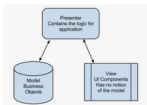


GUI architecture for scientific workflow

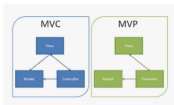
spaghetti-free, lazy-responsive, with capture & replay

Joachim Wuttke

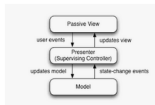
Forschungszentrum Jülich GmbH, JCNS-MLZ Garching



Model View Presenter - Wikipedia
de.wikipedia.org



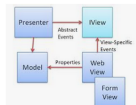
What is Model View Presenter? - Stack Overflow
stackoverflow.com



Model-view-presenter - Wikipedia
en.wikipedia.org



the difference between model-view-viewModel a...
blogs.msdn.microsoft.com



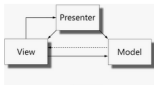
Model-View-ViewModel (MVVM) Explained
writeurl.net



Model View Controller (MVC) and Model View Presenter (M...
gintereid.com



UI Design Using Model-View-Presenter (P...
informatics.dcu.academy.ru.com



Interactive Application Architecture Patterns - Asp...
aspnetmag.affinity.com



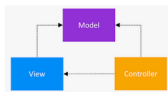
A typical Model-View-Controller (MVC) softwa...
newarchitect.net



1: Model-View-Controller (MVC) design pattern. C...
newarchitect.net



Model-View-Controller design pattern
itam.com



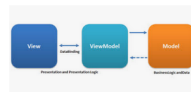
Android Architecture Patterns Part 1: Model-View-Contr...
medium.com



Model View Presenter via .NET - CodeProject
codeproject.com



the difference between model-view-viewModel a...
blogs.msdn.microsoft.com



Model View ViewModel - Wikipedia
de.wikipedia.org



Modern MVC - iOS App Development - Medium
medium.com



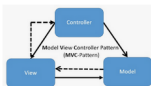
the difference between model-view-view...
blogs.msdn.microsoft.com



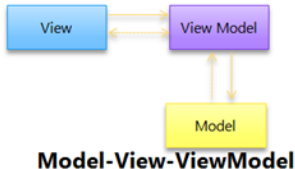
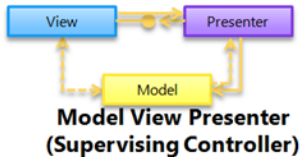
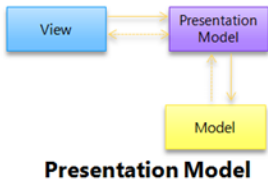
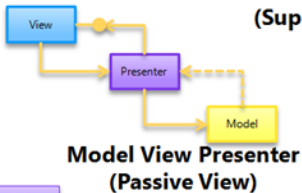
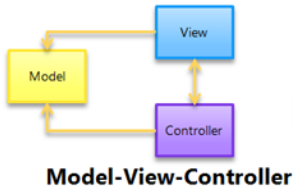
Android Architecture Patterns Part 2: Model-View-Presenter
medium.com



