SuperElastix Documentation

Release 0.1

SuperElastix Developers

Apr 28, 2017

Contents

The objective of image registration is to find the spatial relationship between two or more images. In the last decades numerous image registration methods and tools have emerged from the research community. Implementation of these methods, however, are scattered over a plethora of toolboxes each with their own interface, limitations and modus operandi.

SuperElastix is a joint effort of the Biomedical Imaging Group Rotterdam (BIGR) of Erasmus University Medical Center, The Netherlands, and the Division of Image Processing (LKEB) of Leiden University Medical Center, The Netherlands, to provide an open source, multi-platform image registration toolbox written in C++. SuperElastix aims at covering a wide range of image registration methodologies in a single experience, while considering both user-friendliness and algorithm modularity.

Contents:

CHAPTER 1

Getting Started

This page explains how to install SuperElastix.

SuperBuild On Linux

The SuperElastix SuperBuld is a script that automatically downloads and installs dependencies before compiling the main project. SuperElastix depends on ITK and elastix. The following steps assume that CMake, git and a compiler toolchain is installed. To build SuperElastix, clone the repository and invoke the SuperBuild.

```
$ git clone https://github.com/SuperElastix/SuperElastix
$ mkdir build
$ cd build
$ cmake ../SuperElastix/SuperBuild
$ make -j4
```

To check that SuperElastix was build successfully, go into the `SuperElastix-build` directory and run the test suite with the following command

```
ctest -j4
```

Warning: Be careful not to run out of memory during the build. A rule of thumb is that we need 4GB of memory per core. For example, if we compile SuperElastix with 4 cores (e.g. `make -j4`) we need a machine with at least 16GB of RAM.

SuperBuild On Mac OS X

The Mac OS X installation procedure is identical to that of Linux, so simply follow the Linux installation steps above to install SuperElastix. It is assumed that a CMake, git and a compiler toolchain has already been installed. We can

check for a working compiler by opening the OS X terminal and run `make`. OS X will know if the Xcode Command Line Tools is missing and prompt you to install them if this is the case.

SuperBuild On Windows

In this guide we will use CMake to generate build files and the Visual Studio compiler to compile the project.

1. **Setup directories.**
    - Download and install CMake GUI.
    - `git clone https://github.com/SuperElastix/SuperElastix` into a source folder of your choice.
    - Point the CMake source directory to the `SuperElastix/SuperBuild` folder inside the source directory.
    - Point the CMake build directory to a clean directory. Note that Visual Studio may complain during the build if the path is longer than 50 characters. Make a build directory with a short name at the root of your harddrive to avoid any issues.
2. **Select compiler.**
    - Press configure to bring up the compiler selection window.
    - Preferably, choose Visual Studio 12 2013 Win64 as the generator for the project. SuperElastix can only be compiled with Visual Studio 12 or later, and if the OS is 64-bit that is also the prefered for the compiler.
3. Open Visual Studio, select File -> Open Project/Solution -> Open and choose `SuperElastixSuperBuild` solution.
4. Make sure "Release" build type is selected and build the `ALL_BUILD` project.
5. **Right-click on `ALL_BUILD` and click `Build`.**
    - When building in Visual Studio 2015 you will get a number of error messages during compilation of ITK. Just press the 'Ignore' button for all of them.
6. The SuperBuildSuperElastix solution only shows each library as a project. To have a more detailed view of SuperElastix open the SuperElastix solution file `<build-path>\SuperElastix-build\SuperElastix.sln` in a new Visual Studio environment.
7. Optionally, Unit Tests and example code can be run by right-clicking on `RUN_TESTS` and clicking `Project Only -> Build Only RUN_TESTS`. Alternatively, individual tests can be run by right-clicking on a specific selx<...>Test project and choosing `Set as StartUp Project`.
8. The SuperElastix commandline executable can be found in `<build-path>\Applications-build\CommandLineInte`

Manually Building the Required Libraries

Instead of letting the SuperBuild download and build the required libraries manually build libraries can be used as well. The following approach allows us to use a system version of ITK or our own version of elastix.

