Initial evaluation of the Art framework

Marco Corvo

CNRS and INFN

III SuperB Collaboration Meeting

March 20 2012

Outline

Outline

Outline

1. Why it could be the right candidate for a SuperB framework
2. Introduction to Art
3. Integration
4. Pros, Cons and open issues

Outline

1. Why it could be the right candidate for a SuperB framework
2. Introduction to Art
3. Integration
4. Pros, Cons and open issues

Sources

Sources

Most of the slides here have been inspired by Marc Paterno's talk at Fnal's Workshop on Concurrency in the many-cores era. Here's the workshop agenda

Where from and Who's for

- **Art** is a fork of CMSSW, the framework of CMS
- Art is being currently used by many different 'small' IF experiments
  - Noνa
  - mu2e
  - muon g-2
- It's intended for collaboration with a limited number of developers/maintenance people
  - much less complicated than 'big' LHC experiments frameworks

Where from and Who's for

- **Art** is a fork of CMSSW, the framework of CMS
- Art is being currently used by many different 'small' IF experiments
  - Noνa
  - mu2e
  - muon g-2
- It's intended for collaboration with a limited number of developers/maintenance people
  - much less complicated than 'big' LHC experiments frameworks

Where from and Who's for

- **Art** is a fork of CMSSW, the framework of CMS
- Art is being currently used by many different 'small' IF experiments
  - Noνa
  - mu2e
  - muon g-2
- It's intended for collaboration with a limited number of developers/maintenance people
  - much less complicated than 'big' LHC experiments frameworks

Nice features (for free)

1. It exists and we can use/test/exploit it
2. It has a community of competent developers who could help and give support (unpayable considering our cronic lack of manpower)
3. CMake based build system
   - The dark side is the distribution/setup one, based on **UPS**, a 'Fnal compliant' set of scripts/exe
4. The configuration of jobs is driven by a configuration file which doesn't need to invoke an interpreter (Tcl in our case)
   - Config files are parsed by a parser based on Boost Spirit and then translated into a ParameterSet
5. Art is moving to a concurrent programming model based on *Intel*® Threading Building Blocks

Nice features (for free)

1. It exists and we can use/test/exploit it
2. It has a community of competent developers who could help and give support (unpayable considering our cronic lack of manpower)
3. CMake based build system
   - The dark side is the distribution/setup one, based on **UPS**, a 'Fnal compliant' set of scripts/exe
4. The configuration of jobs is driven by a configuration file which doesn't need to invoke an interpreter (Tcl in our case)
   - Config files are parsed by a parser based on Boost Spirit and then translated into a ParameterSet
5. Art is moving to a concurrent programming model based on *Intel*® Threading Building Blocks

Nice features (for free)

1. It exists and we can use/test/exploit it
2. It has a community of competent developers who could help and give support (unpayable considering our cronic lack of manpower)
3. CMake based build system
   - The dark side is the distribution/setup one, based on **UPS**, a 'Fnal compliant' set of scripts/exe
4. The configuration of jobs is driven by a configuration file which doesn't need to invoke an interpreter (Tcl in our case)
   - Config files are parsed by a parser based on Boost Spirit and then translated into a ParameterSet
5. Art is moving to a concurrent programming model based on *Intel*® Threading Building Blocks

Nice features (for free)

1. It exists and we can use/test/exploit it
2. It has a community of competent developers who could help and give support (unpayable considering our cronic lack of manpower)
3. CMake based build system
   - The dark side is the distribution/setup one, based on **UPS**, a 'Fnal compliant' set of scripts/exe
4. The configuration of jobs is driven by a configuration file which doesn't need to invoke an interpreter (Tcl in our case)
   - Config files are parsed by a parser based on Boost Spirit and then translated into a ParameterSet
5. Art is moving to a concurrent programming model based on *Intel*® Threading Building Blocks

Nice features (for free)