A practical guide to MVP for Vaadin - M...
mikaas.rocks



Model View Controller Pattern Definition & Erklärung...
datenbanken-ventzen.de

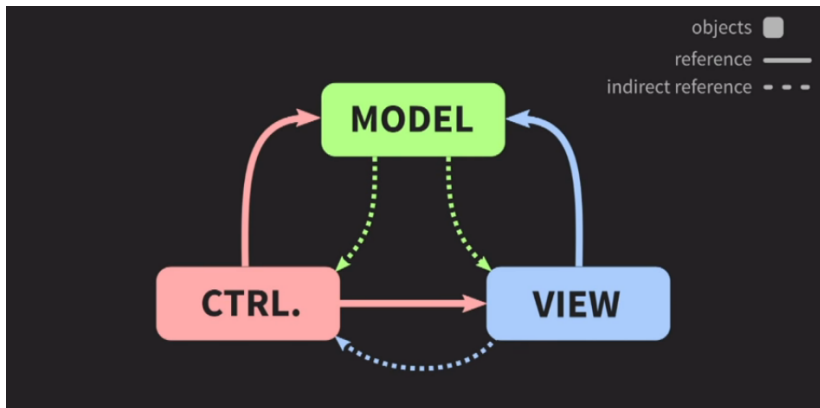


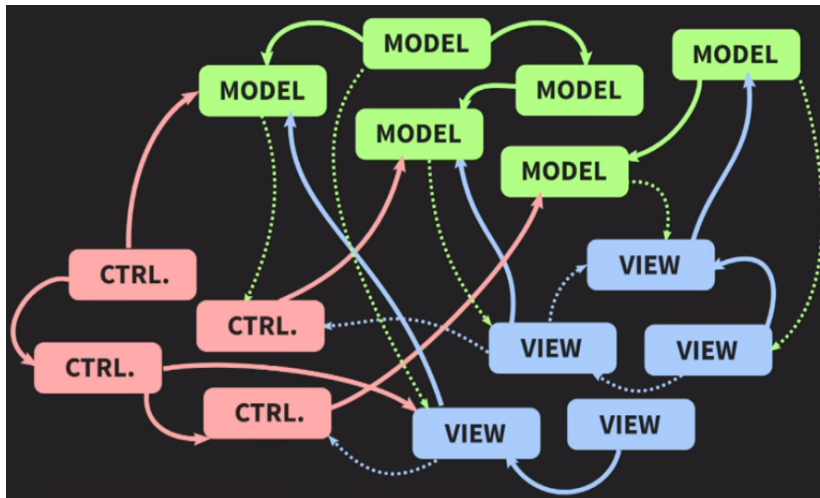
view is based on

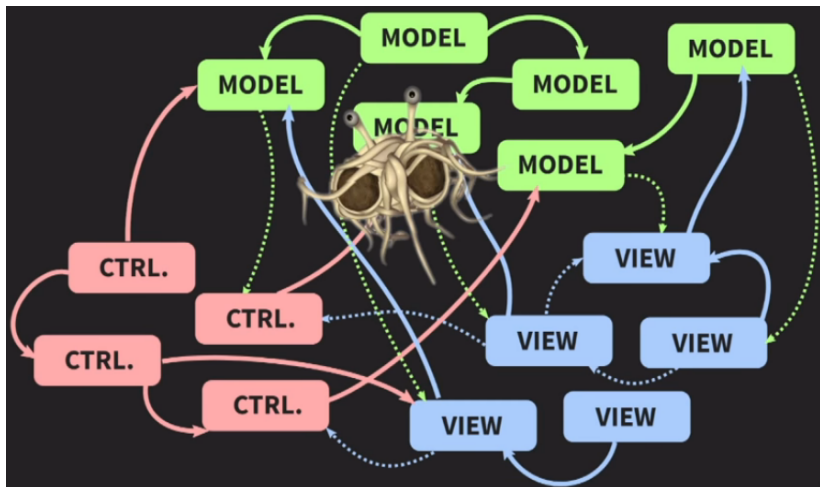
```
QAbstractItemView  
- QListView  
- QTableView  
- QTreeView
```

