MXWARE OVERVIEW

FRAMEWORK DESIGN GOALS

Software framework for

- Simulation
- Analysis
- Reconstruction
- Data acquisition
- Slow control

Versatile and extensible system

- Complex configuration system for user-steering
- Plugin system for simple extensibility
- Simple and automated installation
- Developed and tested on Debian and OSX

Documentation

- All doxygen documented
- User and developer guides under writing in the wiki

SOFTWARE INSTALLATION

Debian repository

- Package name: mxware

Full rebuild

- Checkout MXInstall package from GIT
- Configure and install
- Cache initialization files available for each framework release)

Custom rebuild

- Checkout the single packages independently
- Make sure the installed libraries are in the LD path

Dependencies

- BOOST and CLHEP. (ROOT for optional component)
- On Debian all dependencies automatically managed.
- On OSX, BOOST and CLHEP to be manually installed

BUILD SYSTEM

Cmake build generator

- Cross-platform program to generate build systems.
- Tested on Linux and OSX to generate UNIX makefile build systems
- Able to generate XCode, Visual Studio, Eclipse and many more build projects

Several subpackages

- MXCore, MXIO, MXRandom, MXUitil, MXMCGens, MXSlowControl
- Many plugins already developed
- More to come when necessary
- Modular development, easier to test and understand

Test Units

- All classes tested through unit test routines.
- Test helpers for the test units of new implementations of a main component
- Every new class should have a test unit to be included in the core libraries

USE AND DEVELOPMENT MODELS

Simple user

- Modify the steering file and run
- Test beam operation, rerun of an analysis…

Plugin development

- Create a new plugin with additional code (analysis, simulation, reconstruction…). That's yours to do whatever you want with.
- Install the plugin in your home directory
- Template build with library and plugin in the repository

Core development

- Add new components and functionalities to the core system.
- Reviews plugin components and add them to the common library

CORE COMPONENTS

GENERAL FEATURES

Programming language

- Core written in C++11/14
- Actual C++, not C with classes
- Plugins could be written in other languages (not tried yet)

Core goals

- Common configuration system
- Define the extensibility system (plugin interface and class factories)
- Define the communication interface between modules (I/O system)

CLASS FACTORIES

Robust system to extend the code base

- Allows to add and use new implementations without touching the old ones
- No need to recompile anything but your own code

Define component interface

- What are the logical parts of the program
- How can communicate with them
- If already defined in another module/plugin skip this

Write some basic implementation

- A dummy/test version of the interface is the best starting point

Define a factory for that interface

- Use the existing template
- Load the new implementation in the factory

Choose the implementation to use

- Just a specification in the steering file
- Switch between implementations without recompiling anything

PLUGINS

What is a plug-in

- Shared library loaded dynamically at run-time
- Defined symbol list to access the available code
- A normal shared library is linked at compile-time

MXWare plugins

- Load new interface implementations
- Executes additional code

Plugin manager

- Core component that load available plugins and execute them
- Plugin execution customizable from the steering file

CONFIGURATION SYSTEM

Configuration tree

- The leaves of the tree contain the data attributes to modify
- The "type" attribute allows to choose run-time implementation
- The implementations can be loaded at runtime via the plugin system
- A GUI will be developed to simplify the modifications

Versatility

- Each subtree can be used independently from the parent
  - Great for testing purposes
- Special node to include an external steering file
  - Under development
  - Avoids copy and paste

Steering files

- Parsed to populate the configuration tree
- XML parser already available (BOOST component)
- The working configuration can be serialized back to a steering file
  - Save the process configuration for reruns or documentation

ADDITIONAL CORE FEATURES

Multithreading

- All is compatible with multi-threaded applications

Logging system

- Advanced multi-threaded logging system based on the BOOST log library
- Multiple sinks can be updated in parallel (for example log file and console)
- Details configurable in the steering file

Utility interfaces

- Define the basic interface for the most important concepts that a developed class may need (Named object, configurable object and so on)
- Default test templates for classes that follow the concepts defined by the interfaces

