AN OBJECT-ORIENTED FRAMEWORK FOR THE HADRONIC MONTE-CARLO EVENT GENERATORS

N. Amelin[1,2] *and M.Komogorov*[2,3]

[1] CERN/IT/ASD, Geneva,Switzerland
[2] JINR/LHE, Dubna, Russia
[3] Department of Physics, Jyväskylä University - Jyväskylä, Finland

Abstract

Basic features, code structure and working examples of the first object-oriented framework version for the hadronic MC event generators are shortly explained. It is argued that the development of the framework is in favor of taking into account hadronic model commonalities. The framework provides different possibilities to have user session more convenient and productive, e.g. an easy access and edition of any model parameter, substitution of model components by the alternative model components without changing the code, customized output, which offers neither full information about history of generated event or specific information about reaction final state etc. The framework based on the component approach opens a way to organize a library of the hadronic model components, which can be considered as the pool of hadronic model building blocks. It can indeed increase the productivity of a hadronic model developer, particularly, due to the formalization of hadronic model component structure and model component collaborations.

Keywords: hadronic,event generator,component,framework

Introduction. The object-oriented approach can be adopted to write the hadronic MC event generator codes. Such approach has many advantages as compared with traditional procedural coding [1]. Below we enumerate many observed commonalities for the hadronic MC models as well as for their usage and for their development. We advocate the development of an object-oriented framework [2] for such models. It allows us to separate commonalities from varieties in the hadronic model interfaces and application logics, since common means stable and can be developed only at once. We advocate also the component approach to build hadronic models. We assume that the framework will be useful for two categories: hadronic model users and hadronic model developers. By our definition of users they interact with the framework only by means of an user interface without writing and modifying of model codes. A developer to write its model code needs the knowledge of the framework architecture and libraries. As the framework name we suggest to use the *NiMax* word. The more extended description of our framework as well as several working examples illustrating user session and developer work can be found in the paper [3] and the framework manual.

Commonalities of the MC hadronic models. Even taking a fast look at the MC hadronic models one can see that they have much in common. The most of hadronic models are multi-component models and can include other models as additional or alternative model components.

Especially for application purposes an user needs the set of hadronic models to obtain proper description of the hadronic reactions [4]. Many hadronic models are complicated numerical models. For them it is always important to separate physical input, physical parameters etc influences on the simulation results from the numerical algorithm influences. It is often, when the same numerical algorithm is reused within models describing different physical phenomena. All hadronic models are phenomenological models having a large amount of model parameters and requiring

an additional information, e.g. the information about particle and nuclear static properties. Usually such information is needed in the read-only mode. The MC hadronic models act in similar way neither converting an input into an output or using an input to give an answer on the user request. The input and output are different characteristics of particles and nuclei. Acting so any MC hadronic model has deal with four vectors, e.g.energy-momentum and their transformations. Any hadronic model is required to produce different physical outputs, which should be analyzed. The above list of commonalities can be extended more. However, it is already clear that a hadronic model developer should take into account these commonalities by either common code structure or common used methods or common implementations etc.

Typical requirements from model user and developer. Any user of the hadronic models needs model descriptions. Such descriptions should be accompanied needed examples to demonstrate model usage. An user performing theoretical or experimental study of hadronic collision needs a possibility to ”play” with chosen model, e.g. the possibility to visualize and change model parameters, to configure model or model component, to choose an alternative model component, to customize hadronic model output. Another type of user, which needs the experimentally missing information about hadronic collision final states, is mostly interested in the generated event itself. The configured for a given type of hadronic reaction model should be offered for such user. The output information should be reduced until required minimum and presented in the required form. Model developers may want to rebuild an existing hadronic model or to incorporate an existing (”foreign”) hadronic model for a cooperative work with other existing models. They may also want to build a new hadronic model running standalone or in collaboration with other models. In the conventional approach for the developing of the hadronic MC models a developer or several developers are working independently on a particular model. Such approach, even if an object-oriented language is used, has many drawbacks. The main of them is that hadronic model commonalities as well as design and programming experiences are badly exploited.

Model components. The main idea of our framework is to adopt the component approach, when a complex hadronic model is composed from small and simple pieces, that are self-contained entities. The component approach allows us to formalize the particular type of components by separating interface part and application logic part. The component collaboration and their collaboration with an model user are provided by the interface parts. We offers different component frames (component wizards) with prefabricated interfaces for hadronic model developers. Thus, a model developer should mostly work on the component application logics and each component application logic can be developed independently. An application logic part can also be universal, because the same application logic can be used in several physical models. Model components can be composed in a variety of ways and the new components with their peculiarities can be added, that offer a flexibility for the construction of a powerful hadronic model. The different implementations of a component application logic can be interchanged at runtime enabling a hadronic model user to obtain the needed model properties without redesigning of a model. We distinguish different kinds of model components: algorithm component, data analysis component, table component etc. Each type of component have its specific properties and methods. To provide more flexibility for a developer, we also classify the components according their interfaces. The so-called ”runnable” algorithm components have execution and input methods. Such components can be used as the standalone models or as the main model components, which can include aggregated components.

Framework control sub-system. In the case of developing a composite hadronic model there are very important questions about the component control, their collaboration and their collaboration with an user. Component interface standards help us to suggest the dispatcher mechanism to answer on these questions. The dispatcher is hidden from an user. It controls all inner

framework processes at runtime. It is the chief of all components and creates needed files, obtains information about components, loads, runs and deletes any model component, checks a possibility to substitute a component by an alternative component etc. The dispatcher includes many methods to support several phases of the user-framework interaction: component visualization phase, component object creation phase, edition phase, execution phase and destruction phase. An user is able to overload default aggregated component set, default parameter set, default input map and default output configuration. The dispatcher helps us to organize multi-thread work of our framework and allows us an easy way to integrate our framework into the ”more global” package, e.g. the *GEANT4* [1].

