Magritte 2

Although I'm saying Magritte 2 this equally applies to Magritte 1 – the only difference between Magritte 1 and 2 was some changes for Seaside 30 compatibility

Magritte 3

Although I'm saying Magritte 2 this equally applies to Magritte 1 – the only difference between Magritte 1 and 2 was some changes for Seaside 30 compatibility

Magritte: Describe once, get everywhere

- Introspection
- Reflection
- Documentation
- Viewer building
- Form/Editor building
- Report building
- Data validation
- Query processing

- Object persistency
- Object indexing
- Object setup
- Object verification
- Object adaption
- Object customization
- and much more

Magritte is a meta-description framework - it allows you to add meta-data descriptions to your domain objects which can then be used to automatically generate reports, editors but not just that I'm finding it increasingly useful for serialising into other industry standard formats such as Json, XML then materialising the domain objects back from those formats.

Why Magritte 3

From: Lukas Renggli
November 2010

I wrote a proposal this summer of want to proceed on
that, but of course I had to finish my writing and never actually
found the time to implement it:

I propose to perform the following (non-backward compatible) changes in the Magritte 2 code-base to resolve some major annoyances and issues that keep on reoccurring:

- Move descriptions from class-side to instance-side. This resolves various issues such as cache-invalidation, instance specific descriptions, dynamic descriptions, context dependent descriptions, etc. Furthermore the descriptions will be closer to the code they describe and it will be possible to describe class- and instance-side of an object, not just the instance-side.

- Rename the method #description as the default entry point into Magritte to #magritteDescription. This avoids common problems where the domain model already defines such a method.

- Instead of using a naming convention for description methods, use a pragma called <magritte> to annotate the methods. And to extend and change existing descriptions use <magritte: aSelector>. Finally all Smalltalk implementation reached a consensus of pragmas that can be safely used cross-platform.

All in all the "new" Magritte would look like in the following example. Imagine a shop item with the accessor #place:

```
 Item>>place
    ^ place

 Item>>place: aString
    place := aString
```

The meta-description is defined on the instance-side and annotated. It can refer to itself for the possible places:

```
 Item>>placeDescription
     <magritte>

    ^ MASingleOptionDescription new
        options: self possiblePlaces;
        label: 'Place of Item';
        accessor: #place;
        yourself
```

Class extensions can modify a description using:

```
 Item>>placeDescriptionXmlStorage: aDescription
    <magritte: #placeDescription>

    ^ placeDescription xmlTag: 'xname'
```

Since these changes are not backward compatible I'll try to provide automatic refactorings for most parts. Moving existing code to the new codebase will certainly cause some problems, but in the long run I believe this to be a much better approach than the current one. If people have any feedback, concerns or other changes that would be important in the same run I am happy to hear them.

Issues with Magritte2

- Name collision with `#description`
- class-side description:
  - Cache-invalidation
  - dynamic descriptions
  - instance specific descriptions
  - context dependent descriptions

Magritte 3

allows renaming so that descriptions can be along-side the accessors

Magritte 3

- ✓ Descriptions move from class to instance.

allows renaming so that descriptions can be along-side the accessors

Magritte 3

- ✓ Descriptions move from class to instance.
- ✓ Use `<magritteDescription>` pragmas rather than naming convention.

allows renaming so that descriptions can be along-side the accessors

Magritte 3

- ✓ Descriptions move from class to instance.
- ✓ Use `<magritteDescription>` pragmas rather than naming convention.
- ✓ All descriptions are dynamically generated.

allows renaming so that descriptions can be along-side the accessors

Magritte 3

- ✓ Descriptions move from class to instance.
- ✓ Use `<magritteDescription>` pragmas rather than naming convention.
- ✓ All descriptions are dynamically generated.
- ✓ Rename `#description` to `#magritteDescription`

