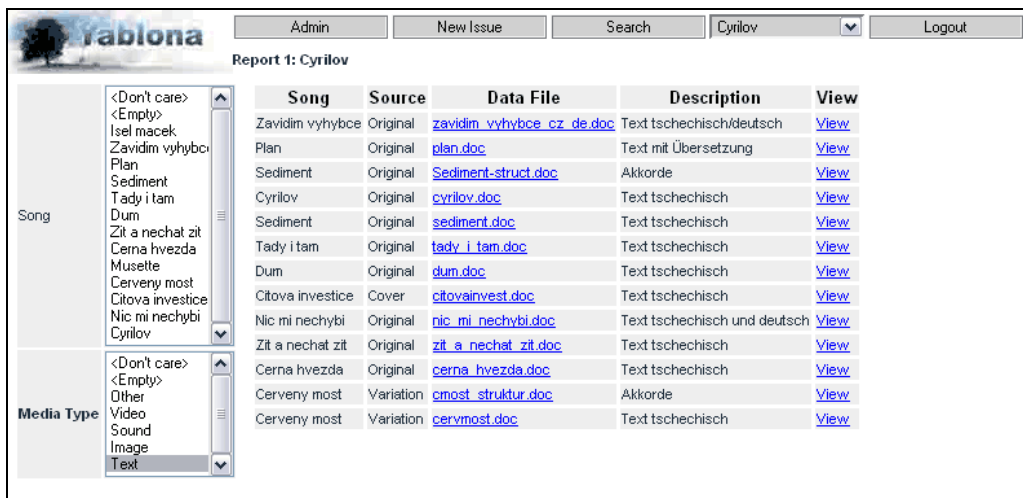


Figure 1: Searching files of 'Media Type' 'Text'

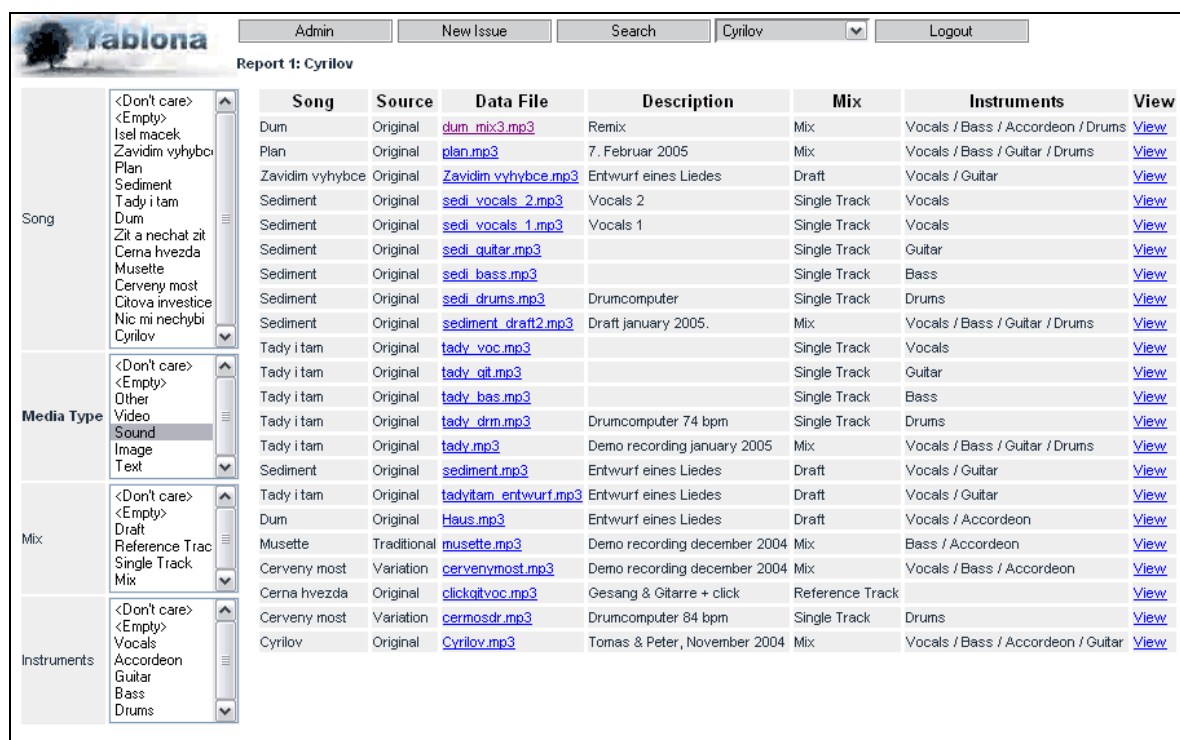


Figure 2: Different attributes for 'Media Type' 'File'

3.2 Dynamic Attributes

When adding a new file resource and classifying it (i.e. creating a new item), the set of available attributes also dynamically changes based on the attribute values already set. Figure 3 shows the attributes for a new file resource. The user can select the song, to which the resource is related, upload the file resource, type in a description and set the media type. If he sets the 'Media Type' to 'Sound', two additional attributes appear (see figure 4).