1. **Build ITK.**
    - Clone ITK from github.com/InsightSoftwareConsortium/ITK.

- Configure CMake. Set the following CMake variables: BUILD_SHARED_LIBS=OFF, ITK_USE_REVIEW=ON, ITK_WRAP_*=OFF.
- Compile ITK. Make sure to note the build settings, e.g. Release x64.

2. Build Boost
3. **Build elastix.**
    - Clone elastix from github.com/kaspermarstal/elastix.
    - Set ITK_DIR to the location of the ITK build directory
    - Configure CMake. Set the following CMake variables: BUILD_EXECUTABLE=OFF, USE_KNNGraphAlphaMutualInformationMetric=OFF
    - Set appropriate ELASTIX_IMAGE_2/3/4D_PIXELTYPES and any components that you might require.
    - If you are developing your own elastix components, make sure they are properly registered by the elastix build system.
    - Compile elastix. Make sure to configure the build settings exactly the same as ITK e.g. Release x64.
4. **Build SuperElastix.**
    - Clone SuperElastix from github.com/SuperElastix/SuperElastix.
    - Configure CMake. Point ITK_DIR to the location of the ITK build directory and ELASTIX_DIR to the location of the elastix build directory.
    - Build SuperElastix. Make sure to configure the build settings exactly the same as ITK e.g. Release x64.

CHAPTER 2

Usage

This page explains how to use SuperElastix.

Commandline tool

The SuperElastix commandline tool can be found at:

- Windows: :code:‘<build-path>Applications-buildCommandLineInterface[Release|Debug]‘
- Linux: `<build-path>/Applications-build/CommandLineInterface/`

Demo Experiments

To run the demo experiments SuperElastix needs to be installed:

- Windows:
  - open `<build-path>\Applications-build\SuperElastixApplications.sln`
  - in project solution explorer right-click on InstallDemo -> Project Only -> Build Only InstallDemo
- Linux:
  - change dir to `<build-path>/Applications-build/`
  - `make InstallDemo`

By building the `InstallDemo` target the SuperElastix executable, image data, configuration files and commandline scripts will be copied to the `<INSTALLDEMO_PREFIX>` directory. The `<INSTALLDEMO_PREFIX>` can be set by CMake and defaults to `<build-path>/SuperElastixApplications-build/Demo` These four scripts run the Demo experiments as described in[1]:

- `1A_SuperElastix_elastix_NC`

[1] *The design of SuperElastix - a unifying framework for a wide range of image registration methodologies, F. Berendsen, K. Marstal, S. Klein and M. Staring*, found at `<source-path>\Documentation\source\paperWBIR16`.

- 1B_SuperElastix_elastix_MSD
- 2A_SuperElastix_itkv4_NC
- 2B_SuperElastix_itkv4_MSD

Library use

SuperElastix supports two ways of library usage:

- SuperElastixFilter: A precompiled SuperElastixFilter library loaded with a default set of components. To link against this library a small number of header files are needed. Compilation time is limited, since most functionality is precompiled and not exposed during compilation of the application (pimpl idiom)
- SuperElastixCustomComponents<ComponentTypeList<...>>: A templated SuperElastixFilter class with a user selectable set of components. The set of components can be of any number and of any variation of each component its template arguments, e.g. dimensionality, pixeltype or other template arguments. New user-developed components can be added to the ComponentTypeList and compiled into the filter without the need of changing any of the SuperElastix source codes. Compilation is longer and it requires the project to include all header of SuperElastix and the selected components.