1. It exists and we can use/test/exploit it
2. It has a community of competent developers who could help and give support (unpayable considering our cronic lack of manpower)
3. CMake based build system
   - The dark side is the distribution/setup one, based on **UPS**, a 'Fnal compliant' set of scripts/exe
4. The configuration of jobs is driven by a configuration file which doesn't need to invoke an interpreter (Tcl in our case)
   - Config files are parsed by a parser based on Boost Spirit and then translated into a ParameterSet
5. Art is moving to a concurrent programming model based on *Intel*® Threading Building Blocks

From CMSSW to Art

1. Removed some unnecessary heavy feature
   - `EventSetup`
   - XML based Framework Job Report
   - POOL file catalog support
   - other design features
2. Replaced the build system with a new one based on CMake
3. Replaced the distribution system with one based on tarball
4. Build with gcc 4.6.1 (part of the distribution) and Boost 1.49.0
5. Moving to the new standard `-std=c++0x`

From CMSSW to Art

1. Removed some unnecessary heavy feature
   - `EventSetup`
   - XML based Framework Job Report
   - POOL file catalog support
   - other design features
2. Replaced the build system with a new one based on CMake
3. Replaced the distribution system with one based on tarball
4. Build with gcc 4.6.1 (part of the distribution) and Boost 1.49.0
5. Moving to the new standard `-std=c++0x`

From CMSSW to Art

1. Removed some unnecessary heavy feature
   - `EventSetup`
   - XML based Framework Job Report
   - POOL file catalog support
   - other design features
2. Replaced the build system with a new one based on CMake
3. Replaced the distribution system with one based on tarball
4. Build with gcc 4.6.1 (part of the distribution) and Boost 1.49.0
5. Moving to the new standard `-std=c++0x`

From CMSSW to Art

1. Removed some unnecessary heavy feature
   - `EventSetup`
   - XML based Framework Job Report
   - POOL file catalog support
   - other design features
2. Replaced the build system with a new one based on CMake
3. Replaced the distribution system with one based on tarball
4. Build with gcc 4.6.1 (part of the distribution) and Boost 1.49.0
5. Moving to the new standard `-std=c++0x`

From CMSSW to Art

1. Removed some unnecessary heavy feature
   - `EventSetup`
   - XML based Framework Job Report
   - POOL file catalog support
   - other design features
2. Replaced the build system with a new one based on CMake
3. Replaced the distribution system with one based on tarball
4. Build with gcc 4.6.1 (part of the distribution) and Boost 1.49.0
5. Moving to the new standard `-std=c++0x`

From Art to SuberB

Disclaimer

Art is just an idea for the new SuperB framework. Next statements are to be read with 'in case we take Art' in front

- Art comes with a set of external dependent packages
  - CppUnit, Sigc++, GccXML but also ...
  - Root, CLHEP, Boost
- This means that our set of externals would increase
  - More work in terms of maintenance and portability. Can we afford it?
- Art itself can be used as an external dependency w.r.t. experiment software
  - Allows more flexibility as it could be plugged in and out of our software

From Art to SuberB

Disclaimer

Art is just an idea for the new SuperB framework. Next statements are to be read with 'in case we take Art' in front

- Art comes with a set of external dependent packages
  - CppUnit, Sigc++, GccXML but also ...
  - Root, CLHEP, Boost
- This means that our set of externals would increase
  - More work in terms of maintenance and portability. Can we afford it?
- Art itself can be used as an external dependency w.r.t. experiment software
  - Allows more flexibility as it could be plugged in and out of our software

From Art to SuberB

Disclaimer

Art is just an idea for the new SuperB framework. Next statements are to be read with 'in case we take Art' in front

- Art comes with a set of external dependent packages
  - CppUnit, Sigc++, GccXML but also ...
  - Root, CLHEP, Boost
- This means that our set of externals would increase
  - More work in terms of maintenance and portability. Can we afford it?
- Art itself can be used as an external dependency w.r.t. experiment software
  - Allows more flexibility as it could be plugged in and out of our software