SIMPLE EXAMPLE

Logging system configuration

Tells where to find the plugins

Loading of particle masses

Custom program (analytic calculation of DP cross-section)
Defined in a user plugin

```
<magix>
        <!-- Enabling the logging system -->
        <logging>
                <logfile>./log/MXLog.txt</logfile>
                <console>true</console>
                <severity>info</severity>
        </logging>

        <!-- Configuring the plugin manager -->
        <pluginmanager>
                <path>@MAGIX_DIR@/@PLUGIN_INSTALL_DIR@</path>
                <path>@CMAKE_INSTALL_PREFIX@/@PLUGIN_INSTALL_DIR@</path>
                <printlist>true</printlist>
        </pluginmanager>

        <plugin pluginname="MXUtilityTools">
                <pdatatable>@MAGIX_DIR@/@DATA_INSTALL_DIR@/defaultPartData.mcd</pdatatable>
                <LogPDataTable>true</LogPDataTable>
        </plugin>

        <plugin pluginname="MXDarkPhoton_Plugin">
                <log_xsect>
                        <model type="DarkPhotonXSectCalculation" fastmode="false">
                                <q_squared_cutoff>0.0</q_squared_cutoff>
                        </model>
                        <particle>
                                <coord_type>SPHERICAL</coord_type>
                                <id></id>
                                <rho></rho>
                                <theta></theta>
                                <phi></phi>
                        </particle>
                </log_xsect>


        </plugin>
```

I/O MODEL

DATA STRUCTURE

IMXData

- Base class for all data objects

Collection

- Vector of data objects with a special iterator to simplify data access

Event

- Map of collections with additional metadata (Event number and timestamp)

Run

- Vector of events with additional metadata contained in a configuration tree
- Each run should define a stable experimental configuration

Runlist

- List of run with additional metadata stored in a configuration tree

DATA PROCESSING

Run List Manager

- Loads or creates the run to process and execute a set of run processors for each of them

Run Processors

- Loads events from a run or create new events and executes a set of event processors for each of them.
- Iterative execution possible

Event processors

- Contain the algorithm to analyze each single run
- It should load any of the existing collection and store the results in a new one

Creators

- Similar object for creating empty runlists, runs and events

SERIALIZATION

Configuration serialization

- Integrated in the configuration systems
- Works already

Data serialization

- Serialization of custom structures
- Serialization of object references
- Extensible to new structure

Possibilities

- ROOT serialization: already works but is not yet standardized in the package
- BOOST serialization: not yet developed. More generic and do not need the ROOT dependency
- They are not alternative to each other