Parameters and input maps. To support model parameter and model input handlings, we have developed the parameter-input set of classes. A component can contain only one group of parameters. Thus, model parameters are structured like tree and have different states. By model component registration an user can visualize and edit the default model parameters and the framework will provide parameter consistency check. Thus, a hadronic model component can be configured neither by its aggregated component substitutions or by its parameter edition without changing the code. For runnable algorithm components, we suggest input maps based on the lists of simple data types. By means of the input maps we organize not only the uniform input for a model user, but we help also model developers since our framework performs needed input map data converting. An user is able to edit them under framework control. An user can add its own input maps.

Data transfer class library. The objects of the data transfer classes are used for an information exchange between hadronic model algorithm components. Any object of the data transfer class supports serialization. Thus, framework output sub-system is based on the extendible data transfer class library. This library helps an algorithm component developer, because he or she does not need to worry about details of the input-output operations. The data transfer classes are tightly connected with the hadronic models domain. Their hierarchy represents physical objects (partons, hadrons, strings, nuclei etc) from this application area. The degree of universality of an algorithm component is determined by the amount of needed input-output information. Our data transfer classes have an hierarchical structure, the most universal algorithms are those, which use objects of the node data transfer class. Since this library fulfills very important functions for our framework, we are working on the navigation sub-system with the aim to offer transfer class wizard with aim to facilitate the library extension.

Framework database. The data transfer classes represent not only dynamical properties of the physical objects, e.g. momenta, positions etc, but also their static properties, e.g. charges, encodings. The static information is stored as tables. In our framework we have other tables, e.g. the interaction cross section table. We are working on the service tools (framework database service tools) to support such and other needed tables within our framework.

Framework output sub-system. Our framework is a tool to perform simulation of hadronic interactions. We have suggested an uniform and powerful output sub-system with main goal to facilitate generated data visualization and analysis. We write the configuration to be read at the beginning of an output file. This configuration is based on the list of basic types and it can be read within any other package. We have developed the framework input-output file system to avoid dependence from the different platforms. The output, which can be written on an output file, includes the output file header, an extra information (can be used by a developer for some reasons), the channel definition, the event configuration and the data part. A variable is stored in a channel with its own identifier. Each identifier of channel can have special prefix, if given channel is repeatedly stored. Any object of our framework has its unique identifier. The knowledge of object’s identifier helps us to obtain full information about this object. We have defined a program event

object as an unit of information has to be written. Each program event object has its own identifier, which helps us to find the configuration (array of channel's identifiers) in order to read the binary data from the output file. The event configurations are defined before run session. Thus, we able to protect any channel or group of channels from their output. There are also predefined event objects. They include the values of input map and parameters of model components, which are executed in the current run session. The output methods are writing the data part of an output file will write also the history count fields. They are used to make connection between program event objects and model component objects, which are responsible for the event object creations. Thus, in the case of a multi-component model to obtain the history of the generated event an user can activate any hadronic model component or all hadronic model components to write information about result of its work.

Visualization and analysis sub-system. We are developing the generated data analysis and visualization sub-system, which is based on the output sub-system. Now simple data filters and some plot facilities are provided for the users, whose use graphical user interface. The output sub-system is also able to prepare the output data to be suitable as the input for the external analysis and visualize tools.

Runtime information. There are three types of the runtime messages. The information messages are used to tell about normal execution process. The warning messages inform an user about the situations, which are able to destroy normal execution process. An user can control runtime information by enabling or disabling the information messages and the warning messages. The error messages appear, when the execution process will be aborted. The error message informs also about place and type of the error. The place of an error is detected by the framework itself and model developer does not need to worry about it. Each warning or error message has its unique identifier.

Framework help sub-system. The help sub-system based on the *HTML* is under development as well as the printed version of the framework manual. Having unique object identifiers we are able to bind these identifiers with the object describing *HTML* files. The help sub-system will be developed with aim to provide neither convenient access for help information or uniform way to add new documentation by means of component information frames. Such frames have been developed in complement of the component frames. They help a component developer to present information for the standard appearance.

Component library. Discussing different aspects of the hadronic model framework, we have paid attention mostly for its interfaces. But hadronic model component to be used for the reaction simulation should have its application logic. Developing the framework we assume that the implementation of this logic will be the main job of a hadronic model developer. The large number of the runnable algorithm components (see for model physics description [4]) as well as other components have already been implemented and included into the component library.

Utility class library. The utility classes contain useful data and methods: random number generators, different integrators, equation solvers, physical units and constants etc. Very convenient strategy to use physical units and constants was adopted from the *GEANT4*[1].

User interfaces. For communications with user we have developed command line and graphical (for the Windows platform) user interfaces. The user-framework collaboration within graphical user interface, where an user deals with several threads, is similar as in the case of command line user interface. In the first case it is going through parameter,input, output, runtime information etc graphical pads. In the second case an user operates with the corresponding ASCII files.

Acknowledgments. N. Amelin is grateful for S. Giani, who invited him to work on the hadronic models for the *GEANT4* toolkit. The framework idea was born during this work.

References

1 Wenaus T. *et al.*, GEANT4: An Object-Oriented Toolkit for Simulation in HEP, CERN/LHCC/97-40.

2 Taligent's Guide: Building Object-Oriented Frameworks, `http://www.ibm.com/java/education/oobuilding/index.html`

3 Amelin N. and Komogorov M., An Object-Oriented Framework for Hadronic MC Generators, accepted for publication in PEPAN Letters.

4 Amelin N., Physics and Algorithms of the Hadronic Monte-Carlo Event Generators. Notes for a developer., CERN/IT/99/6.