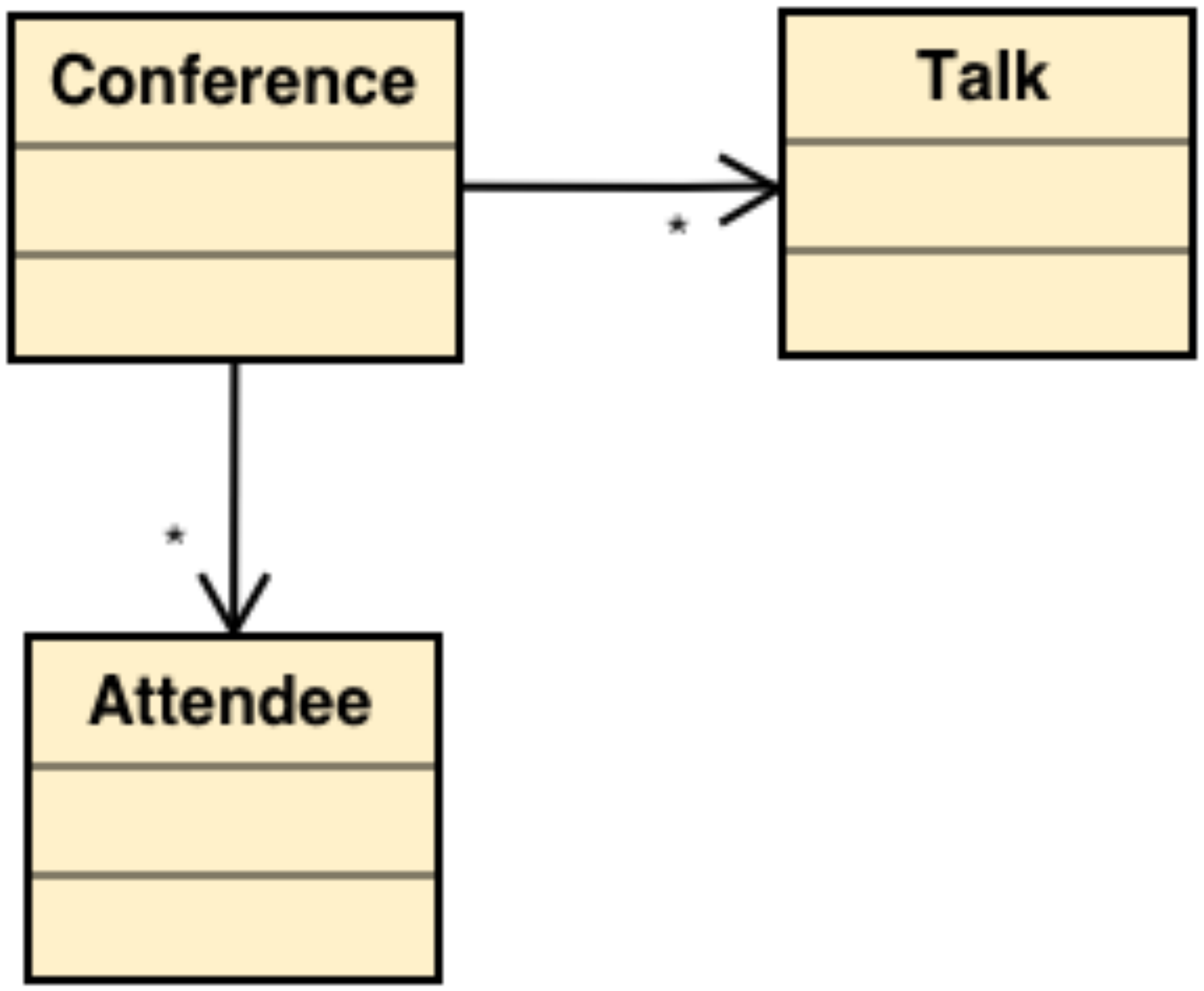
allows renaming so that descriptions can be along-side the accessors

Magritte 3

- ☑ Descriptions move from class to instance.
- ☑ Use `<magritteDescription>` pragmas rather than naming convention.
- ☑ All descriptions are dynamically generated.
- ☑ Rename `#description` to `#magritteDescription`
- ☐ Non-backward compatible

allows renaming so that descriptions can be along-side the accessors

Demo



Magritte 2	Magritte 3
#description	#magritteDescription
class-side descriptions	instance-side description
#descriptionXXX	<magritteDescription>
#magritteDynamicObject	Not required
#descriptionContainer	<magritteContainer>

Refactoring support

```
Gofer it
   squeaksource: 'MetacelloRepository';
   package: 'ConfigurationOfMagritte3';
   load.

ConfigurationOfMagritte3 project stableVersion load:'Tools'
```