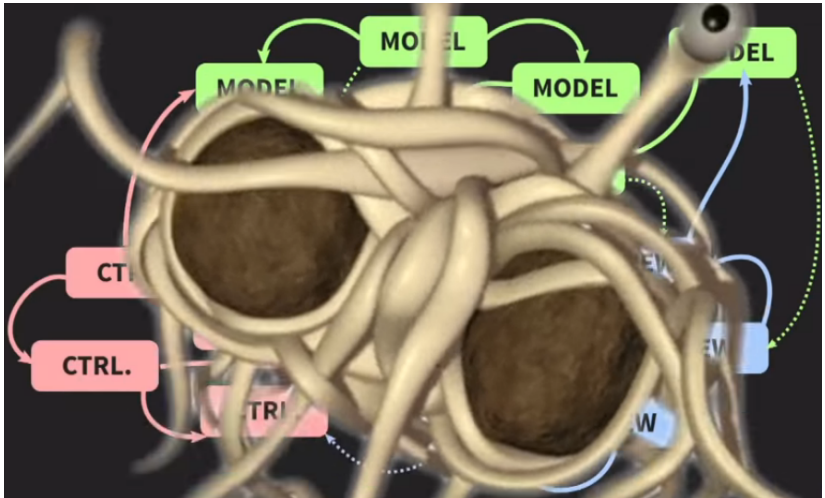
model inherits from

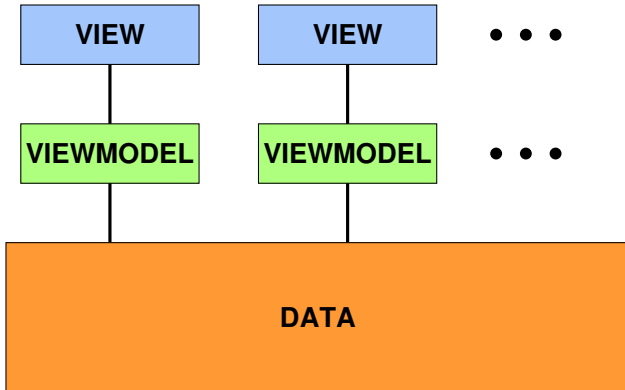
```
QAbstractItemModel  
- QAbstractListModel  
- QAbstractProxyModel  
- QAbstractTableModel
```