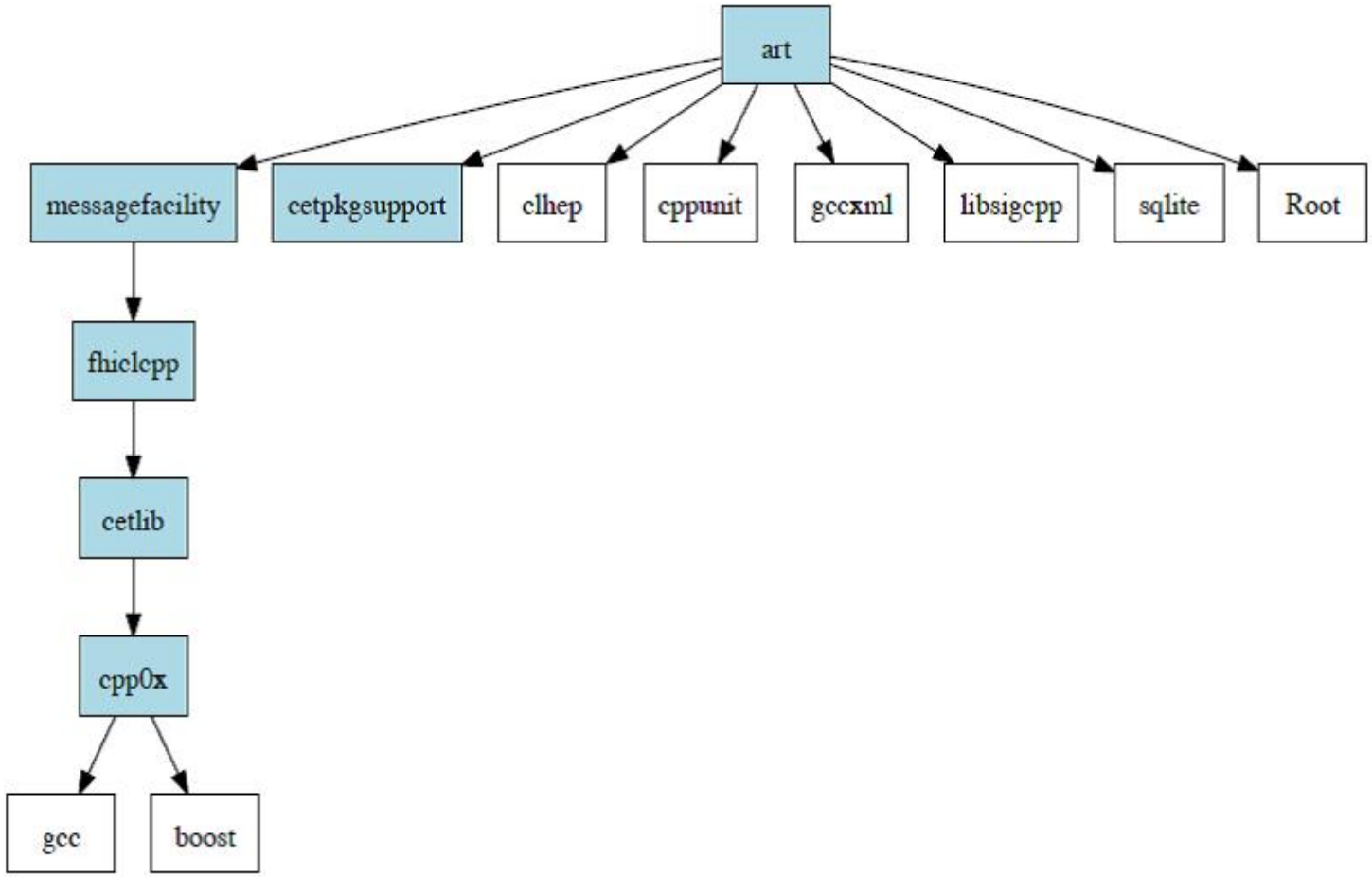
From Art to SuberB

Disclaimer

Art is just an idea for the new SuperB framework. Next statements are to be read with 'in case we take Art' in front

