impose the restriction that only image formats can be used where the drivers support scaling during load. It is also assumed that the files are present on the hard disk. Since the processing of each file does not influence or depend on the processing of any other, all files can be processed in parallel. The individual three steps to process one file need to be performed in sequence.

But that is not all. Since this is an example with a graphical user interface, the expectations of the user have to be kept in mind. It is assumed that the user is interested in visual feedback, which also gives him the impression of progress. Image previews should be shown as soon as they are available, and progress information in the form of a reliable progress bar would be nice. The user also expects that the program will not grind his computer to a halt, for example by excessive I/O operations.

Different ThreadWeaver tools can be applied to this problem. First of all, processing an individual file is a sequence of three job implementations. The jobs are quite generic and can be part of a toolbox of premade job classes available to the application. The jobs are (the class names match the ones in the example source code):

- A `FileLoaderJob`, which loads a file on the file system into an in memory byte array
- A `QImageLoaderJob` to convert the image's raw data into the typical representation of an in-memory image in Qt applications (providing the application access to all available image decoders in the framework or registered by the application)
- A `ComputeThumbNailJob`, which simply scales the image to the wanted size of the preview

All of those are added to a `JobSequence`, and each of these sequences is added to a `JobCollection`. The composite implementation of the collection classes allow for implementations that represent the original problem very closely and therefore feel somewhat natural and canonical to the programmer.

This solves part one of the problem, the parallel processing of the different images. It could easily lead to other problems, though. With the given declaration of the problem to the ThreadWeaver queue, there is nothing that prevents it from loading all files at once and only then starting to process images. Although this is unlikely, we haven't told the system otherwise yet. To make sure that only so many file loaders are started at the same time, a resource restriction is used. The code for it looks like this:

```
#include "ResourceRestrictionPolicy.h"

...

static QueuePolicy* resourceRestriction()
{
    static ResourceRestrictionPolicy policy( 4 );
    return &policy;
}
```