The SuperElastix commandline tool in <source-path>\Applications serves as an example of the usage of the SuperElastix library. [in progress]

CHAPTER 3

Development

With SuperElastix we aim to capture a wide range of registration methods, accessible via a single high-level user interface. At the core of our design is a single collection of components with heterogeneous levels of functionality and granularity. This means that a component can implement a single concept (metric, transform, etc.) to be reused in many methods, can implement multiple (tightly coupled) concepts in one, or even be a full registration algorithm. Breaking up algorithms in small components allows a user to mix-and-match component, whereas treating a larger part of algorithms as monolithic blocks lowers the barrier for integration of new methods and paradigms and reuse of existing codebases. By a high level user configuration, components are selected at run-time and are connected via a user-defined network layout. A generic handshake mechanism verifies whether connected components are compatible both mathematically as on a software level. The user is notified if specific combinations are incompatible or not supported (yet).

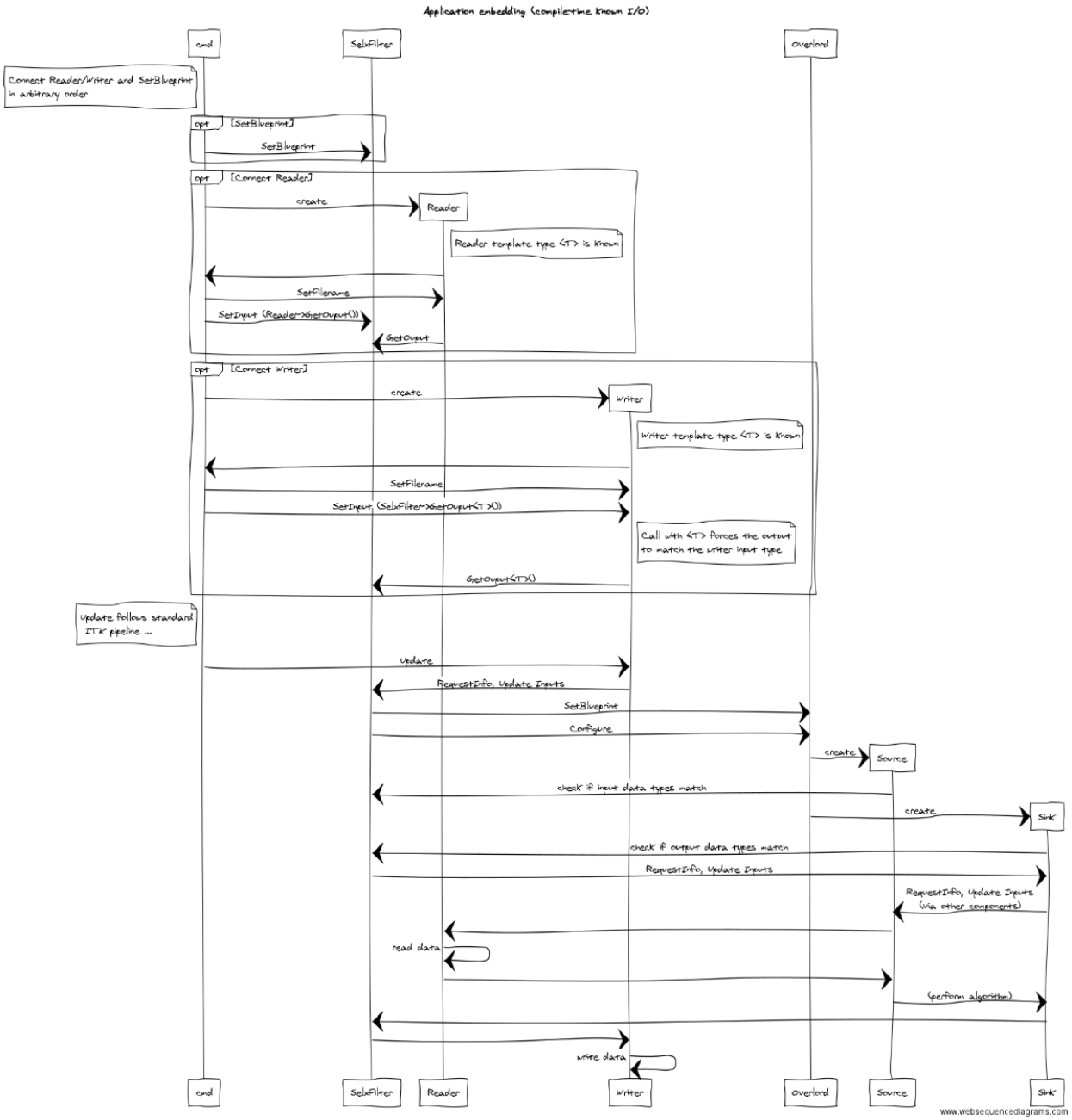
SuperElastixFilter input and output datatypes