The screenshot shows the 'Issue new' form in the Yablona application. At the top, there are buttons for 'Update', 'Save', 'Cancel', and 'Logout'. The form is divided into several sections:

- Song:** A dropdown menu with the following options: <Don't care>, <Empty>, Isel macek, Zavidim vyhybce, Plan, Sediment, Tady i tam, Dum, Zit a nechat zit, Cerna hvězda, Musette, Cerveny most, Citova investice, Nic mi nechýbi, and Cyrilov.
- Data File:** An 'Upload' button.
- Description:** A large empty text area.
- Media Type:** Radio buttons for 'Don't know', 'Other', 'Video', 'Sound', 'Image', and 'Text'. 'Don't know' is selected.

Figure 3: Adding a new file resource

This screenshot shows the same 'Issue new' form after the 'Media Type' has been set to 'Sound'. The form now includes additional attributes:

- Song:** The same dropdown menu as in Figure 3, with 'Sediment' selected.
- Data File:** The text 'sediment.mp3' is entered, and the 'Upload' button is present.
- Description:** The text 'Remix with drumcomputer' is entered.
- Media Type:** The 'Sound' radio button is now selected.
- Mix:** Radio buttons for 'Don't know', 'Draft', 'Reference Track', 'Single Track', and 'Mix'. 'Don't know' is selected.
- Instruments:** A new dropdown menu with the following options: <Don't care>, <Empty>, Vocals, Accordeon, Guitar, Bass, and Drums.

Figure 4: Additional attributes depend on 'Media Type' attribute

3.3 Object References

Attributes can be of a base type (Number, String, Date etc) or of link type. In link type attributes, the values are items themselves. E.g. the 'Song' attribute points to items that represent a song.

Id	Caption	Source	bpm	Edit
328	Isel macek	Traditional		Edit
326	Zavidim vyhybce	Original		Edit
280	Plan	Original	118	Edit
230	Sediment	Original	106	Edit
214	Tady i tam	Original	74	Edit
119	Dum	Original	62	Edit
92	Zit a nechat zit	Original		Edit
89	Cerna hvezda	Original	130	Edit
35	Musette	Traditional		Edit
34	Cerveny most	Variation	84	Edit
33	Citova investice	Cover		Edit
32	Nic mi nechybi	Original		Edit
31	Cyrilov	Original	144	Edit

Figure 5: Available values for attribute 'Song': Items of a certain type

For example, the column 'Song' in figures 1 and 2 doesn't show a (direct) attribute value of the item, it rather shows the value of attribute 'Caption' in the referenced 'Song' item.

This is explained in more detail in chapter 4.

3.4 Historization

When the user changes attribute values of items or modifies the attribute and type structure, the old settings are not physically overwritten. The full change history is stored and you can e.g. run an item search for an arbitrary state, i.e. point in time. Chapter 5 explains the mechanism in use.

4 Dynamic Types

4.1 Types and Subtypes

A type T in Yablona has

- a set of zero or more attributes A_i that are applicable if an item I is of type T
- a condition (a restriction on attribute values) that specifies which items are of type T

Subtyping is expressed by defining appropriate conditions for types:

Definition: If the condition C_1 of a type T_1 is a subset of the condition C_2 of a type T_2 , i.e. C_2 is more restrictive than C_1 , then T_2 is a subtype of T_1 .

