D11-2001-175

N. S. Amelin, M. E. Komogorov

COMPONENT-ORIENTED APPROACH TO THE DEVELOPMENT AND USE OF NUMERICAL MODELS IN HIGH ENERGY PHYSICS

Contents

Chapter 1

Introduction

1.1 Motivation

The full descriptions of relativistic hadronic and nuclear interactions from the first principles of quantum chromodynamics (QCD) are very limited. As a rule we are able to obtain the QCD predictions for the short distance processes, when interaction takes place with large momentum transfer. However, the processes with the relatively small transferred momenta play a dominant role in hadronic and nuclear interactions. It is very important task to develop and use the QCD motivated models for the predictions, analysis and description of these soft processes. Besides, modern experiments related to the high-energy physics are very complicated and very expensive. The detectors design, their construction and their performance require careful numerical simulations. This makes Monte Carlo hadronic models to be extremely popular. As phenomenological models they have many parameters that are fixed by comparison with experimental data. After parameter determination one can apply the Monte Carlo model to predict the interaction results.

Thus, the MC hadronic models can be used as the hadronic event generators with the main goal to study hadronic collision phenomena as well as the source of information about hadronic collision final states with the aim to utilize this information [1].

The MC hadronic models are complicated physical models. A typical MC event generator deals with numerous physical processes. They are based on very complicated numerical algorithms. Thus, the essential efforts should be done to formulate, program and test such a model.

There are also different problems faced with the model users in their every day work on processing experimental data or preparing new experiment. The most of existing model codes are very difficult in management and commonly

used only by their authors. Even for theorists with good understanding of the hadronic models, it is not easy to apply foreign codes to solve their research problems.

An object-oriented approach based on the C++ can be adopted to write the hadronic MC event generators codes. Such approach has many advantages as compared with traditional procedural coding (see, e.g., [2]). The most important of them are the various facilities of the C++ language to define new data types along with the operations that are used to manipulate the types. These are *native* abstractions (e.g., particle 4-momentum) for MC event generators domain and we can use them with the flexibility of the built-in data types. We advocate the work on an object-oriented framework [3], since we have observed many commonalities for the hadronic MC models as well as for their usage and for their development. The framework approach is more justified, when the list of models that chosen for development is very large and potentially can be increased.

A typical framework [3] can be considered as well-documented thematic collection of software to build related applications. It outlines the main architecture for the application to be developed. The successful framework should not only support needed features and provide default implementation and built-in functionality as much as possible, but it should also allow an easy modification and extension of the built-in functionality. A goal of any framework is the reusability. The software developer should be able to reuse written code, e.g., classes from the framework libraries, and the design of a framework. A framework design is closely connected with the design patterns are used to document certain elements of it. A design pattern is the concise definition of a technique that demonstrates some successful solution for particular coding problem. Particularly, the factories and proxies design patterns (see book [4]) have been applied in our system version.

In this manual we advocate a component approach to the development, assembling and use of numerical models in high-energy physics. The component can represent a model of either single physical process, e.g., hadron elastic scattering, or very complicated physical phenomenon, e.g., ultra-relativistic heavy-ion collision.

Here we describe a software system that is designed and implemented to support this approach as well as an application of this system for Monte Carlo hadronic event generators. We refer it as the NiMax system. The central system part is an object-oriented framework. We have in mind that such system can be a base to build a library of the model components having different numerical algorithms. It can allow us to extract model components from the library pool and compose them into powerful physical models.

We assume that NiMax system will be useful for two categories: the

numerical model users and numerical model developers (advanced users). We consider a model user as a person who interacts with the system by means of a user interface without writing and modifying of the model codes. A model developer is assumed to work with the system on the level of internal system interfaces. A developer needs knowledge of the system concepts, system structure and libraries as well as knowledge of C++ language.

In this manual we provide numerous examples of the system user session. We describe also physics and numerical algorithms of the implemented model components that are included into hadronic modules. These components are developed to perform MC simulations of hadronic and nuclear collisions at high energies. We would note that this list of components does not exhaust a list of all developed components and components that are under development. Of course, the NiMax system could be applied to find solutions of many other tasks, where the development and use of complicated numerical models is required. The chosen list of the MC hadronic components is connected with professional activity of the system authors. In our opinion, it is complete enough to demonstrate the most of system features and may help a reader and potential system user to obtain right impression about the system goals.

1.2 Content of this manual

The manual consists of an introduction, three chapters, conclusion and appendix chapter.

In the introduction we stress the importance of problem, explain the subject of work, then we formulate main goals of the work and shortly describe the structure of the manual.

In the first chapter we discuss commonalities of MC hadronic event generators, formulate user and developer requirements for a component-oriented software system. Here we present and shortly discuss basic ideas of the component-oriented NiMax software system.

In the second chapter we present more detailed description of the NiMax system and explain its functionality. At first we give architectural overview of the main parts of the system and then discuss each part in separate. We discuss the component concept. We describe the component interfaces and their views. We explain the packaging of components and related software into application domain modules. We consider the development process of an application module and component. In particular, we consider application data type classes and some tools for the component development. Here, we discuss the component documentation. Then we explain the NiMax data model concept. We formulate a data event. Then we describe a data file and

its views, library of the data transfer classes as well as pre-defined events and channels. We provide some details of a component collaboration and explain the data matching mechanism. Here we formulate the concept of event-oriented component nets and describe net's view. At the end of this chapter we present short description of the framework control and navigation methods.

In the third chapter we describe a user session in more details. Here we explain how to select a particular component from the list of available components. We describe the component edition, i.e., an edition of their inputs, parameters and sub-component substitution. We describe how to reconfigure a component output. Then we discuss how to create a component net from several components and edit an existing component net. We explain the component net execution control. In this chapter we discuss in details how to work on a data file, e.g., how to select data. We explain histogram and plot facilities of our system. At the end of the chapter, we discuss how to obtain needed help information.

In the conclusion we give the system and implemented component summaries stressing their scientific importance and novelty.

In the appendix chapter we explain physics and numerical algorithms of the several implemented model components that are included into developed hadronic modules. Some of the implemented components are sub-components of the implemented composite components. We avoid repeating description of the sub-component physics and algorithms. We would like to inform an interested reader that physics and numerical algorithms of the implemented hadronic model components can also be found in the report [1].

Bibliography

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

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

[3] Taligent's Guide: Building Object-Oriented Frameworks, *http* : *//www.ibm.com/java/education/oobuilding/index.html*.

[4] Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns. Elements of Reusable Object-Oriented Software, Eddison-Wesley, 1995.

Chapter 2

System requirements

2.1 Commonalities of the numerical models

Even taking a fast look at the MC hadronic models one can see that they have much in common [1]. First of all they are phenomenological models having large amount of model parameters. We can specify the parameters as the physical parameters (hadronic model tuning constants) and hadronic model configurators. The first type of the parameters gives a possibility to change hadronic model results. They operate similarly as a physical input. This type of parameters fulfils very important job to store physical information about hadronic processes. The second type of parameters also offers a possibility to change hadronic model results, but by the changing of an application logic of a numerical algorithm.

Additionally to the parameters much more information should be provided for some hadronic models. For example, the information about physical properties of particles: quark contents, electric charges, masses, decay branching, etc, is required. The information about physical properties of stable and excited nuclei: binding energies, spins, level density parameters, fission barrier heights, etc, is also required. Usually such information is needed in the read-only mode.

The MC hadronic models act in a similar way. They convert an input data into an output data answering on the user requests. An input can be the characteristics of particles (hadrons, partons, gammas, etc) or characteristics of nuclei (stable nuclei, excited fragments, etc). An output can be again the characteristics of particles or characteristics of nuclei. Acting so any MC hadronic model deals with four vectors (energy-momentum, time-position, etc) and their transformations. Any hadronic model somehow handles the n-body kinematics.

The most of hadronic models are multi-component models. A multi-component model includes other models as additional or alternative model components and has a complicated execution flow. Especially for the application purposes a user needs a set of hadronic models to obtain proper description of the hadronic reaction final states [1].

Practically, all hadronic models are complicated numerical models. For them it is not always trivial to separate "physics" from "algorithm", i.e., to separate physical input, physical parameters, etc influence on the results of a simulation from a chosen numerical algorithm influence. It is also often, when the same numerical algorithm can be reused within several models describing of different physical phenomena, e. g., the decay of resonances and excited nuclei according to the relativistic phase space, the elastic scattering of partons, hadrons and nuclei, the search collision and decay algorithm for hadron transport model and parton transport model, etc.