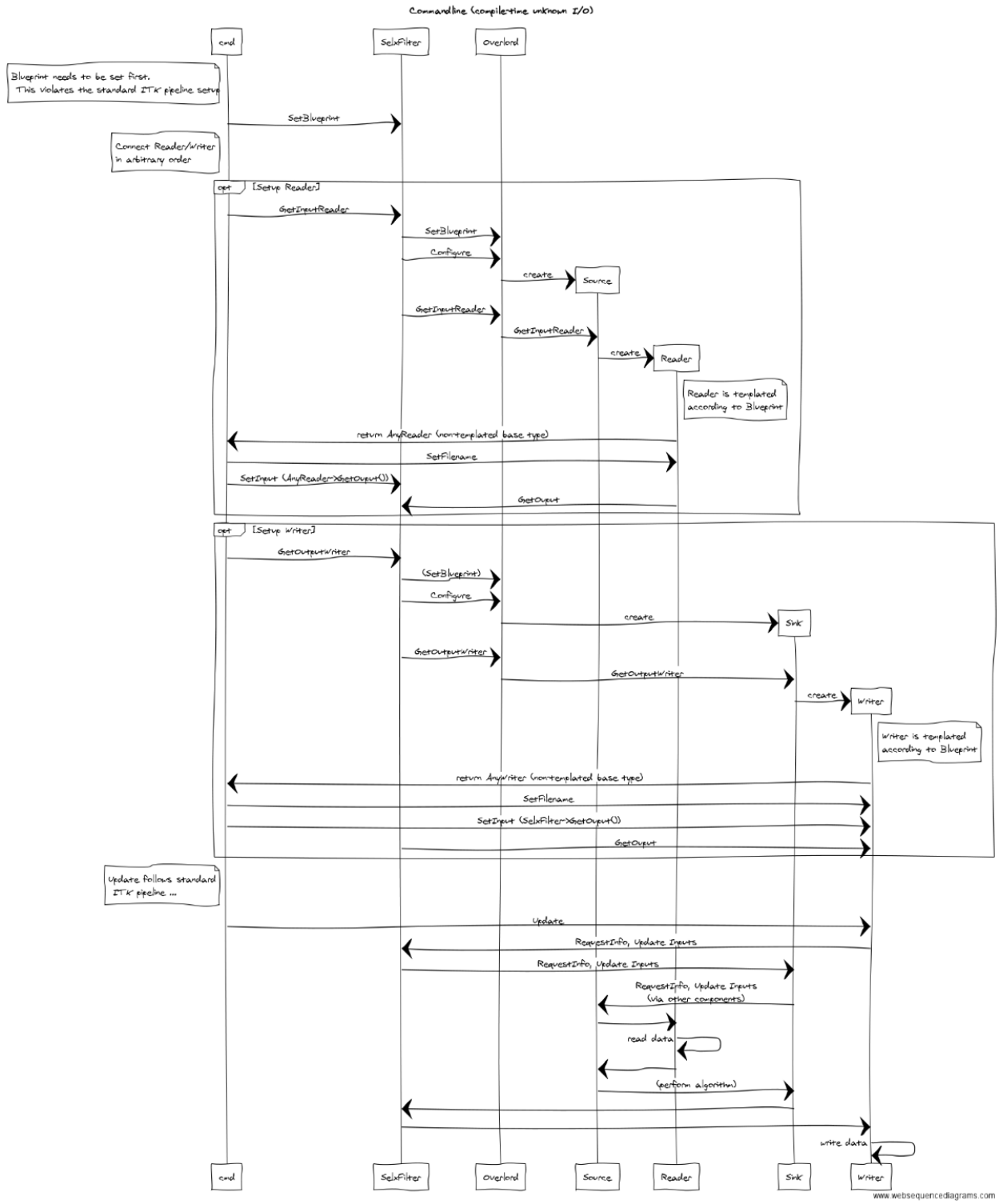
The SuperElastixFilter is designed to be part of an itk pipeline such that it can connect to other itk filters and supports the itk update mechanism. All input and output data required by the (configured) algorithm are exposed by the SuperElastixFilter. In this philosophy the SuperElastixFilter does/should not read or write data (images, meshes, etc) from disk directly. Therefore, in the commandline tool, which uses the SuperElastixFilter, readers and writers reside outside the SuperElastixFilter. However, unlike common itk filters, the inputs and outputs of the SuperElastixFilter are typically unknown at compile time, because they depend on the Blueprint configuration describing the actual algorithm to execute. This complicates the setup of a pipeline, since up and downstream itk filters are typically templated over their datatypes. To stay as close as possible to the itk philosophy, the SuperElastixFilter supports 2 modes of operation:

- *Known input and output types at compilation time*: E.g. an application embedding a dedicated registration task. That is, the application developer makes sure that any Blueprints to be used, will correspond to the (compile-time defined) number and types of inputs and outputs by known identifier names (defined by the Sink and Source Components). In this mode, the order in which the inputs and outputs are connected to other filters, and the Blueprint (Object) is set, is arbitrary. However, to connect the output of the SuperElastixFilter a templated version of GetOutput must be used: `ImageFileWriter<KnownImageType>::Pointer my_writer; ... my_writer->SetInput(superElastixFilter->GetOutput<KnownImageType>(identifier))`.

- *Unknown input and output types at compilation time*: E.g. the class implementing the commandline interface is not aware of the datatypes used by all components. (In this way, adding custom components with new types does not affect the source code of the commandline interface). The commandline interface is invoked by pairs of filenames and identifier names. The identifiers refer to Sink or Source Components as defined via the Blueprint that, in turn, define the data types. In this mode, the commandline interface typically cannot instantiate readers or writers because they are templated over the data types. Instead, the SuperElastixFilter is requested to return appropriate readers and writers corresponding to the identifier names. SuperElastix will return respectively an AnyReader or AnyWriter, which are non-templated Base Classes that, if updated, use the appropriate reader of writer internally (by use of polymorphism): `AnyWriter::Pointer my_writer; ... my_writer->SetInput(superElastixFilter->GetOutput(identifier))`. In this mode, it is required to set the Blueprint prior to request and connect readers or writers.

The following sequence diagrams show the order of function calls of each mode of operation.





Note that these are simplified diagrams and may not reflect all details and naming as found in the source code.

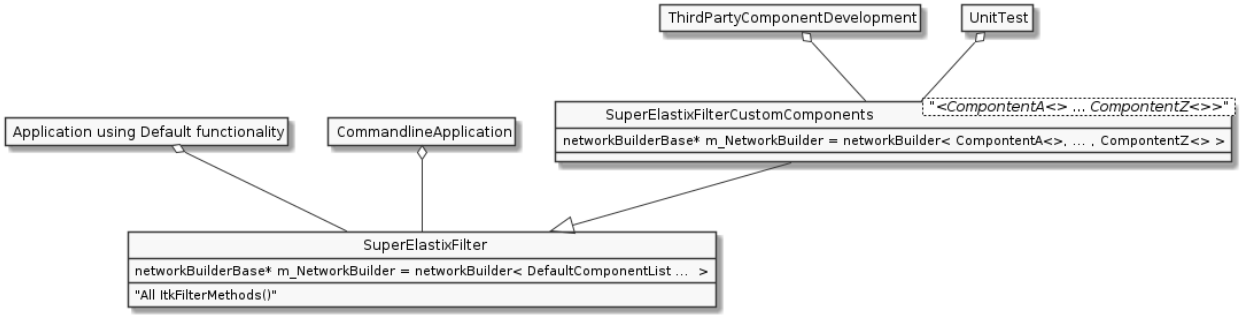
SuperElastixFilter component database manipulation

We provide two library interfaces, each supporting a different use case:

- *"Precompiled" SuperElastix ITK filter*, designed to be used in external applications, such as the commandline interface or company applications.
- *"Templated" SuperElastix ITK filter*, offering the most flexibility, useful for external third-party components and extreme use cases.