EXAMPLES

Simulation output

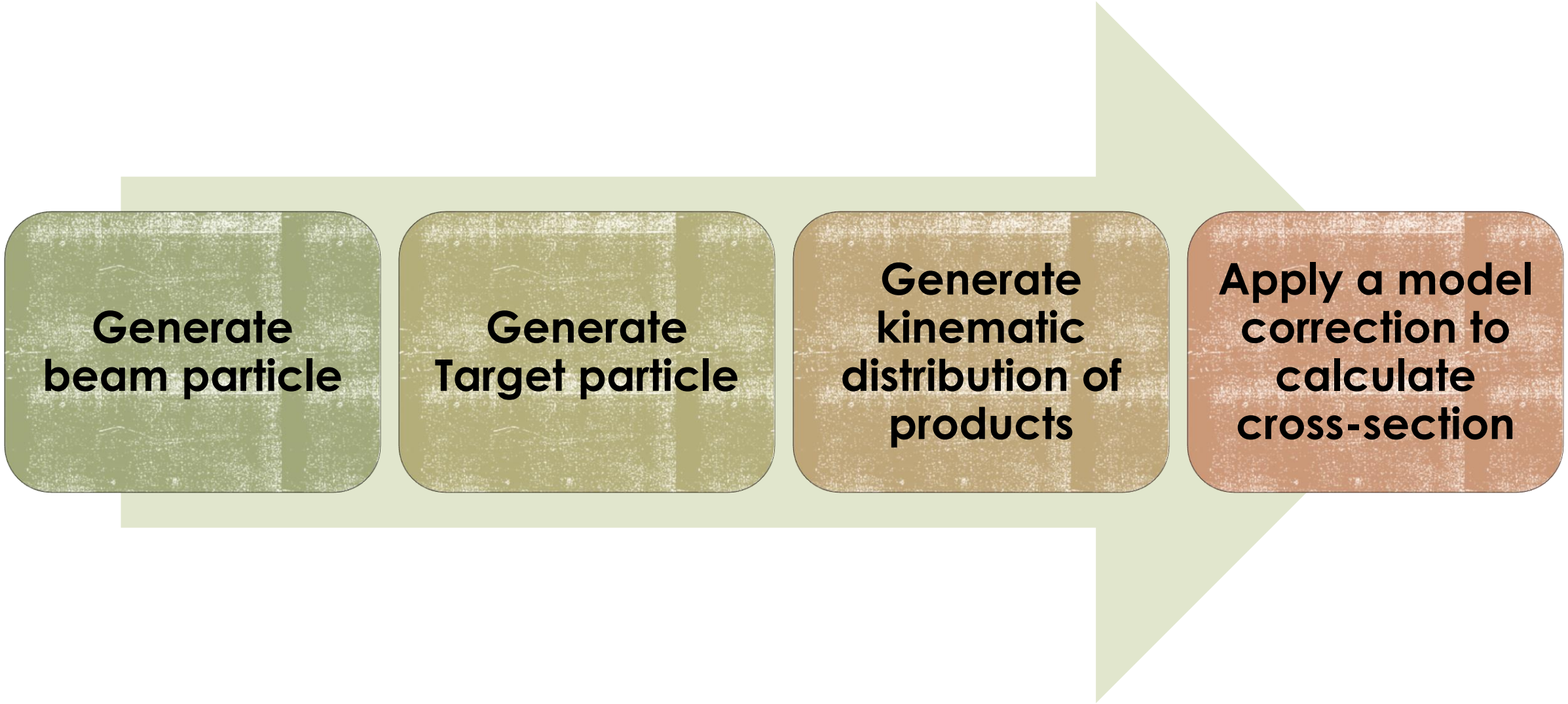
- Collection of MC Particles
- Collection of reconstructed particles from MC processing
- Each event has all collections
- Each run represents one experimental setting
- The run list represent the whole experiment
- Histograms produced at the run and runlist level

Testbeam output

- 1 collection for the raw input data
- 1 collection for the slow control data
- Several collections for the different reconstruction steps
- Each run is an experimental run
- Multiple runs analyzed together to make a scan (for example)

MC GENERATORS

PROCESS FLOW



BEAM GENERATOR

Describe the beam characteristics

- Nominal momentum
- Current
- Position
- Density distribution

Generate one MC Particle per call

```
<beam type="BeamIdeal" name="ElectronBeam">
        <nominalP>105.0</nominalP>
        <current>1.0</current>
        <offsetx>0.0</offsetx>
        <offsety>0.0</offsety>
        <helicity>0.0</helicity>
</beam>
```

TARGET GENERATOR

Describe the target characteristics

- Density distribution
- Material
- Polarization
- Size and position

Generates the target particle

- Beam particle as input
- Calculate the interaction position based on the input parameters

```
<target type="GasTargetUniform" name="Target">
        <centerz>0.0</centerz>
        <length>200.0</length>
        <spinx>0.390957</spinx>
        <spiny>0.0</spiny>
        <spinz>0.920409</spinz>
        <pressure> 3E-06 </pressure>
        <densityNTP>89.9</densityNTP>
        <pid>2212</pid>
</target>
```

DETECTOR DESCRIPTION

Defines a list of detector

- Number and type of detector is completely custom
- Acceptance tree loaded from an external file
- Each detector is represented by its acceptance
- Acceptance includes efficiency

```
<detector type="BasicSpectrometer" name="SpecR">
        <angledeg> 41.999 </angledeg>
        <offplanedeg> 0.0</offplanedeg>
        <centralP> -97.3355 </centralP>
        <distance> 700.0 </distance>
        <offsetz> 0.0 </offsetz>
        <acceptancefile>@MAGIX_DIR@/@DATA_INSTALL_DIR@/SpectrometerAcceptanceV1.dat</acceptancefile>
        <acceptV>false</acceptV>
</detector>
<detector type="BasicSpectrometer" name="SpecL">
        <angledeg> -66.9859 </angledeg>
        <offplanedeg> 0.0</offplanedeg>
        <centralP> 70.7609 </centralP>
        <distance> 700.0 </distance>
        <offsetz> 0.0 </offsetz>
        <acceptancefile>@MAGIX_DIR@/@DATA_INSTALL_DIR@/SpectrometerAcceptanceV1.dat</acceptancefile>
        <acceptV>false</acceptV>
</detector>
```