Different kind of errors may be occurred during the initialization of a hadronic model or model runtime. The source of errors may be due to the inconsistent user input as well as due to the complexity of numerical algorithms. The last situation is often unexpected situation.

Any hadronic model is required to produce different physical outputs, which should be analyzed. An output can be only specific information about hadronic reaction final states or complete information about the history of a generated event.

The above list of commonalities can be extended more. For example, besides of the kinematics all hadronic MC models deal with random sampling of different variables according to the different probability distributions, i. e., a large set of the random number generators is required. Different mathematical utilities: equation solvers, integrators, interpolators, etc, are needed to perform numerical operations. However, it is already clear that the hadronic model developers should take into account these commonalities by either common code structure or common used methods or common implementations, etc.

2.2 Typical requirements from model users

The different usage strategies lead to different user requirements for a hadronic model package. A user performing theoretical or experimental study of hadronic collision needs a possibility to *play* with a chosen model, e.g., the possibility to visualize and change model parameters, configure the model or a model component, choose an alternative model component, customize the model output. Thus, a mechanism to check consistency of the user al-

terations should be provided. The run control requires runtime information and an exception handling mechanism for this type of users.

Another type of users (applied users) is mostly interested in a generated physical event itself. The model that is configured for a given type of hadronic reactions with default values of parameters should be offered for the applied user. The output information should be reduced until required minimum and presented in the required form.

Of course, the both types of users need to have much more, e.g., a simple and self-explanatory mechanism for hadronic reaction input preventing from errors due to the inconsistent input, analysis and visualization tools are also required to analyze a generated output, etc. Thus, for any kind of users their user sessions should be convenient and productive.

2.3 Typical requirements from model developers

Model developers may want to rebuild an existing hadronic model with the goals to extend a range of its applicability or to increase its predictive power. They may want to incorporate an existing hadronic model for a cooperative work with other existing models. Also they may want to build a new hadronic model running standalone or in collaboration with other models.

The enumerated situations are primer tasks of a hadronic model developer that are directly connected with the improvements and extensions of numerical algorithms. But in reality he or she should do much more. For example, to satisfy user requirements a developer should realize a user interface. The interfaces between created hadronic model code and the outside world are also needed as well as different adapters, if one wants to use external packages. A developer should have in mind a possibility for a hadronic package to work within another package. A hadronic model developer should facilitate the tasks of future developers as well.

In the conventional approach (see, e.g., [2]) for the developing of the hadronic MC models a developer or several developers are working independently on a particular model. Such approach, if even an object-oriented language is used, has several drawbacks. First of all, the hadronic model commonalities as well as designing and programming experiences are badly exploited. For example, each new developer has usually started to develop a particular hadronic model from scratch. As result of it each new model developer starts from analysis and design stages. Design duplications are manpower consuming. The different designs have different qualities and they

provide different degrees of satisfaction for the user as well as for the future model developer requirements. The design duplications lead also to duplications in code implementations. The quality of a particular hadronic model code depends strongly from the coding experience of a model developer or developers. It is also becomes more difficult to learn, maintain and extend a set of hadronic models created by different developers as well as to connect them for collaborative work. As a rule the hadronic model developers are not software experts, they are physicists and experts in their subject domains. For them it may be difficult to find a proper solution of the specific software tasks.

Thus, as result of the analysis of MC generators and analysis of different user and developer requirements it was decided to develop of an object-oriented framework to take into account the hadronic model commonalities, to facilitate user work and to increase the productivity of a developer work. In the frame of a framework the hadronic model users and developers can be more deeply concentrated on the solution of application domain problems. The developer needs to write much less code since an essential part of the program already exists. He or she does not need to be a software expert to write a robust code. A new model code inherited from the framework could also be much easily tested since it is already integrated with the rest of a framework.

2.4 Basic ideas of the system

As it was already discussed the hadronic models as well as their usage and their development have a lot of commonalities. During the system development we try to separate the observed commonalities from variabilities in hadronic model interfaces and their numerical algorithms since common means stable and can be developed only at once.

The central concept of our approach is the concept of a component. We consider the model components as unit blocks to construct a composite numerical model [3], [4]. All such blocks can be stored as an extensible collection of the model components.

Any model component can be structured into an interface part and the part presenting its numerical algorithm. The interface part of a model component allows component interactions with the outside world. By means of this part a user can also handle model component parameters and its input-output. We try to formalize the component concept. Such formalization becomes visible, when we provide component interface standards. On the other hand the interface standards, if they are required, dictate a model de-

veloper to follow definite rules during the component implementation. The interface standards facilitate the developing of model component interface part, if these rules are taken into account by means of component generation tools. Similar tools we also provide to help the writing of component documentation.

The component developer should mostly work on the implementation of a particular application algorithm. To facilitate programming of application algorithms for the MC event generators we have developed a library of classes representing physical quantities and many operations on them from a given application domain. We refer these classes as the application data type classes. To simplify programming of component input and output we have created a library of the data transfer classes.

Object-oriented programmers often use the inheritance mechanism to modify existing software with minimal efforts. In our case any component is extensible by the inheritance with adding new interface or algorithm functions.

Besides of the inheritance mechanism we need another mechanism allowing a component developer to use the codes (interfaces and algorithms) of already implemented components. Such possibility should not create any problem for the component developer to apply C++ programming technique during the implementation of a complicated numerical model or a set of numerical models. Such possibility should not create any problem for the component user, e.g., for the access to a sub-component or substitution of a sub-component. We offer such mechanism that is referred as the component aggregation.

A component developer can share some data, functions and classes among several components that belong to a particular application domain or component category. We have suggested an application module idea for the packaging of components and component related software. Such module (after its compilation) is loaded in memory as a dynamically linked library. The application modules fulfil twofold function they offer useful software for component developers and hide it from component users.

Either a component or several components can be loaded in memory and each component is executed as a separate process. A user has a possibility to control execution processes, particularly, due to the different runtime information messages.

Thus, we have defined a component model or formulated a standard component [3], [4], [5] and suggested several mechanisms of its development and offered different ways of its management. We have developed a set of classes to support the component development and component management.

Any component deals with data. It may generate very large bulk of the

data and store it in a disc. It may read the data from a disc and process it. The collaborative work of several components requires a data exchange. To fulfil these needs we have introduced an idea of the data event [3], [4], [6]. It is a tree-structured portion of data that consists of only values of the basic data types. Any data event has its definition that describes event configuration and can be placed directly into memory or stored in a disc. To support data streams we have offered a data file and the component may write and read data in this file. We have also suggested a mechanism to control virtual data streams. We have suggested a component net concept that is based on the virtual data streams. The component nets are collections of different components that are collaborated through their standard interfaces by sending and receiving data event-messages and connected in a sequence to process numerical data.

Thus, we have defined the data model [6], [7] and developed a set of classes, to support data management. The system user has obtained possibilities to configure a component for reliable input and output, navigate through stored data, select and visualize data and assemble several components into a component net.

We try to build a loose system, e.g., by decoupling of objects from their view objects, by decoupling of different component development, by decoupling of the file and help sub-systems (see the next chapter).

Bibliography

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

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

[3] Amelin N. and Komogorov M., An Object-Oriented Framework for the Hadronic Monte-Carlo Event Generators. JINR Rapid Communications, 1999, No. 5-6 [97]-99, p. 52-84.

[4] Amelin N. and Komogorov M., An Object-Oriented Framework for the Hadronic Monte-Carlo Event Generators. In Proc. of Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP2000), 7 - 11 February 2000, Padova, Italy, p. 119-123.

[5] Komogorov M. and Amelin N., Component-oriented framework for hadron Monte-Carlo event generators. The abstract published in Proc. of XXXIV annual conference of the Finish Physical Society, March 9-11, 2000, Espoo, Finland, p. 91.

[6] Amelin N. and Komogorov M., NIMAX: A New Approach to Develop Hadronic Event Generators in HEP. PHYS. P&N LETTERS, 2000, No. 3 [100]-2000, p. 35-47.

[7] Amelin N. and Komogorov M., NIMAX System: A New Approach to Develop, Assemble and Use Numerical Models in HEP. The talk presented at the XV Int. Seminar on High Energy Physics Problems, Dubna, Russia, 25 - 29 September, 2000; JINR Preprint, E1-2001-31, p. 1-14.