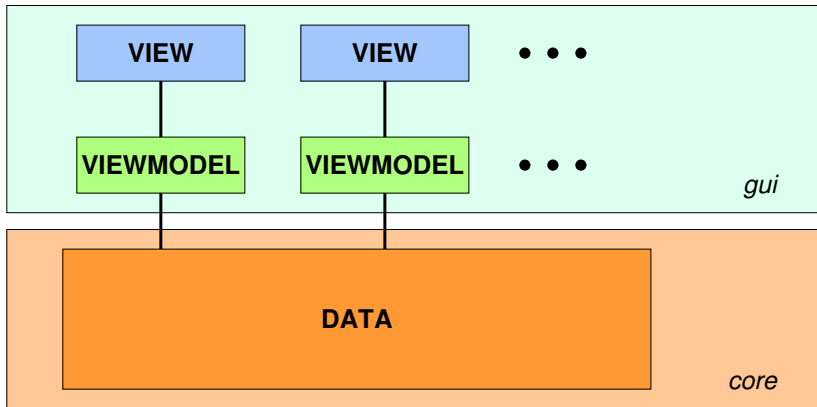


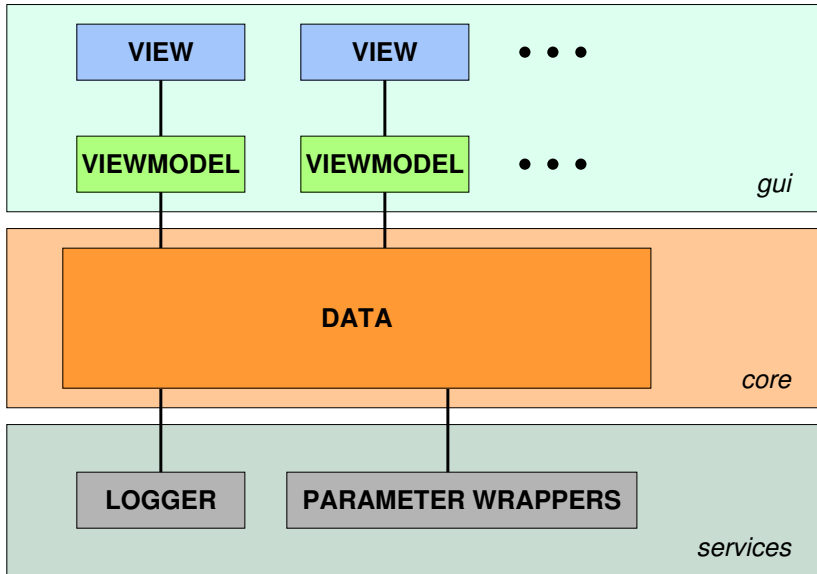


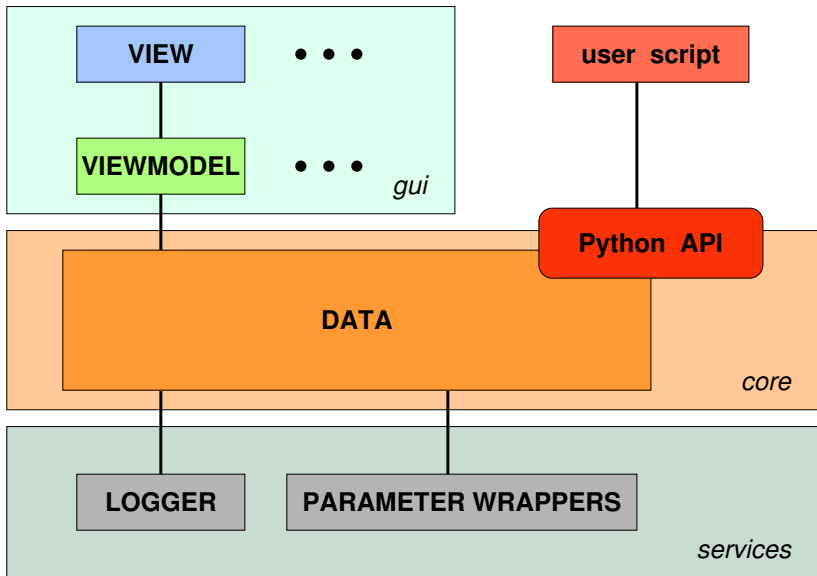






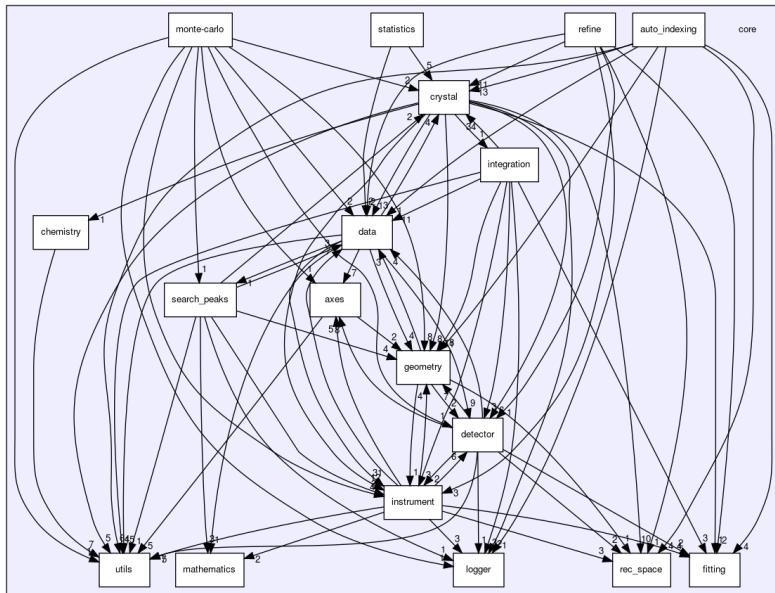


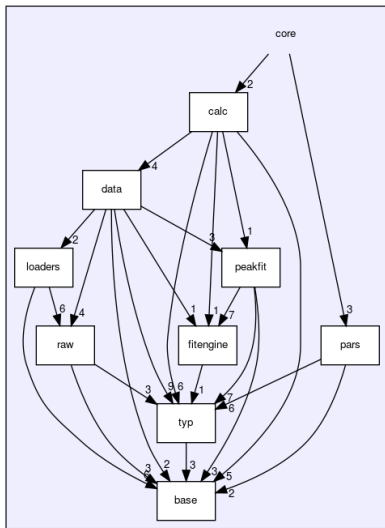
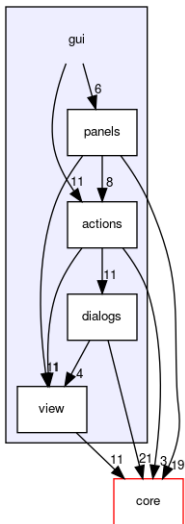