- Art comes with a set of external dependent packages
  - CppUnit, Sigc++, GccXML but also ...
  - Root, CLHEP, Boost
- This means that our set of externals would increase
  - More work in terms of maintenance and portability. Can we afford it?
- Art itself can be used as an external dependency w.r.t. experiment software
  - Allows more flexibility as it could be plugged in and out of our software

Art dependencies



Blue boxes show the set of packages composing the Art suite, that is packages developed directly by the Art team

From Art to SuperB II

- Art libraries are all built as shared objects
- There's a plugin system to load and (possibly) unload modules needed at runtime by the executable
- Opposite to our current system, where libraries are all static and executables carry all they need to run
- Which is the cost in terms of performance, on one side, and of additional code reorganization, on the other?
- Art has an executable (like **cmsRun**) which takes a config file to run
  - Different from, e.g., FastSim where there are many different executables

From Art to SuperB II

- Art libraries are all built as shared objects
- There's a plugin system to load and (possibly) unload modules needed at runtime by the executable
- Opposite to our current system, where libraries are all static and executables carry all they need to run
- Which is the cost in terms of performance, on one side, and of additional code reorganization, on the other?
- Art has an executable (like **cmsRun**) which takes a config file to run
  - Different from, e.g., FastSim where there are many different executables

From Art to SuperB II

- Art libraries are all built as shared objects
- There's a plugin system to load and (possibly) unload modules needed at runtime by the executable
- Opposite to our current system, where libraries are all static and executables carry all they need to run
- Which is the cost in terms of performance, on one side, and of additional code reorganization, on the other?
- Art has an executable (like **cmsRun**) which takes a config file to run
  - Different from, e.g., FastSim where there are many different executables

From Art to SuperB II

- Art libraries are all built as shared objects
- There's a plugin system to load and (possibly) unload modules needed at runtime by the executable
- Opposite to our current system, where libraries are all static and executables carry all they need to run
- Which is the cost in terms of performance, on one side, and of additional code reorganization, on the other?
- Art has an executable (like **cmsRun**) which takes a config file to run
  - Different from, e.g., FastSim where there are many different executables

From Art to SuperB II

- Art libraries are all built as shared objects
- There's a plugin system to load and (possibly) unload modules needed at runtime by the executable
- Opposite to our current system, where libraries are all static and executables carry all they need to run
- Which is the cost in terms of performance, on one side, and of additional code reorganization, on the other?
- Art has an executable (like **cmsRun**) which takes a config file to run
  - Different from, e.g., FastSim where there are many different executables

Issues

- In case we take Art as the new Framework, should we use it just as a baseline for future development, or should we use it also for existing software?
- The question is delicate as a (still to be quantified) amount of work has to be done to eventually adapt the existing software
- In general, moving to a concurrent programming paradigm has many implications
  - evident in a simple parallelization exercise where standard containers (`vector`) have to be changed with concurrent ones (`tbb::concurrent_vector`)
  - This is not always possible due to a different interface of `concurrent_vector` w.r.t. `std::vector` (e.g. the lack of the `erase()` method)

Issues

- In case we take Art as the new Framework, should we use it just as a baseline for future development, or should we use it also for existing software?
- The question is delicate as a (still to be quantified) amount of work has to be done to eventually adapt the existing software
- In general, moving to a concurrent programming paradigm has many implications
  - evident in a simple parallelization exercise where standard containers (`vector`) have to be changed with concurrent ones (`tbb::concurrent_vector`)
  - This is not always possible due to a different interface of `concurrent_vector` w.r.t. `std::vector` (e.g. the lack of the `erase()` method)