Chapter 3

System concepts

3.1 NiMax architectural overview

In this chapter we are going to discuss several important parts of the NiMax system that are shown in Fig. 3.1 and their interactions as well.

At the beginning we would like to describe the component concept from the software design point of view. Here we will explain different component properties that facilitate component management and development.

We will explain the packaging of components that are related to a par-

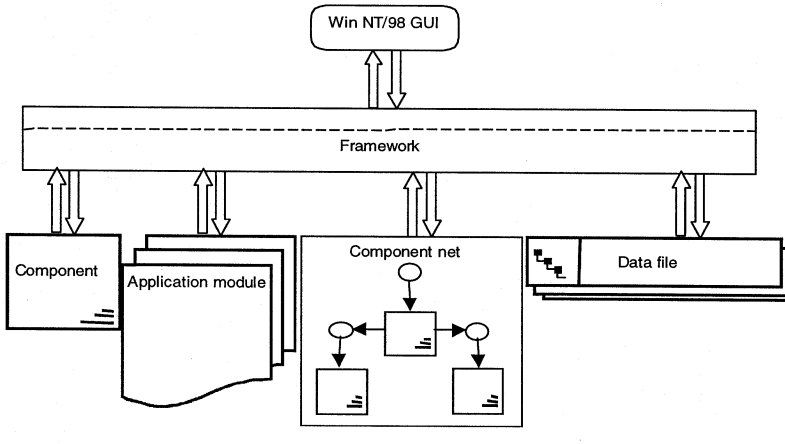

Figure 3.1: NiMax system: an architectural overview.

ticular application domain and some software that are shared by these components into application modules. From the developers point of view these modules are parts of component projects that include also the module and component development tools. For system users the modules are prepared as dynamically linked libraries (DLLs).

Then we would like to pay attention for the data management. Particularly, we will explain a system data file. In spite of the data file importance it is only a part of more extended concept that is referred as the data model concept. The key point of this concept is the data event.

We will explain the idea of component nets and discuss some details of the net assembling and execution as well as net peculiarities in the comparison with composite component development.

At the end of this chapter we will give short description of the graphical user interface (GUI). In the next chapter, where the system user sessions are described, we will provide many more details about the GUI.

We have developed a large set of classes to support component management and development, data management and visualization. This set of classes is thought as a framework. Its application programming interface (API), i.e., public or protected methods (open arrows in Fig. 3.1), links together the components, application modules, data files, component nets and GUI.

The NiMax system is written in pure C++ and only the GUI is based on the Microsoft foundation classes (MFC) library [1]. Only the GUI is an operational system-dependent part. Now our system with the GUI is working on different Windows platforms. However, the NiMax system is portable (on the level of source code), if it is applied in the command line mode.

We would note that we apply a document - view technology (see [1]) within our system. The main idea of this technology is a separation of document objects that hold data from the objects that display data and allow editing. The views show different facets (or several the same facets) of the same data and, e.g., if a user edits an active view, then all other views must be updated. Each document (component, component net, etc) is connected with a file having unique extension. This technology allows a development of different visualization and edition systems without changing of a data control system.

The recent version of the NiMax system includes four types of documents. The first type is referred as a workspace. The main information that is kept in the workspace document is a list of registered components. The second type is a component net file. It holds information about component states (parameter values, data output configurations, description of component relations, etc). The third type is a data file. The fourth type is a file for the storage of graphical information.

The document - view technology allows independent extensions of numbers of document's types and their views. For example, we have developed a help system that uses html documents. These documents can be browsed and edited by some standard tools that belong to an operational system. Our help system could be used separately from the NiMax system for advertisement and for studying of the implemented model components.

3.2 NiMax component

In this section we are going to discuss the component interfaces and their views. Then we explain different properties of the components. The component packaging is an important issue for component development and management will also be described. Finally, we shall discuss the component documentation.

3.2.1 Component interfaces

We can consider any component as a set of standardized interfaces. By means of the interfaces the component client (user, framework or another component) can talk with the component asking a definite service. Only through interfaces a component communicates with the outside world. An interface includes several methods and some related data. Let us explain functionality of the standardized NiMax component interfaces that are presented in Figs. 3.2 - 3.3.

By means of *the input interface* a user sends a request for a component and provides necessary input data to fulfill this request. The request as well

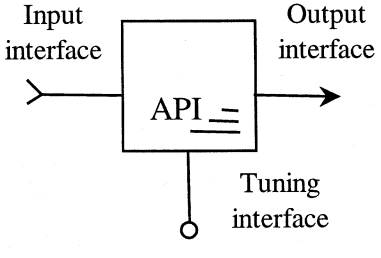

Figure 3.2: Main component interfaces.

as input data is provided in a form of *an input map* [2], [3]. The input map is a list of simple data types and has linear structure of data. We can refer input maps as user-friendly linearly structured data. If the component may start to run from a request obtained by its input interface the system considers such component as a main component.

The tuning interface gives a possibility to tune a component with aim to obtain reliable result from its execution. By means of this interface a user has access for model parameters and switches. The framework includes a set of classes to support the parameter and input map management [2]. We would note that the input interface and tuning interface have many similarities from the developer's point of view (the same classes in use, the same data structures in use). An input map data can be understood as a list of mandatory parameters that must be specified by users before the component execution.

The result of a component execution (output data) is obtained by means of *the output interface.* This interface assists either to write the component output data in a data file or to send the component output data for other components. The output data are produced as configured data events and have a tree structure [4], [5] in the common case (see the next section for more details).

A component can also read its input from a data file or receive it from other components. In these cases it starts to run from a request obtained by *the matching interface.* The matching interface allows for the component to select data from an input data flow according to a matching configuration [4], [5]. A particular matching configuration is realized as *a matching map.* The matching maps are different from the input maps. They do not include data and include only data configurations (see the next section for more details).

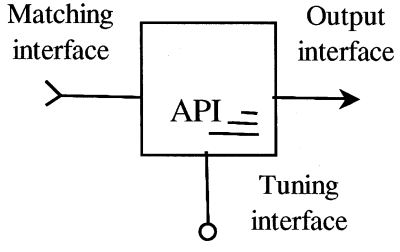

Figure 3.3: Component interfaces.

The component user is not able to change these configurations, but he or she can enable or disable a particular matching map for a chosen component. We can refer the matching maps as developer-defined data configurations.

We have also developed *a runtime information interface* (it is not shown in Figs. 3.2 - 3.3). It is used to obtain and control component runtime messages (see the NiMax framework methods section for more details).

Besides of the outlined standard interfaces each particular component has its API, i.e., a set of public and protected methods, which implements the component functionality and can be called directly or indirectly by means of the component standard interface methods.

We can classify different types of the components according to a presence or absence of a particular interface or interfaces. For example, we distinguish runnable components that have input interfaces or matching interfaces from general components that have neither input nor matching interfaces and cannot be executed, but they are useful as sub-components.

The system allows an extension of existing interfaces as well as adding of new ones. We can imagine a component that needs an external data interface to receive the data having external format. For example, a component can need an additional standard interface to manage data from a database. Thus, by adding more standard component interfaces we can extend an application area of the NiMax system and facilitate the work of a component developer in this application area.

3.2.2 Component interface views

Each component interface is complemented by at least one view as it is shown in Fig. 3.4. To obtain the component interface views the component objects should be created (see below). These views allow a visualization of the component input maps, component matching maps, component parameters, its output configuration, a structure of a composite component as well as component runtime information. These views help the component user to perform many actions on the component state (see the next chapter). A component interface can be connected with several views to realize different ways of its data display and edition, but each view object is related to only one interface. It is important to stress that changing of a view or adding of a new view does not require a changing of its interface. It allows an independent (from rest of the system) improvement and extension of the graphical user interface. Besides the component interface views there are more views that pertain a component, e.g., the component documentation view that helps a user to learn this component.

3.2.3 Component properties

Besides of the standardized interfaces and views, the system components have other common elements (see Fig. 3.5). Each component has its own component factory [6] to create the component objects. By means of the component factory one can obtain static information about the component. The term *static* points out that there are no created component objects. This information is either used for a component object creation [6] or to navigate through a component list. On the contrary dynamic information such as a list of component parameter values, its input and output data configurations, etc, requires that component objects should be created.