In both cases SuperElastixFilter has an internal database of components that can be used to dynamically construct the registration algorithm of choice. In the "Precompiled" library this database is populated with a predefined list of components (each with predefined template arguments, such as dimensionality and pixel type, etc). Predefinition of the components allows for hiding the implementation details of the components and speeds up the compilation process of the application (done via the Pimpl idiom). The "Precompiled" library is still and ITK filter and depends on the (templated) header files of the itk library.

In the "Templated" library the database of components can be populated by the user at compilation time by passing the component classes as template arguments. Applications using this library need access to all of SuperElastix internal source and header files at compilation time. This approach provides the flexibility to compile an instance of the SuperElastix ITK filter with, for instance, a sub- or superset of the default components, a set of components with exotic dimensionality or pixel types or even with third party components. Compiling the SuperElastix ITK filter with a small set of components is typically done in our Unit tests when testing a specific component or combination of components. Adding a third-party component to SuperElastix via template arguments does not require any modification of the source code files of the SuperElastixFilter. A third-party component can adhere to the existing already defined interfaces classes, but op top of that it can also define new interface classes.

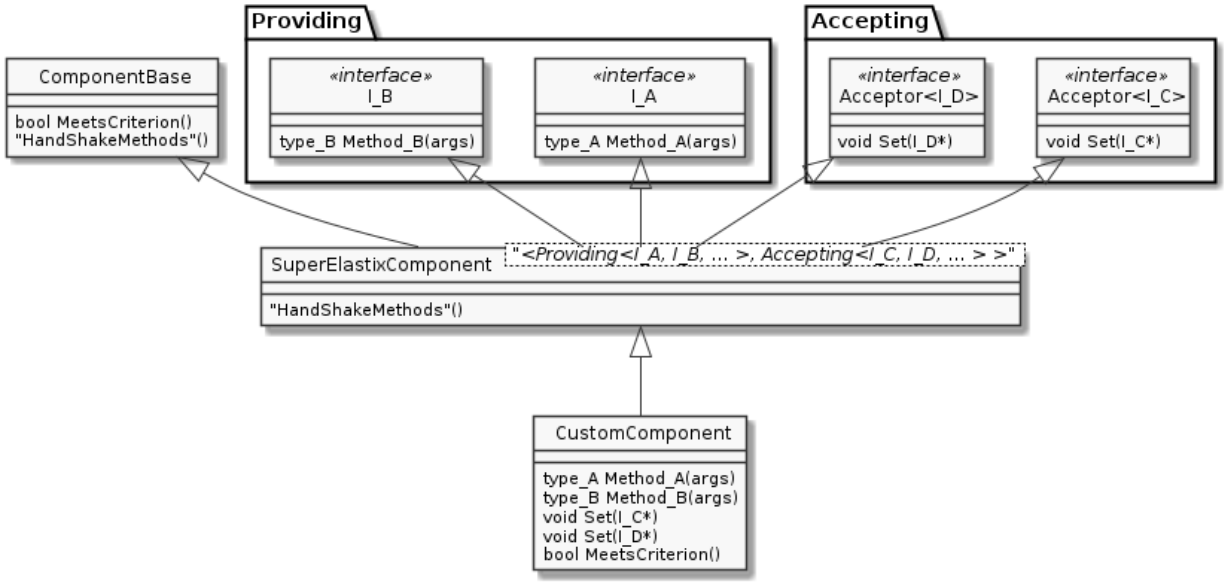


User Component Creation

A SuperElastix Component consists of accepting and providing interfaces. To let the handshake mechanism handle a component correctly the component (class) must adhere to the following structure. The component class must derive from the `SuperElastixComponent` class (solely). The `SuperElastixComponent` is a templated class with signature `< <Providing<I_A, I_B, ... >, Accepting<I_C, I_D, ... > > >`, with classes `Providing` and `Accepting` acting as placeholders to indicate the role of the interfaces `I`. By inheriting from the `SuperElastixComponent` class the component developer needs to provide the implementation for a number of methods. These are:

- All methods that have been defined in the providing interface classes that component developer selected.