Figure 6 shows an excerpt of the 'Issue Tracker' configuration of Yablona (dark arrows: subtype to supertype. Light arrows: links):

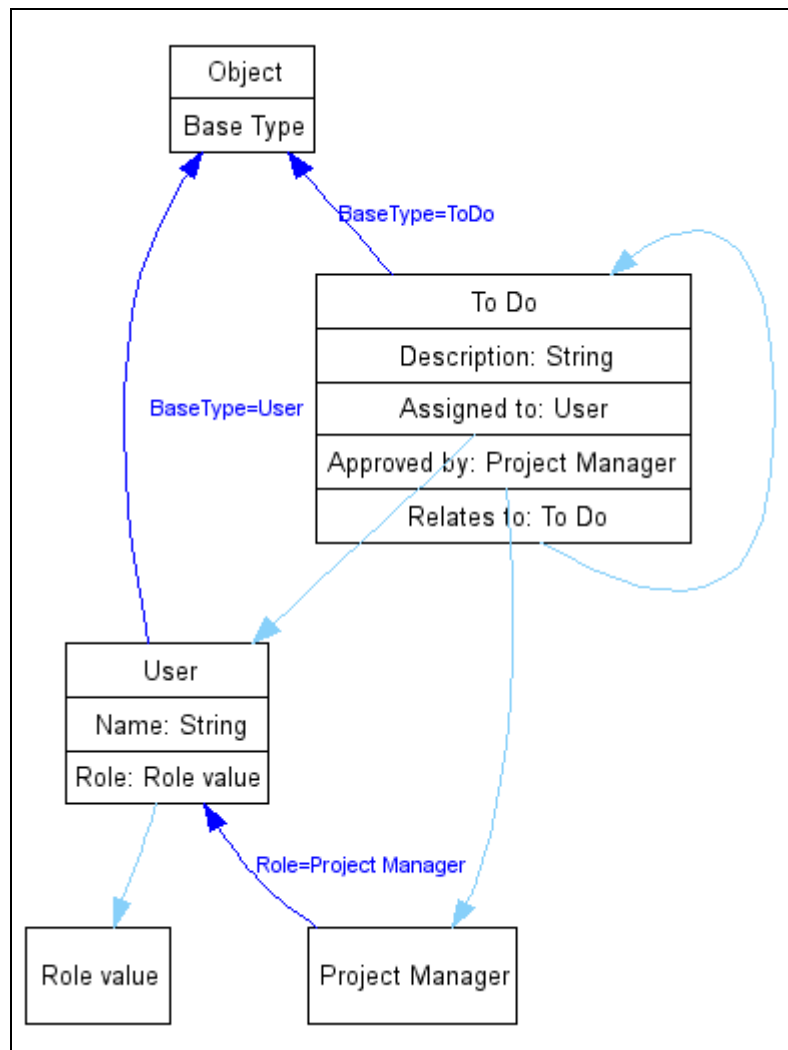


Figure 6: Class diagram

There are five types in the example above: *Object*, *To Do*, *User*, *Role Value* and *Project Manager*. The condition of type *User* (T_1) is $[BaseType=User]$, i.e. all items that point to a certain 'User' item in their *Base Type* attribute are of type *User*. The condition of type *Project Manager* (T_2) is $[BaseType=User; Role=Project Manager]$, i.e. it is more restrictive than type *User*. All *User* items that point to the 'Project Manager' item in their *Role* attribute are of type *Project Manager*. According to the definition above, *Project Manager* is a subtype of *User*.

Figure 7 shows an instance diagram for the types in figure 6 (the arrows show the references from the items' attributes to other items):

- Items I-1, I-2 and I-3 are of type *To Do* (their *Base Type* attribute points to I-31).
- Item I-10 is of type *User*, I-11 is of types *User* and *Project Manager*.