The main element of the component static information is its unique identifier [7]. Knowledge of a component's identifier helps us to obtain full information about the component. The attachment of unique component identifiers allows us to develop a simple and efficient component control mechanism. Particularly, a composite component that is aggregating (see below) other components should include their proxies [6], which have the component identifiers as proxy's members.

Fig. 3.6 demonstrates an existence of *the component inheritance* that is supported by the NiMax system. From any component one can derive a new component. Thus, we distinguish base and derived components. Standard interfaces of a base component and derived component are joined as well as their public APIs (see Fig. 3.6).

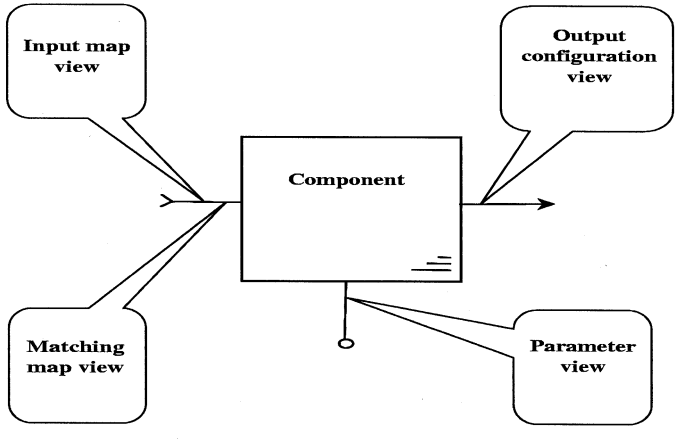

Figure 3.4: Component interface views.

The NiMax system supports also *the component aggregation.* In Fig. 3.7 we show that a composite component may include several aggregated components [4], [5].

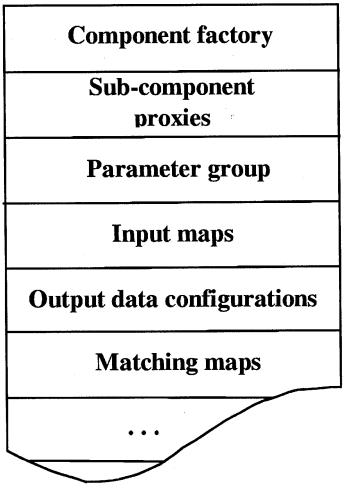

Figure 3.5: A schematic example of the component frame.

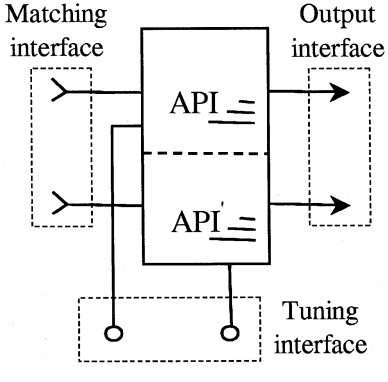

Figure 3.6: Component inheritance.

We would note that these components could belong to the different DLLs. An aggregated component may aggregate other aggregated components. A particular component may include a tree of the aggregated components or sub-components. The tuning and output interfaces of an aggregated component are simply joined to the relevant aggregating component interfaces (see Fig. 3.7).

We would stress several important features of the suggested aggregation mechanism. A component user is able to see a composite component structure and has access to any sub-components by means of the GUI.

The inheritance and aggregation mechanisms provide a unique possibility

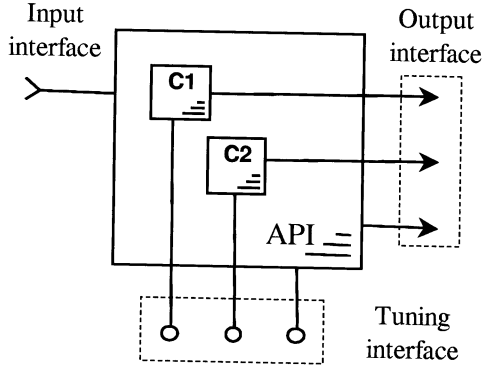

Figure 3.7: Component aggregation.

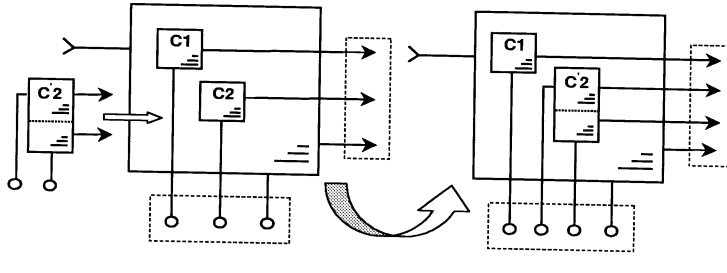

Figure 3.8: Sub-component substitution.

for the composite component users to change component algorithms without touching of the component source codes and their recompilations. We refer this possibility as *the runtime substitution of sub-components in composite components* (see Fig. 3.8).

3.2.4 Application modules

Several implemented components may share common functions and classes as well as common data that are related to a particular application domain. The development of a component related software that does not belong to particular components facilitates the work of component developers and increases developer's productivity. Such development is also argued by more efficient use of computer resources and gives a possibility to create scalable system from independent blocks.

We suggest packaging of built components with some related software into modules. We refer these modules as *the application modules* due to the relation of their content with application domains. For example, they can be the hadronic model modules (see below) or modules to predict electromagnetic processes or modules to describe a low-energy neutron transport in fissile media, etc. An example of the module content is presented in Fig. 3.9. Besides of the components, application data types (see the next section) and

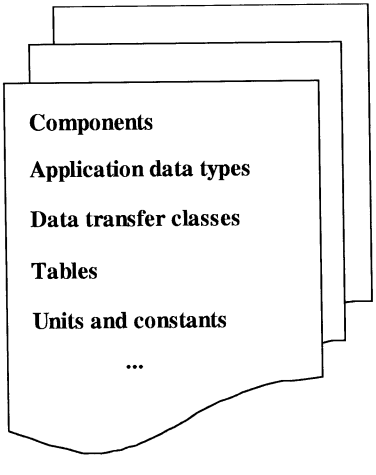

Figure 3.9: An example of the application module content.

data transfer classes [2] are members of the application modules. It is important to stress that application modules hide all software from the component users excepting the components.

The application modules can be self-sufficient on the level of source codes, i.e., no external methods, no external implementations, etc, are required to compile and execute the module components. It allows us to use these modules as distribution or exchange units. However, the module independence does not forbid use of components from other modules. It does not pose a limitation for the inheritance and aggregation mechanism. For example, it does not forbid the sub-component substitution in the case, when an aggregating component and some alternative component belong to different dynamically linked libraries (compiled modules). Thus, the application modules can either be completely independent from each other or they connected with other modules for some applications. In the last case they include lists of required module references.

A large set of hadronic model components have been already implemented and included into hadronic model modules (see the hadronic modules chapter).

The MeV, GeV, barn, Plank constant and other units and constants are included into the hadronic model modules. We have adopted a convenient strategy to use the physical units and physical constants from the GEANT4 toolkit [8].

Physical tables that store information about physical properties of particles and nuclei may be parts of the modules. Table information is requested in read-only mode. This fact opens a possibility to handle the tables by external tools, e.g., by external databases. We have already created several database tables containing particle and nuclear properties. Each of these tables can be visualized, edited, extended, etc, separately by the Microsoft database tools. We are working on a standard Structured Query Language (SQL) interface for the components to provide their access to data tables stored as database files.

We have included many special and utility functions and classes, e.g., 3 and 4-vectors, random number generators, sorting methods, etc, into the hadronic modules. A large number of such classes are borrowed from the CLHEP library [3].

3.2.5 Component documentation

Each component is accompanied with its documentation. The component documentation includes names of the component authors, copyright description and license agreements, description of the component applicability, its

input maps, parameters, matching maps, output configurations, component's API, sub-components in use, etc. The component documentation is realized as a set of HTML files.

3.3 NiMax component project

In this section we are going to discuss several mechanisms and tools for the component development as well as *application data types.* They are different parts of a component development concept referred as *the component project.* Within a component project the component developers are also able to use data transfer classes (see the data model section).

3.3.1 Component wizards

The observation of many common component elements allow us to develop *component frames* (see Fig. 3.5) [2], [3] with the aim to produce the so-called skeleton components simply by editing of the particular component frame. Such frame can be thought as a component template that is supporting a correct programming style. Thus, a component developer needs to design only a component application algorithm. To help component developers we have designed *component wizards* that are similar to the Microsoft Visual C++ 6.0 class wizard [1]. A component wizard is a code generator that produces a component skeleton by means of the use of dialog windows. The result of its work is a set of correctly related files with classes and interfaces.