- A `virtual void Set(I_x*)` for each interface class `I_x` that has been selected as accepting interface. (This example uses raw pointes, but in the reality we use code:*std::shared_ptr* for this).
- The `virtual bool MeetsCriterion( const CriterionType & criterion )`, which returns true if and only if the component has an implementation for which the criterion (read from the Blueprint)

holds or can be fulfilled.



Cmake module selection system

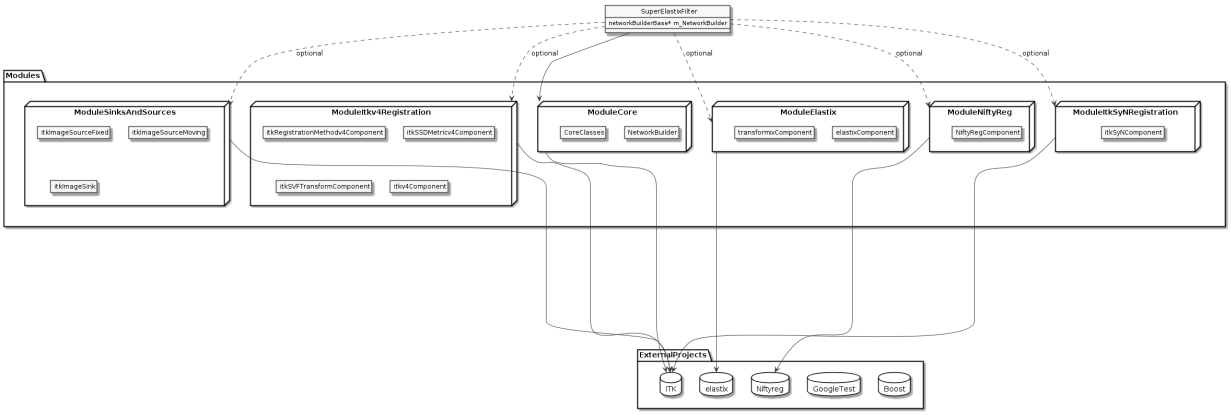
The modules can specify on which of the other modules they depend, and the build system make sure dependencies are enabled, and that they are enabled in the correct order. This means that users are always building the smallest possible binary, reducing binary size and compilation time. The following output shows the result of the default build, which builds the library interface along with elastix, nifityreg and the ITKv4 registration methods.

```
-- Found the following SuperElastix modules:
--   ModuleBlueprints
--   ModuleCommon
--   ModuleComponentInterface
--   ModuleController
--   ModuleElastix
--   ModuleExamples
--   ModuleItkSmoothingRecursiveGaussianImageFilter
--   ModuleNiftyreg
--   ModuleSinksAndSources
--   ModuleItkImageRegistrationMethodv4
--   ModuleItkSyNImageRegistrationMethod
--   ModuleConfigurationReader
--   ModuleFileIO
--   ModuleFilter
--   ModuleLogger
-- Enabling ModuleFilter requested by SuperElastix.
-- Enabling ModuleBlueprints requested by ModuleFilter.
-- ModuleBlueprints enabled.
-- Enabling ModuleController requested by ModuleFilter.
-- ModuleController enabled.
-- Enabling ModuleElastix requested by ModuleFilter.
-- ModuleElastix enabled.
-- Enabling ModuleExamples requested by ModuleFilter.
-- Enabling ModuleComponentInterface requested by ModuleExamples.
-- Enabling ModuleCommon requested by ModuleComponentInterface.
```

```
-- ModuleCommon enabled.
-- Enabling ModuleFileIO requested by ModuleComponentInterface.
-- ModuleFileIO enabled.
-- ModuleComponentInterface enabled.
-- ModuleExamples enabled.
-- Enabling ModuleItkImageRegistrationMethodv4 requested by ModuleFilter.
-- ModuleItkImageRegistrationMethodv4 enabled.
-- Enabling ModuleItkSmoothingRecursiveGaussianImageFilter requested by ModuleFilter.
-- ModuleItkSmoothingRecursiveGaussianImageFilter enabled.
-- Enabling ModuleSinksAndSources requested by ModuleFilter.
-- Enabling ModuleController requested by ModuleSinksAndSources.
-- ModuleController already enabled.
-- ModuleSinksAndSources enabled.
-- Enabling ModuleNiftyreg requested by ModuleFilter.
```