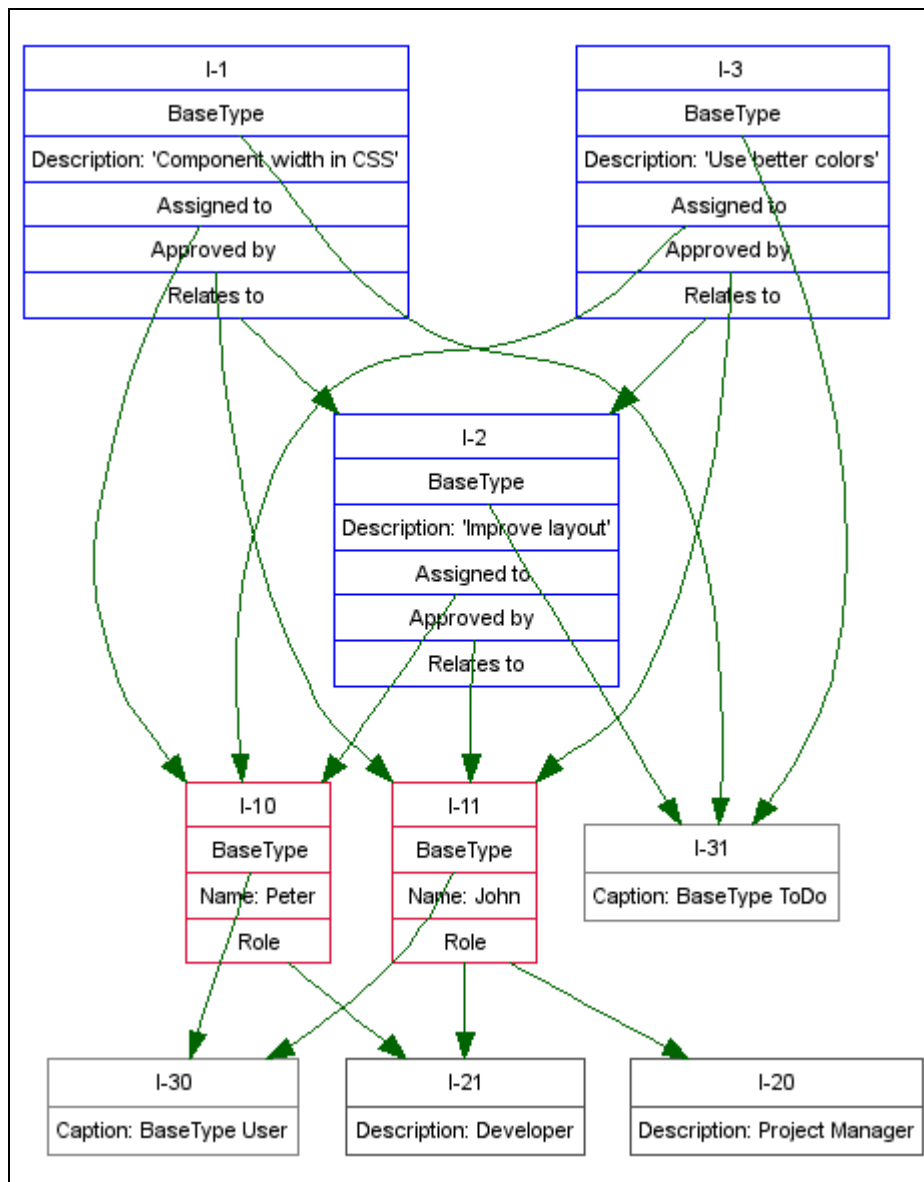


Figure 7: Instance diagram

4.2 Attributes

There are two types of attributes: Base type attributes (String, Number, Date, etc) and link attributes.

A base type attribute (string, number, date, etc) contains a simple value. Such a value does NOT have an impact on the type of an issue.

The values of a link attribute are items of a certain type. This needn't be a predefined type, it can be any filter on (link-) attribute values. In the example above, there is no need to define a type *Project Manager* (because it defines no attributes). We can define the filter of attribute *Approved by* in type *To Do* as [*BaseType=User; Role=Project Manager*], Such an attribute filter can be seen as an anonymous type definition. The type of an item is determined by the set of its link-type attribute values.

5 Update Points

Yablona is fully historized. This means you can

- View the complete change history of items
- Run queries for the current or any past state of Yablona

To store the full history of all objects in a SQL database, the following pattern to transform an ordinary SQL data model to a historized model is applied:

We first create a table 'UpdatePoint' with fields 'Id', 'Timestamp', 'User'. We then add the fields 'UpdatePoint' and 'Deleted' to all tables we want to historize. The 'UpdatePoint' fields reference the 'Id' of table 'UpdatePoint' and are added to the primary keys of the tables.

For every write transaction (that may contain several write operations to different tables), a new 'UpdatePoint' entry is created.

Each database operation from the original, unhistorized model is transformed into a new operation: Existing data is never modified (or deleted). Instead, for every write operation we insert new rows of data. The following table shows that transformation for an (example) table 'Person' with primary key 'Id' and a field 'Name', and update point ids 100, 101, 102:

Operation in original model	Operation in historized model
INSERT INTO Person(Id, Name) VALUES (1, 'John')	INSERT INTO Person(UpdatePoint, Id, Deleted, Name) VALUES (100, 1, false, 'John')
UPDATE Person SET Name='Johnny' WHERE Id=1	INSERT INTO Person(UpdatePoint, Id, Deleted, Name) VALUES (101, 1, false, 'Johnny')
DELETE FROM Person WHERE Id=1	INSERT INTO Person(UpdatePoint, Id, Deleted) VALUES (102, 1, true)
SELECT Id, Name FROM Person	SELECT Id, Name FROM Person p1, (SELECT MAX(UpdatePoint), Id FROM Person p2 WHERE p2.UpdatePoint <= 102 GROUP BY Id) m WHERE m.UpdatePoint = p1.UpdatePoint AND m.Id = p1.Id