3.3.2 Application data types

With the aims to increase a productivity of hadronic component developers and provide a robustness of the component algorithms we have created *an application data types (ADT)* library of classes. The library was developed to represent key abstractions (application domain data types) and operations that are used to manipulate the types. The development of MC event generators is our problem domain. Thus, the most of the ADT objects are counterparts of real high-energy physics objects (particle, parton, nucleus, string, etc).

We are extending our ADT library to help algorithm developers in their work on the hadronic model components and on components for simulation of electromagnetic processes of particle and nuclear interactions at wide energy range. Our goal is to implement a large set of the ADT classes allowing an algorithm development for a simulation of the particle and nucleus transport

through composite media. We plan that new ADT libraries will include already mentioned classes for a description of particles and nuclei, classes for a description of particle transport, the so-called tracers, classes for a description of materials and geometry of the media.

3.3.3 Component inheritance and aggregation

As we already said a component developer can extend the functionality of an existing component by applying of the component inheritance mechanism (see Fig. 3.6) that is supported by the NiMax system. For example, the component inheritance is a convenient way to add either new input maps or new matching maps. A new component can be developed by the aggregation of existing components [4], [5] (see Fig. 3.7). There are no limitations for component developers to create a large and efficient component code as compared with the standard C++ coding, if they are applying the component aggregation. In this case the component coding is even more simplified. For example, no efforts are required to create and destroy sub-component objects. The framework fulfills a control of the sub-component objects (see the NiMax framework methods section). Thus, C++ programmers can use the components as usual C++ classes excepting of component object creation and destroy.

3.3.4 Application module wizards

As we already explained built components are packed with some related software into the application domain modules. We consider the application modules as development's units. For a component developer it is more natural and more efficient to work on a component code within some component related software environment than on an isolated component code. The work on isolated codes leads to code and data duplications.

To support an implementation of the module structures we have developed *application module frames.* We are working on application module generators (wizards) for component developers and a module navigator for component users that is allowing of the use of the graphical user interface.

3.3.5 Component documentation wizard

The writing of component documentation is very important part of the component development process. We have developed *documentation frames* and *documentation wizards* (for the Windows platform only) for component developers. These tools help the developer to prepare and present information for

a standard appearance. The documentation frames are different for different types of model components. They are similar to the component frames discussed below and developed in a complement of the model component frames. The documentation wizards scan source codes of the application modules (or separate components) and generate initial versions of documentation files. These files could be further edited and extended by the developer by means of dialog windows coming one by one. After the developer has passed all steps, it is possible to run the Windows HTML Help compiler with the aim to create help topics files for a search of the necessary information by keywords or contents. Obtained files are integrated into the help sub-system.

3.4 NiMax data model

In this section we are going to consider different mechanisms of the data exchange, data storage, data visualization and data processing that are supported by the NiMax system. Here we shall also introduce the main concepts that are used to describe the NiMax data model such as the data event, data file and its views, pre-defined events and pre-defined channels and data transfer classes.

3.4.1 Data events

The data event [4], [5] is an elementary data unit for the data exchange. It is a portion of data that consists only of values of the simple data types (int, float, double, etc). Any data event has its definition. The definition includes a unique identifier (sometimes we refer it as a data event type) and describes a data event configuration. The data event configuration is a set of data channel definitions. The data channel definition includes a channel identifier, channel type, channel name and more information can be added. We would stress that the data channel is an elementary unit for data processing in the NiMax system, since the most of the data operations can be fulfilled on a separate channel. An example of the data event configuration is shown in Fig. 3.10 in a comparison with C-structures. Any data event configuration is bounded by external brackets. Inside of the external brackets there can be separate channels, arrays of channels and groups of channels bounded by inner brackets. The array of channels has a special flag in the channel definition and the information about size of the array that is written in the channel data part. Inside of the internal brackets there can be again separate channels, arrays of channels and the group of channels bounded by the inner brackets and so on. Thus, each data event configuration can include a tree

of channel definitions. The basic or simple data types can be grouped inside of the data event to represent more complicated data structures, e.g., to represent C arrays, C structures as well as C array of arrays and array of structures (see Fig. 3.10). We would note that the data event is not a C++ object, i.e., it is not an instance of the definite class.

The data event data can be placed directly into a memory or stored in a disk according to the data event definition (see Fig. 3.11).

3.4.2 Data file and its views

A component my write and read some data in the data file. The simplified structure of the data file is shortly explained in Fig. 3.12. Of course, the data file has its header (it is not shown in Fig. 3.12), which is needed to identify the file and facilitate navigation through it. The data file is separated into two parts: the data event configuration part and data part. We would stress that the presence of the data event configurations in the data file allows the system user to perform a structural data analysis. The data event configuration part has also its own header and includes a list of the event definitions with their unique identifiers. The event configuration header keeps some statistical

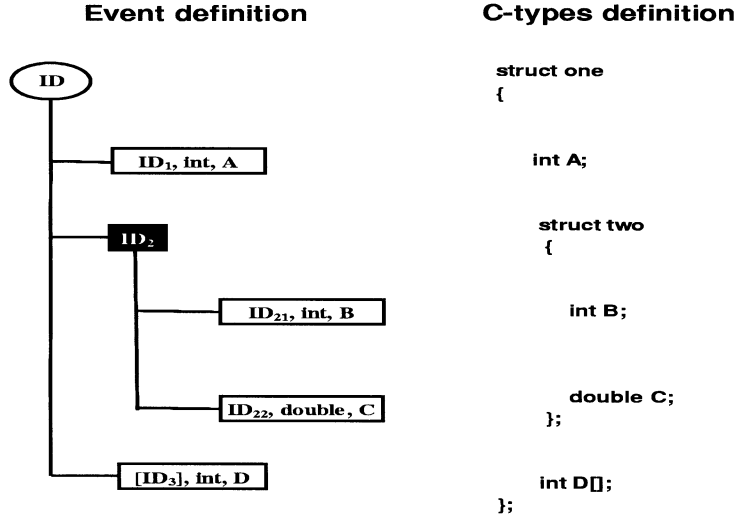

Figure 3.10: Data event configuration.

information about the data events. The data part consists of data records, where data are written in the correspondence with their configurations. Each data record has its header showing of a record size and its position in the file. We would stress that an essential feature of our data file is that it can easily be read outside of the system.

For the system users we have created several views for the data file. The most important of them are the data configuration view and data view (see Fig. 3.13). The configuration view allows the system user to perform structural selection of the written data (see some examples in the next chapter).

3.4.3 Data transfer classes

Many of the implemented components (see the appendix) can be considered as generators or converters of objects of *the data transfer classes* (DTC) [2], [3]. The DTCs help the component developers to design and implement component outputs. They give a possibility to obtain whole histories of the data processing or data generation by composite components. The DTCs help component developers to design and implement the component inputs that can ether be read from the data file or received from other components

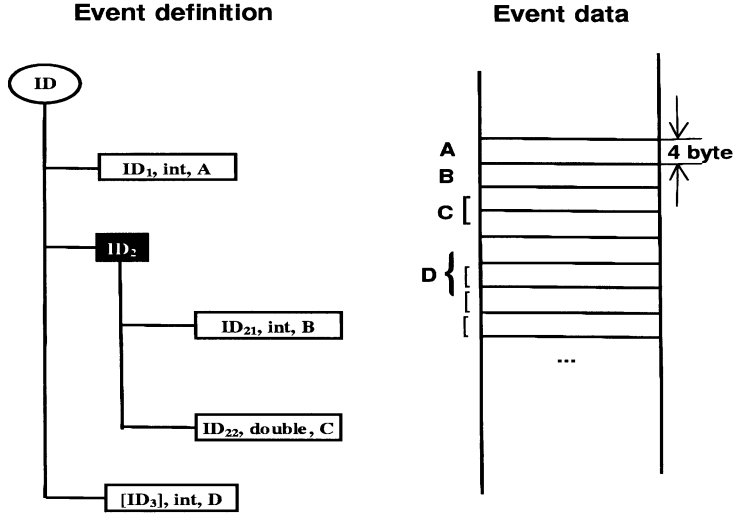

Figure 3.11: Memory placement of the data event data.