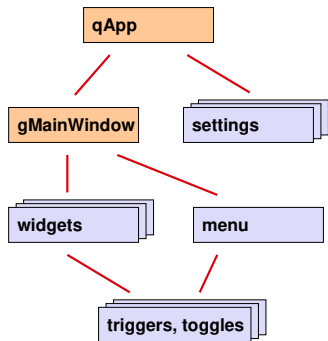
My Project

Main Page	Namespaces ▾	Classes ▾	Files ▾
<h2>File List</h2>			
Here is a list of all files with brief descriptions:			
<hr/>			
<div><div>▼</div><div>core</div><div>▶ auto_indexing</div><div>▶ chemistry</div><div>▶ crystal</div><div>▶ detector</div><div>▶ experiment</div><div>▶ find_peak</div><div>▶ geometry</div></div>			

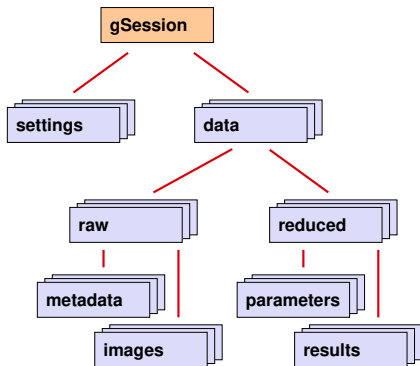





GUI:



Core:



 = *global variable*

- prefer plain global variables over equivalent singletons
- prefer one global variable `gSession` over function calls like

```
f(session.Data, session.Parameters);
```

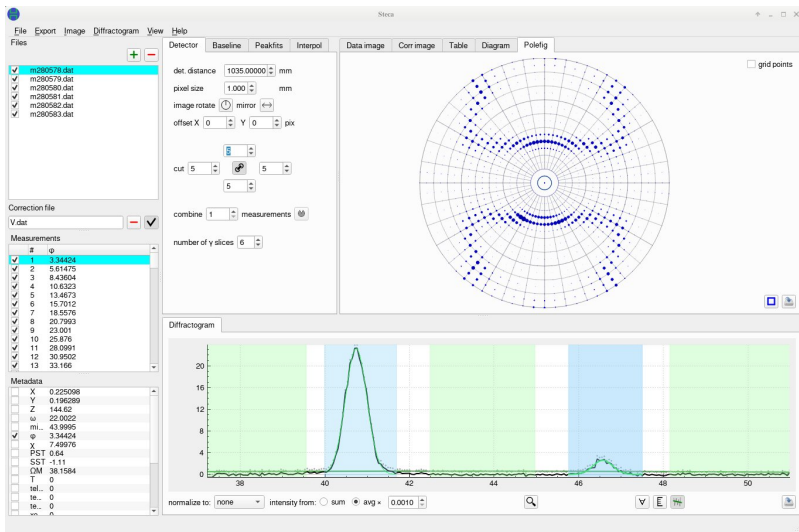
which calls

```
g(datafiles[iFile], parameterX, parameterZ);
```

which calls

```
h(datafile.metadata.A, parameterZ);
```

How to keep this up to date?



Signalling spaghetti (GUI \leftrightarrow Core) has its root in efforts

- to restrict Core recomputation to data that have changed,
- to restrict GUI redrawing to Widgets that have changed.

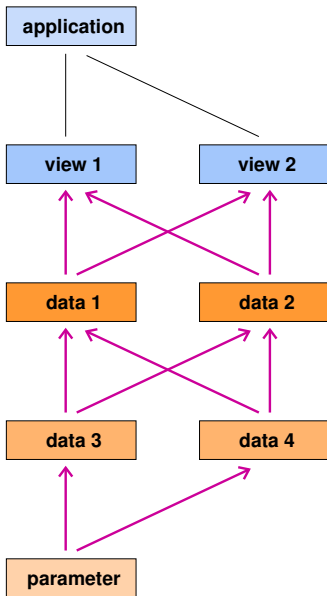
Both are premature optimizations.