If we want to retrieve data for a certain point in time (in the past), we simply enter the appropriate update point (102 in the example). The desired update point can be determined as follows:

```
SELECT MAX(Id)
FROM UpdatePoint
WHERE Timestamp <= '2005-01-26'
```

6 Architecture and Environment

6.1 Overview

Yablona is implemented in Java. The communication between the application server and the clients is based on stateless session EJBs and can be accessed by clients applications written in Java:

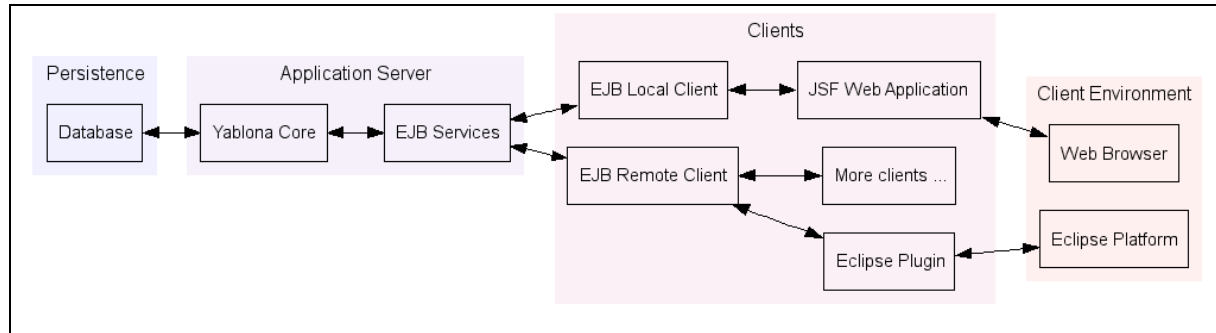


Figure 8: Architecture Overview

6.2 Persistence

Yablona data is persisted in a SQL database. As all business logic is implemented in Java, only very simple database operations are used. The Yablona database does not use stored procedures or any other database vendor specific features. Therefore, Yablona should work with most SQL databases. At the moment, HypersonicSQL is in use.

6.3 Application Server

The Yablona core, i.e. the dynamic data model engine, is wrapped with EJBs that specify the services available to clients. The EJBs are only used as a communication infrastructure between clients and server. Sophisticated EJB features like Entity Beans or Container Managed Persistence are not used in Yablona.

6.4 Clients and Client Environment

There are currently two clients available for Yablona:

- The Java Server Faces (JSF) web application to use Yablona in a web browser
- A plugin for the Eclipse platform. With this plugin, a developer can query and modify the ToDo list without switching to a different application (like a browser etc).

7 Future of Yablona

This chapter contains some ideas on possible future developments of Yablona. Feel free to participate in implementing these or researching new territories.

7.1 Events

To appropriately notify users of changes in certain items or to perform specific validations, it would be helpful to have events. For example:

Event	Event is fired when ...
EnterType(T)	... someone changes the attribute values of an item in a way that it is of type T after the change, but hadn't been of type T before.
Change(T)	... someone changes the attribute values of an item of type T (before and after the changes in type T)
Change(T, A)	... someone changes the value of attribute A in item of type T (before and after the changes in type T)
LeaveType(T)	... someone changes the attribute values of an item in a way that it is not of type T after the change, but had been of type T before.

Examples:

- Send me a mail if the priority of an item is set to 'high'
- Send me a mail if a new sound file is uploaded

7.2 Database

At the moment, Yablona data is persisted in a SQL database. SQL is used for practical reasons: It's already there. But it would be an interesting and quite challenging task to implement this database functionality specifically for Yablona. The projection of the data to an update point is quite an expensive operation in SQL and might lead to performance problems with growing database size. I suppose, this could be implemented efficiently by storing some indexes that give you fast access to items of a certain type at a certain update point.

7.3 File System

The media file library example is in some way similar to a file system, but a file can be in several folders at a time. Additionally, you can create a new folder (i.e. a new type) that already contains the matching files. Come on, folks, someone out there who would write the Yablona file system?

8 References

http://www.yablona.org	The home of Yablona
http://www.yablona.org/issuetracker	Yablona Issue Tracker
http://www.yablona.org/cyrilov	Yablona Media File Library
http://www.yablona.org/downloads.html	Source code for download