(see the component net section). Thus, the DTC library allows component developers to pay more attention for the development of component algorithms, because the developer does not need to think about details of the input-output operations. The DTCs help component developers to design and implement universal numerical algorithms. A degree of the universality

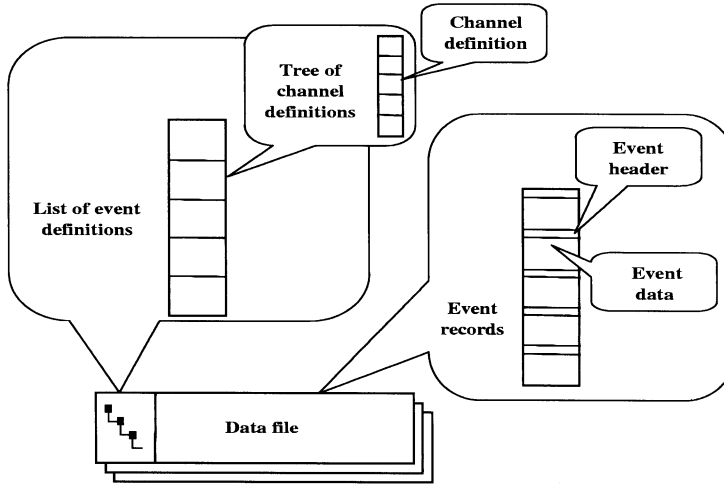

Figure 3.12: Data file structure.

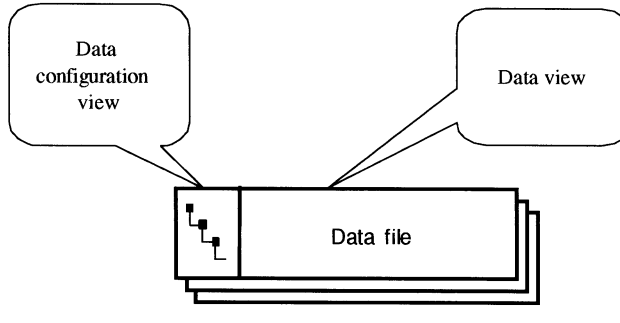

Figure 3.13: Data file views.

of an algorithm is determined by the amount of needed input-output information. Our DTC library has a hierarchical structure. The most universal algorithms are those that use, as an input and output, the objects of node data transfer classes.

We would note that data transport classes are essentially the same as the application data types classes, which were discussed before. By introducing of the data transfer term for a set of classes we would stress their importance for the data exchange. Beside of the implementation of MC model components it is necessary to develop similar sets of classes in other application domains. Within our system we have no strict recommendations how to build such classes. The most important thing is that any object of the application data type class should support a serialization in the sense that it needs methods to write to the data file and read from the data file (see the NiMax data model section) its object states. Taking into account this fact we have created *data transfer class frames* (they are similar as the component frames mentioned above) with the aim to help the component developers.

3.4.4 Pre-defined data events and channels

We have suggested [4], [5] a concept of *the pre-defined data events and channels* within our system. These events and channels are associated with definite services that are offered for the system users. The main idea of this concept is to increase a productivity of the system users that belong to particular application domains. This concept allows an independent extension of system service facilities, e.g., to create user interfaces adapted to particular applications. For example, in the high-energy physics domain the users often use histograms. The contents of one- and two-dimensional histograms and two-dimensional plots can be written, read and visualized as the pre-defined groups of channels as it is shown in Fig. 3.13 (see some histogram and plot examples in the next chapter). To deal with such data events and channels the framework needs to know only their identifiers. Thus, the framework knows (due to the unique identifiers) how to display table and graphical views for the pre-defined histogram and plot channels The component parameters, input maps and proxy sets are other examples of the pre-defined events representing component states. Again, the framework knows how to display and execute such data events.

We would mention that the pre-defined channels and pre-defined group of channels have fixed configurations (similarity with C++ objects) and their identificators can be considered as their types. The system user is not able to change their configurations.

3.5 NiMax component net

In this section we are going to discuss the interaction between components, when components exchange their data. They exchange data by means of the data file or through the virtual data stream. In the second case some of the interacting components are joined into a component net and a direct data exchange takes place. Before to describe the component net concept we would like to explain how a component is able to select needed input data from the data produced by another component.

3.5.1 Matching of the data configuration

The idea of an input data selection for a component from the data produced by another component [4], [5] is illustrated by the Figs. 3.15 - 3.16. Produced data events have linear or tree structures. They are described by their event configurations as it was explained above. The framework analyses event configurations and searches a necessary sub-configuration for a component. We refer this process as *the matching of a data configuration.* The necessary data configurations for a given component are described by the component matching configurations or matching maps. If the necessary configuration is found we refer this situation as an observation of an entry point into the data described by the configuration.

There are some peculiarities, when a component receives a data event produced by another component. It either takes a part of the event data

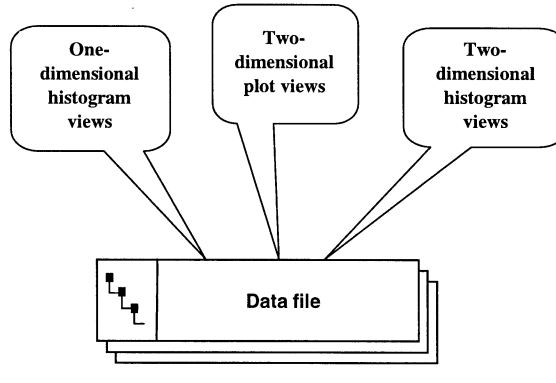

Figure 3.14: Pre-defined data event views.

according to its matching configuration or it absorbs the whole event data, where such matching configuration was found. We consider components with the second type of their behaviors as *data event filters*. The filtered data events can be modified as well.

The component developers may offer several matching maps. Thus, there is a room for the component users to tune such component with the aim to recognize an event channel or several channels from sent or read data events that it expects to receive. Any of the matching configurations can be registered as a default one. It can be a situation, when the framework will find several suitable (according to a matching map) sub-configurations in the analyzed data event and several entry points into the data are obtained. It is another room for the component users that is connected with a choice of an entry point.

It is important to stress that working on a particular component implementation component developers do not need to learn or use system or component classes (objects) with the aim to support a component collaboration [4]. The component developers do not need to learn configurations of the data events that can be received by a component. The component developers are not required to have knowledge about the source (the data file, other components) of events that the component will receive. He or she has to know how to write configurations of the required input data.

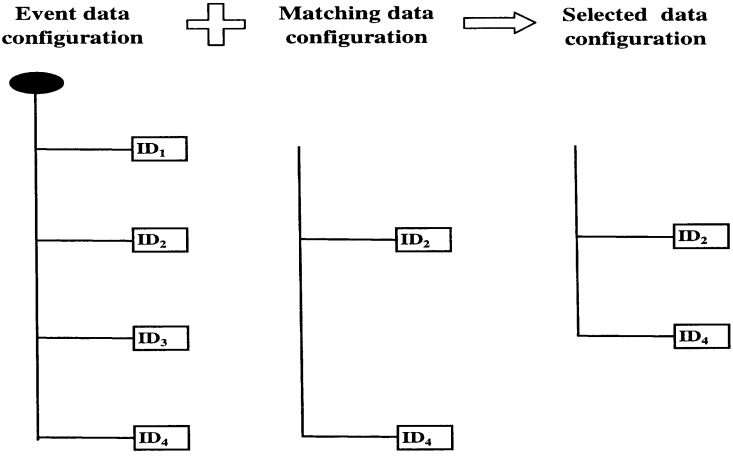

Figure 3.15: Selection of the linearly structured data.

3.5.2 Component collaboration

The components collaboration is their interaction through standard output and matching interfaces [4], [5]. We distinguish two different situations for the component collaboration (see Figs. 3.17 - 3.18, where black arrows show data flows).