Modules are enabled once, even when requested multiple times, and can be turned off and on via CMake.

To add a module to SuperElastix, the developer creates a new directory and a CMake file that honor some naming conventions. The name of CMake file should Module[Name].cmake where [Name] is the name of the module. The CMake file contains a collection of CMake variables that the build system will use to integrate the module as component in the SuperElastixFilter. Users will never have to touch code outside module directory.



CHAPTER 4

Dashboard

Mac OS X

- Install docker

  :: $ https://github.com/boot2docker/osx-installer/releases

- Install pip

  :: $ sudo easy_install pip

- Install docker-compose

  :: $ sudo pip install -U docker-compose

- Build elastix with ELASTIX_BUILD_DASHBOARD set to ON.

- To fire up the dashboard server, run the following command in the Testing/Dashboard build directory.

  :: $ docker-compose up

  Go to http://elastix to see the dashboard.

- Download the logstash forwarder from https://download.elastic.co/logstash-forwarder/binaries/logstash-forwarder_darwin_amd64 and run the executable with the associated configuration file

  :: $ curl -O https://download.elastic.co/logstash-forwarder/binaries/logstash-forwarder_darwin_amd64 $ chmod +x logstash-forwarder_darwin_amd64 $ ./logstash-forwarder_darwin_amd64 -config elxLogstashForwarder.conf

- If you see "code:*DNS lookup failure "elastix": lookup elastix: no such host* add the elastix domain and associated boot2docker ip to the /etc/hosts file. For example, if `$ boot2docker ip` shows you `192.168.59.103`, you would the following line:

  :: 192.168.59.103 elastix

- We provide a key and certificate pair for testing purposes. DO NOT use this in production as they can be viewed by everybody (they are in public repository). To generate a new certificate, use the following command:

:: openssl req -subj "/CN=elastix/" -x509 -batch -nodes -newkey rsa:2048 -keyout elxLogstashForwarder.key -out elxLogstashForwarder.crt

Replace elastix with the domain name where you host the dashboard (e.g. elastix.isi.uu.nl). In production, point the ELASTIX_DASHBOARD_LOGSTASH_RSA_PRIVATE_KEY and ELASTIX_DASHBOARD_LOGSTASH_CERTIFICATE CMake variables to the new files in order for CMake to update the docker, logstash and logstash forwader configuration files with the new values.

Docker Cheat Sheet

- To stop containers run

  :: $ docker stop $(docker ps -aq)

- To delete containers run

  :: $ docker rm $(docker ps -aq)

- To delete images run

  :: $ docker rmi $(docker ps -q)

- Bash into docker container

  :: $ docker exec -i -t dashboard_elastix_1 bash

  where dashboard_elastix_1 is the name of the container (could be different on your machine). Persisted data is located in the `/data` directory and logs are located in the `/var/log/elasticsearch`, `/var/log/logstash` and `/var/log/kibana` directories.