KINEMATICS GENERATOR & MODEL CORRECTION

Generates the product particles

- Calculates the kinematics distribution of the products
- Stores the output particle in the output collection
- Use the detector information to calculate only if there is a chance for detection

Model correction

- Calculates the cross section based of the specific kinematics configuration

```
<processor type="EventWeightingMCGenerator_U64" name="TestGen">
        <out_collection>MCPart</out_collection>
        <rndengine type="NR_SobolSequence" seed="0" dim="15" />
        <rndengine>
                <type>NR_HQGen</type>
                <seed>0</seed>
        </rndengine> -->
        <out_rate>false</out_rate>
        <kinematics type="DarkPhoton_BetheHeitler_KinGen" name="KinGen">
                <!-- <el_pcutoff>1.0</el_pcutoff> -->
                <el_p_min>0.0</el_p_min>
                <el_p_max>104.0</el_p_max>
                <el_costh_min>-1.0</el_costh_min>
                <el_costh_max>1.0</el_costh_max>

                <el_phi_min>0.0</el_phi_min>
                <el_phi_max>0.0</el_phi_max>     -->
                <dp_mass_min>10.0</dp_mass_min>
                <dp_mass_max>10.0</dp_mass_max>  -->
                <rec_costh_min>0.0</rec_costh_min>
                <rec_costh_max>1.0</rec_costh_max>
                <rec_phi_min>0.0</rec_phi_min>
                <rec_phi_max>0.0</rec_phi_max>  -->
                <leptPair_mass>105.0</leptPair_mass>
                <lept_costh_min>-1.0</lept_costh_min>
                <lept_costh_max>1.0</lept_costh_max>
                <lept_phi_min>0.0</lept_phi_min>
                <lept_phi_max>3.14159</lept_phi_max>     -->

        </kinematics>
        <model type="BremsstrahlPairCreationXSectCalculation" fastmode="false">
                <q_squared_cutoff>0.0</q_squared_cutoff>
        </model>
```

CONCLUSIONS

MORE COMPONENTS

Data Reconstruction	First implementation operational
Slow control system	Configured through MXWare core
	Data synchronized in the event structures
	Running with Epics (external software)
Full detector simulation	Parametrize detector efficiencies
	Interface our generators with GEANT or other similar software
DAQ	Online event generation and synchronization
	Yet to start
GUIs	Configuration tools should have an easy to use GUI

SUMMARY

Working system	The software framework works efficiently
	A few people already contributing to it
	Completely available through our git or debian repository
	Easy to install and extend
MC Generators	The first practical application of the system
	1 Bachelor thesis based on its use
	Dark Photon, Elastic scattering and S-Factor generators already working
Test beam data reconstruction	Two master thesis already done using it
	Need a bit of clean up to make it work smoothly
More development needed	More people working on it can help improve its functionalities

BACKUP

RANDOM NUMBERS

GENERATORS

Interface template

- Compatible with the BOOST URNG concept
- Can be used with all BOOST random distributions
- Templated according to the different numerical representations of the return type
- Adaptor to use all the random generator already present in the BOOST and STL libraries

Additional generators

- Pseudo-random generators from the Numerical Recipes 3rd ed.
- Sobol Sequence generator from the Numerical Recipes 3rd ed.

Configuration

- Custom factory to choose the generator at runtime with the configuration system

DISTRIBUTIONS

Using pseudo-random generators

- All the distributions implemented in the BOOST Random library can be directly used with any pseudo-random generator
- The distributions in the STL library are not compatible (but they are the same as those in BOOST)
- Sobol sequence cannot be reliably used with most distributions

Using quasi-random sequences

- N-dimensional distributions need n different sequences not n calls to the same sequence.
- All distributions implemented in available libraries call n-times are incompatible with the way a sobol sequence is implemented
- An implementation for uniform distribution is available.
- Need to find a better general solution

GEM DATA RECONSTRUCTION

DATA STRUCTURES

Raw data

- Whatever comes from the electronics

Pulses

- Processed data per channel
- Define charge and time of each signal

Hit

- 3-dimensional point in space

Track

- Particle trajectory in space
- Not yet defined in the code