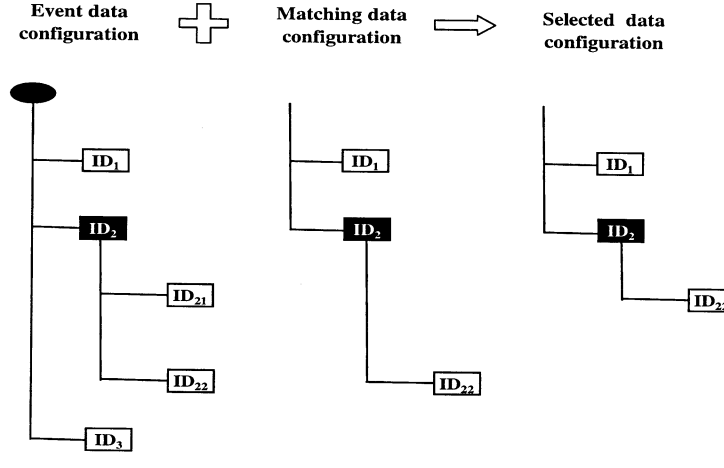

Figure 3.16: Selection of the tree structured data.

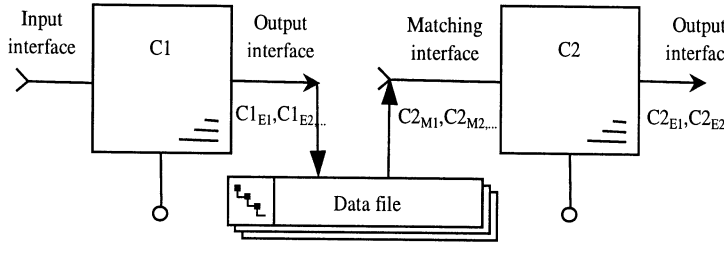

Figure 3.17: Component collaboration through the data file.

The first one is that a component writes its output in a data file and another component reads data from the data file for further processing. We demonstrate the first situation in Fig. 3.17. In the figure a component $C1$ produces many data events having different configurations: $C1_{E1}$, $C1_{E2}$, etc, and writes these events in a data file. A component $C2$ reads the data file and selects data according to matching configurations: $C2_{M1}$, $C2_{M2}$, etc. We consider the shown situation as the component collaboration through a data file bus having in mind a hardware analogy. For this type of interaction component objects are completely isolated from each other.

We would stress that a reading component does not need to wait for when a writing component finishes to write all events. It can start to read data events immediately after the first event has been written. This feature is particularly important for a data monitoring.

The second situation (see Fig. 3.18) is that one component produces a data event output, which will be received by another component as an input. We consider this situation as the component collaboration by a data event bus. For this type of interaction several components are executed inside a common process.

3.5.3 Event-oriented component nets

A set of components that collaborates through the standard output and matching interfaces by sending and receiving data event messages is referred as an

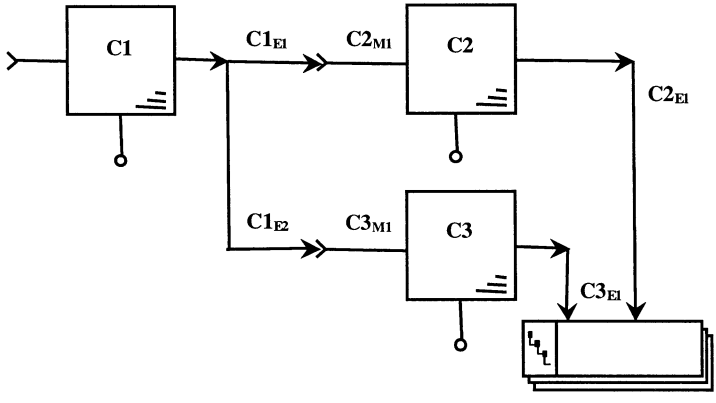

Figure 3.18: Event-oriented component net.

event-oriented component net [4], [5]. Fig. 3.18 demonstrates an example of a net. Here, a component $C1$ produces two events having configurations: $C1_{E1}$ and $C1_{E2}$, then components $C2$ and $C3$ receive some selected data of these events. The data are selected in the accordance with matching configurations $C2_{M1}$ and $C3_{M1}$, respectively. The components $C2$ and $C3$ produce new data events that are configured as $C2_{E1}$ and $C3_{E1}$, respectively. The last events are written in a data file.

There is a direct hardware analogy of the described component net example and the matching mechanism of input data selection that is presented in Fig. 3.19. In this figure different lines denote different wires that may link components. Several wires can be screwed into a cable. A cable can include sub-cables ("tree structure" of a cable) as well. The wire analogy is a data event channel (simple data type) and the cable analogy is a group of channels or a data event configuration. The analogy of a data configuration matching is that only some wires or sub-cables from an outgoing cable are chosen for component connections.

Following the hardware analogy we would like to provide more details about component connections. Inside the output and matching interfaces we can distinguish groups of methods that are referred by us as the output and matching connectors. A component processes data through a connector. Components can be wired through the connectors. Any connector deals with two important things: a data stream (a data file, memory) and data

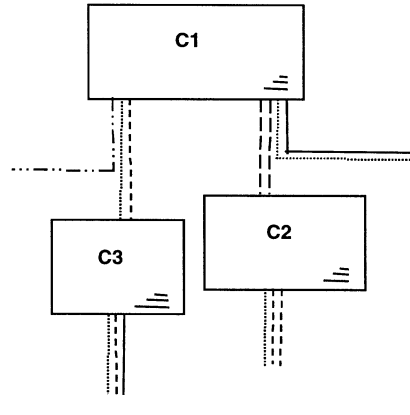

Figure 3.19: Hardware analogy of the event-oriented component net.

event configuration (an output configuration, matching configuration). The connector concept allows the component users to redirect and reconfigure component output and input data. The user can choose a particular stream or tune a particular output configuration or choose a particular matching map (see some examples in the next chapter).

We would note that a sub-component of an aggregating component cannot be linked with some components or a data file through its matching connector. All components in a component net deal with only one data file.

We consider only the nets, those component execution orders are defined by data flows, i.e., we consider only pipeline nets. Thus, any net includes the component that is executed at first. We refer this component as the start or main component (it expects to receive the so-called start event). It reads its input by the input or matching interface. We should note that any component or sub-component from a net is able to write its output to a data file. So far each component can have only one input connector and one output connector and such components could be linked only into "linear" pipeline nets.

Any event-oriented net needs an event control procedure should be written besides the information about component connections. But, for pipeline nets, such procedure can be written and compiled at once and hidden from the system users.

3.5.4 Component net views

Any net is a NiMax system file that consists of two parts. As any file it can be saved for later re-use, re-named, deleted, etc. The first part can be considered as the net definition part. The second part includes a data event control procedure. This first part is constructed by the framework on the basis of the information obtained from a user through the net views. The framework generates the second part by itself. The net definition part includes the information about pairs of collaborated components. It notifies also a start component and end-net components. Providing such information one assembles a net through the net views. By the net views one can select any net component and show its structure, parameters, output configuration and matching maps as well as input maps for a main component of the net (see the net editor examples). Very often a net includes only one component. Thus, performing a component execution, we always deal with a net file. Any component, which can be a net member, is referred as a runnable component.

3.5.5 Component aggregation and collaboration

At the end of this net section we would like to stress the main differences between the component aggregation, when a composite component includes some sub-components, and the component collaboration, when several components are working together within a net (see Fig. 3.20). The aggregated sub-components interact through their APIs, i.e., by calling of their methods. A developer writes the code of an aggregating component. This code should be compiled before the component execution. In case of the component collaboration they interact through their standard interfaces. A user assembles a component net by the graphical user interface. Any net is a system file that is ready for execution, i.e., no compilation is required. It can be loaded into a memory and executed as a standalone process having a good performance.

3.6 NiMax framework methods

The full description of the framework classes, data transfer classes and other classes that belong to the NiMax system will require a separate manual. We plan to write such manual with the aim to help component developers as well as programmers, who wants to integrate our system into other packages, e.g., the GEANT4 [8] toolkit or ROOT [10] framework. It is also a necessary task for supporting of a further development of our system.

Here, we would like to explain shortly the basic functionality of the framework API based on the application of control and navigation methods.

We would note that we separate data file control and navigation methods

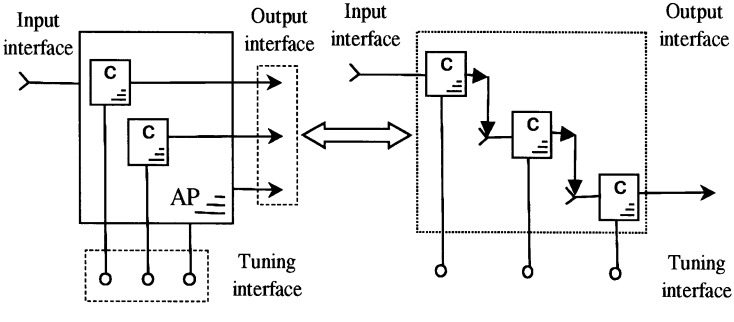

Figure 3.20: Component aggregation and collaboration.

from the rest framework methods. Such separation makes sense, because the data file, data file views and data file control and navigation methods are joined into an independent data file system having its own applications.