Issues

- In case we take Art as the new Framework, should we use it just as a baseline for future development, or should we use it also for existing software?
- The question is delicate as a (still to be quantified) amount of work has to be done to eventually adapt the existing software
- In general, moving to a concurrent programming paradigm has many implications
  - evident in a simple parallelization exercise where standard containers (`vector`) have to be changed with concurrent ones (`tbb::concurrent_vector`)
  - This is not always possible due to a different interface of `concurrent_vector` w.r.t. `std::vector` (e.g. the lack of the `erase()` method)

Issues

- In case we take Art as the new Framework, should we use it just as a baseline for future development, or should we use it also for existing software?
- The question is delicate as a (still to be quantified) amount of work has to be done to eventually adapt the existing software
- In general, moving to a concurrent programming paradigm has many implications
  - evident in a simple parallelization exercise where standard containers (`vector`) have to be changed with concurrent ones (`tbb::concurrent_vector`)
  - This is not always possible due to a different interface of `concurrent_vector` w.r.t. `std::vector` (e.g. the lack of the `erase()` method)

Issues II

- Also the ongoing discussion on our definition of Modules baseline would be affected
  - In our view, Modules have a (almost well defined)) 'require-provide' structure
    - Art has three types of Modules: EDProducer, EDAnalyzer and EDFilter
    - Producers can *modify* the state of an Event, Anlyzer can only *read* and Filter can only *decide* whether a Module should or should not run over an Event
- Need to understand whether this 'classification' fits the architecture of Art Modules
  - Or if we have to bend Art to our specifications

Issues II

- Also the ongoing discussion on our definition of Modules baseline would be affected
  - In our view, Modules have a (almost well defined)) 'require-provide' structure
  - Art has three types of Modules: EDProducer, EDAnalyzer and EDFilter
  - Producers can *modify* the state of an Event, Anlyzer can only *read* and Filter can only *decide* whether a Module should or should not run over an Event
- Need to understand whether this 'classification' fits the architecture of Art Modules
  - Or if we have to bend Art to our specifications

Issues II

- Also the ongoing discussion on our definition of Modules baseline would be affected
  - In our view, Modules have a (almost well defined)) 'require-provide' structure
  - Art has three types of Modules: EDProducer, EDAnalyzer and EDFilter
  - Producers can *modify* the state of an Event, Anlyzer can only *read* and Filter can only *decide* whether a Module should or should not run over an Event
- Need to understand whether this 'classification' fits the architecture of Art Modules
  - Or if we have to bend Art to our specifications

Conclusions

1. Art could be a good candidate as a baseline for SuperB Framework
2. It's lighter than a big experiment Framework
3. It's being used and supported by many different experiments
4. It's moving to TBB
5. But a deeper insight is needed to understand to which extent we need to modify/adapt our code

Conclusions

1. Art could be a good candidate as a baseline for SuperB Framework
2. It's lighter than a big experiment Framework
3. It's being used and supported by many different experiments
4. It's moving to TBB
5. But a deeper insight is needed to understand to which extent we need to modify/adapt our code

Conclusions

1. Art could be a good candidate as a baseline for SuperB Framework
2. It's lighter than a big experiment Framework
3. It's being used and supported by many different experiments
4. It's moving to TBB
5. But a deeper insight is needed to understand to which extent we need to modify/adapt our code

Conclusions

1. Art could be a good candidate as a baseline for SuperB Framework
2. It's lighter than a big experiment Framework
3. It's being used and supported by many different experiments
4. It's moving to TBB
5. But a deeper insight is needed to understand to which extent we need to modify/adapt our code

Conclusions

1. Art could be a good candidate as a baseline for SuperB Framework
2. It's lighter than a big experiment Framework
3. It's being used and supported by many different experiments
4. It's moving to TBB
5. But a deeper insight is needed to understand to which extent we need to modify/adapt our code