Magritte-Deprecated - describe
Magritte-Pharo-Tools

Refactoring demo

```
MMAttendee class>>#descriptionBookingDate
  ^ MADateDescription new
    priority: 30;
    label: 'Booking date';
    tooltip: 'When the attendee booked';
    accessor: #bookingDate;
    default: [ Date today ] magritteDynamicObject;
    yourself
```

```
MMAttendee class>>#descriptionBookingDate
  ^ MADateDescription new
    priority: 30;
    label: 'Booking date';
    tooltip: 'When the attendee booked';
    accessor: #bookingDate;
    default: [ Date today ] magritteDynamicObject;
    yourself
```

```
MMAttendee>>#descriptionBookingDate

  ^ MADateDescription new
    priority: 30;
    label: 'Booking date';
    tooltip: 'When the attendee booked';
    accessor: #bookingDate;
    default: [ Date today ] magritteDynamicObject;
    yourself
```

```
MMAttendee class>>#descriptionBookingDate
  ^ MADateDescription new
    priority: 30;
    label: 'Booking date';
    tooltip: 'When the attendee booked';
    accessor: #bookingDate;
    default: [ Date today ] magritteDynamicObject;
    yourself
```

```
MMAttendee>>#descriptionBookingDate
  <magritteDescription>
  ^ MADateDescription new
    priority: 30;
    label: 'Booking date';
    tooltip: 'When the attendee booked';
    accessor: #bookingDate;
    default: [ Date today ] magritteDynamicObject;
    yourself
```

```
MMAttendee class>>#descriptionBookingDate
  ^ MADateDescription new
    priority: 30;
    label: 'Booking date';
    tooltip: 'When the attendee booked';
    accessor: #bookingDate;
    default: [ Date today ] magritteDynamicObject;
    yourself
```

```
MMAttendee>>#descriptionBookingDate
  <magritteDescription>
  ^ MADateDescription new
    priority: 30;
    label: 'Booking date';
    tooltip: 'When the attendee booked';
    accessor: #bookingDate;
    default: Date today;
    yourself
```

Refactoring

- ✓ Move class-side descriptions to instance-side
- ✓ add `<magritteDescription>`
- ✓ Works for description extensions.
- ✓ Operates at a method, class or package level
- ✓ Battle-tested in translating Pier

Refactoring will not

- ☐ remove empty class-side categories
- ☐ convert class-side message sends within descriptions to add `class`
- ☐ convert calls from `#description` to `#magritteDescription`
- ☐ Simplify Magritte 1 or 2 dynamic description work-arounds
- ☐ Remove `#magritteDynamicObject` and it's associated block

Porting guidelines

1. Use the refactoring support to move class-side descriptions to instance side descriptions with pragmas - making sure that any accessors to class side methods are either prefixed with ‘class’ or moved to the instance side.
2. remove `#magritteDynamicObject` and remove the block around the method.
3. search for all senders and implementors of `#description` in your code, if they are Magritte descriptions rename the selector from `#description` to `#magritteDescription`
4. Remove any empty categories on the class side.

Status

- ☑ All tests green.
- ☑ Although the API changes appear significant the code changes within Magritte are less significant and mainly localised to `MAPragmaBuilder`
- ☑ Pier has been ported to Magritte 3 (Pier 3) and is stable.
- ☑ `Magritte-Json` and `Magritte-XMLBinding` ported
- ☐ Other Magritte add-ons such `Magritte-JQuery` un-ported
- ☐ No performance testing - descriptions are no longer cached. Is this significant in the context of an application?

Next steps

- Install:

```
Gofer it
    squeaksource: 'MetacelloRepository';
    package: 'ConfigurationOfMagritte3';
    load.

ConfigurationOfMagritte3 project stableVersion load:'Seaside'.

Gofer it
    squeaksource: 'MetacelloRepository';
    package: 'ConfigurationOfMagritte3AddOns';
    load.

ConfigurationOfMagritte3 project stableVersion load.
```

- Use the refactoring tools to move to Magritte 3.
- Profile performance within your application.

Load Sample

```
Gofer it
   url: 'http://ss3.gemstone.com/ss/MagritteMagic';
   package: 'ConfigurationOfMagritteMagic';
   load.

(Smalltalk at: #ConfigurationOfMagritteMagic) load.
```

Further information

- Seaside Book:
  http://book.seaside.st/book/advanced/magritte
- Lukas's site:
  http://www.lukas-renggli.ch/smalltalk/magritte
- Magritte/Pier mail list:
  <smallwiki@iam.unibe.ch>