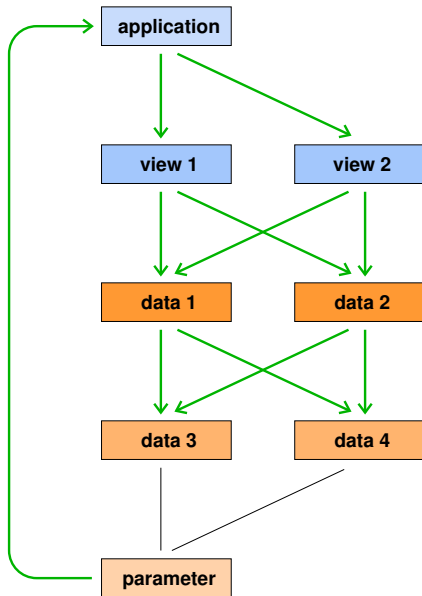
Signalling spaghetti (GUI \leftrightarrow Core) has its root in efforts

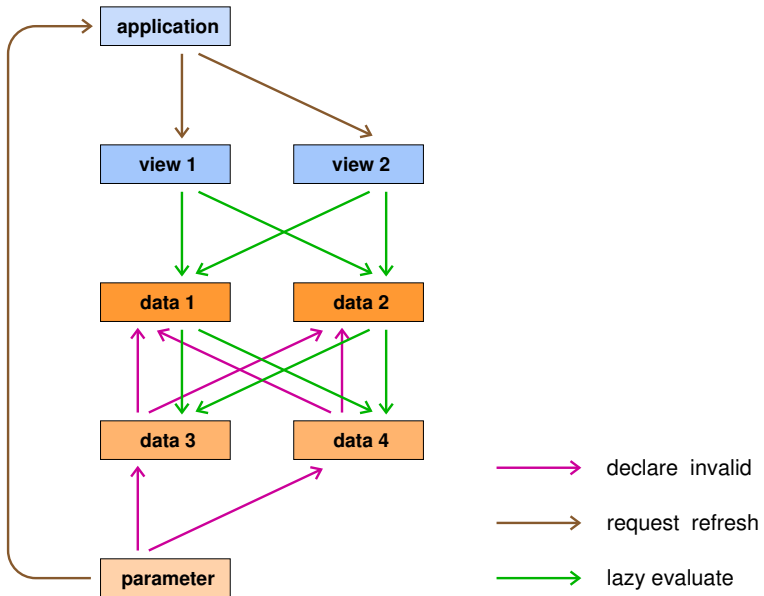
- to restrict Core recomputation to data that have changed,
- to restrict GUI redrawing to Widgets that have changed.

Both are premature optimizations.

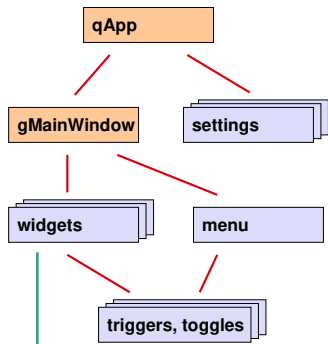
They do not even preclude duplication of Core computations.



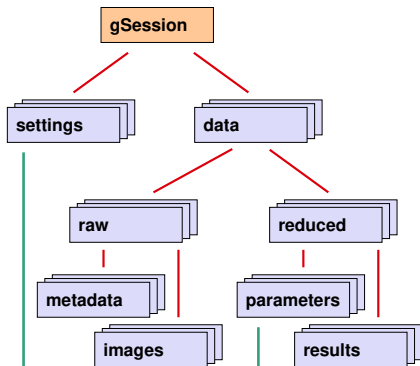




GUI:



Core:



request redisplay ←

→ *request recompute*

Bonus section: capture & replay

Why?

- to debug,
- to profile,
- to document provenance,

and to replay

- tests during development,
- functional tests,
- user sessions, especially upon bug reports.

Also related:

- Undo/Redo.

How?

By recording

- keyboard and mouse events,
- interrupts,
- user actions at widget level,
- GUI-to-core calls.

How?

By recording

- keyboard and mouse events,
- interrupts,
- user actions at widget level,
- GUI-to-core calls.