Before to start the description of the control and navigation methods we would like to say a few words about object identifiers. Any object of our system has its unique identifier as well as any data event discussed above. The knowledge of an object's identifier offers full information about the object. At the moment for identifier's values we use integer numbers (unsigned type). Thus, we have a possibility to assign more than 4 billion different values. The integer numbers are very convenient for searching and navigation.

3.6.1 Control methods

There are many methods to control the component life cycle, i.e., a component instance creation phase, edition phase, execution phase and destruction phase.

In spite of the fact that the component life cycle consists mostly of internal framework processes, which are hidden from the framework users we would like to give an idea about it. There are many possibilities for the user to influence the component life cycle.

Before to start an object creation procedure the framework creates an environment for the component object. The content of this environment defines creation mode, optional variables, which are set to default values, and output and input files, if they will be used. The users are able to modify default values of these optional variables by means of the user interface, e. g., the users can either set their own default parameters and input map or suppress some data event output or suppress some runtime information output, etc.

There are two special modes of any component (excepting the virtual one) for an instance creation. The first special mode is the instance creation for only information purpose. This means that the user cannot make any changes of a component object. This mode provides a possibility for the user to learn the component structure by the user interface. The second special mode is the debug mode. It gives a possibility of creating a general component instance without the permission to execute it. This mode is added in order to debug the component interfaces.

In the case of a composite component, the aggregating component instance is created at first. Then the framework will create its aggregated component or sub-component instances. The order of the aggregated instance creation follows their definition order in the aggregating component. In order to create any component object the framework needs to know only

the component's identifier. It uses sub-component's identifiers to look for their factories. If a factory is not found, the framework tries to find an alternative sub-component according to the component proxy definitions and component inheritance hierarchy. The user is able to control this process by an enabling or disabling of a component substitution. The creation process is repeated for each sub-component component and for their sub-components until all component objects will be created.

The destruction phase for created component objects is fulfilled in a reverse order as compared to the construction phase without the user influence.

During the component edition phase the user is able to edit parameters and input or matching maps as well as reconfigure component's output. Check methods are called to control a consistency of the edition. In the case of a non-consistency these methods send warning messages and set back to the default values the values of edited variables.

The framework helps the system users to create, edit and execute net files and controls these processes. During the execution phase the framework supports component runtime information output: information messages, warning messages and error messages (see some examples in the next chapter). In the case of warning or error messages help information (see below) is offered for the interested user.

The framework allows the user to execute several component nets as separate processes and control their executions. In the case of an error the framework detects itself a place of the error and the component developers do not need to make some special efforts to solve this task.

3.6.2 Navigation methods

The framework fulfills component librarian functions. It allows the user to visualize a total list of the components are included into the system and register a required component. Before a component registration the component views show static information, because the component objects are not created yet. The component registration means the creation of component objects. The framework allows the user to look through net files, open them for net editions and save the edited nets.

Tree structures are heavily used in our system, e.g., the tree structure of composite components and tree structure of data events. Thus, the methods to navigate through a composite component and through a data file are the same.

Here we would like to mention that using file navigation methods the model developers are also able to write adapter or driver tools to transform

the format of a data written down to the system data file into data formats, which are acceptable for the external packages.

Besides of runtime information and different information messages, which can appear during a component life cycle, the system users are offered more detailed help information. The users can navigate through the help documentation either by contents or by keywords.

As we already said any object in our system such as a component, parameter, error message, etc, has a unique identifier. It opens a possibility to bind these identifiers with HTML files that are describing the objects. Thus, the users may obtain a help from the "inside" of a code by means of a unique identifier, e.g., by a parameter identifier the framework opens the HTML file that describes the parameter.

3.7 NiMax graphical user interface

The NiMax GUI is developed in the accordance with the document - view technology [3] that was shortly discussed at the beginning of this chapter. This technology is very suitable to take into account the system user requirements.

After starting of the NiMax system, the main window will appear. This window includes many views. These views can be thought as different navigators and editors. All available components are displayed for system users. The system user can launch any component and open any available component net file for a further edition. By means of the component editor the user can edit input maps, parameters and substitute sub-components. We would stress that the system user sees the whole structure of a composite component and can edit and perform substitution of any sub-components of this component. The user can reconfigure the component and sub-component outputs. By the net editor the user can assemble several components into a component net and modify an existing net. The system user can perform controlled executions of several nets as separate processes. The user can navigate through a data event configuration and data event data in a data file performing the structural data selection. The user can visualize the data produced by components as one-dimensional histograms and two-dimensional histograms and plots. The user can navigate through the help documentation looking for necessary information about a particular component or the NiMax system itself.

Bibliography

[1] Kruglinski D. J., Shepherd G., Wingo S. - Programming Microsoft Visual C++, Fifth Edition, Microsoft Press, 1998.

[2] Amelin N. and Komogorov M., An Object-Oriented Framework for the Hadronic Monte-Carlo Event Generators. JINR Rapid Communications, 1999, No. 5-6 [97]-99, p. 52-84.

[3] Amelin N. and Komogorov M., An Object-Oriented Framework for the Hadronic Monte-Carlo Event Generators. In Proc. of Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP2000), 7 - 11 February 2000, Padova, Italy, p. 119-123.

[4] Amelin N. and Komogorov M., NIMAX: A New Approach to Develop Hadronic Event Generators in HEP. PHYS. P&N LETTERS, 2000, No. 3 [100]-2000, p. 35-47.

[5] Amelin N. and Komogorov M., NIMAX System: A New Approach to Develop, Assemble and Use Numerical Models in HEP. The talk presented at the XV Int. Seminar on High Energy Physics Problems, Dubna, Russia, 25 - 29 September, 2000 (will be published in Proceeding); JINR Preprint, E1-2001-31, p. 1-14.

[6] Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns. Elements of Reusable Object-Oriented Software, Eddison-Wesley, 1995.

[7] Orfali R., Harkey D., Edwards J., The essential distributed objects. Survival guide., John Wiley and Sons, Inc., 1996.

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

[9] Class Library for High Energy Physics,
http : *//wwwc*1.*cern.ch/asd/lhc* + +*/clhep/index.html*

[10] Brun R. *et al*, ROOT System, CERN/HP, 1997. See *http* : *//root.cern.ch/*.

Chapter 4

User session

4.1 Component and net navigations

The available model components and component nets are displayed for the system users. There are three different possibilities to look through the total component list (see Fig. 4.1): by categories (default view) to show the components from different application categories, by modules (DLLs) to show the DLL component contents, by hierarchies to show the component inheritance relations. The component views show also the component types that are defined by a presence or absence of particular standard interfaces [1]. The icons mark the component types: runnable components, general components, virtual components, etc). The file browser fulfills the function of a net navigator. It offers access to component nets by the open command from the file menu. The net user can also rename, copy and delete the net files.

4.2 Working on the component states

The framework offers for the component users a possibility to change a component state (see Figs. 4.2 - 4.3) by means of the component interface views. The user can see, register (instead of a default input map) and edit component input maps as well as see and edit component parameters. The framework checks an inconsistency of the parameter edition (see, e.g., the warning message in the bottom of Fig. 4.2). During input map and parameter editions the user always has a possibility to set back their default values. The user has also access to any parameter of a sub-component inside a composite component (see Fig. 4.3).

A sub-component can be substituted by another sub-component, if they

have a common base component. To see that components have a common base component the user can use the component hierarchy view. Thus, inside the composite component an aggregated component can be substituted by an alternative component without coding, i.e., by means of the user interface as it is shown in Fig. 4.4. The component user can perform substitution inside any sub-component, if it is a composite one. It is a way to change and update the application algorithms of composite components.

If a component is expected to produce an output, the component user has a possibility to control it. The total events or only some selected channels can be disabled for the output. In the case of composite components, the output reconfiguration can be performed for any sub-components (see Fig. 4.5).

If a component is expected to receive data either from another component or from a data file, the component user has a possibility to choose and register

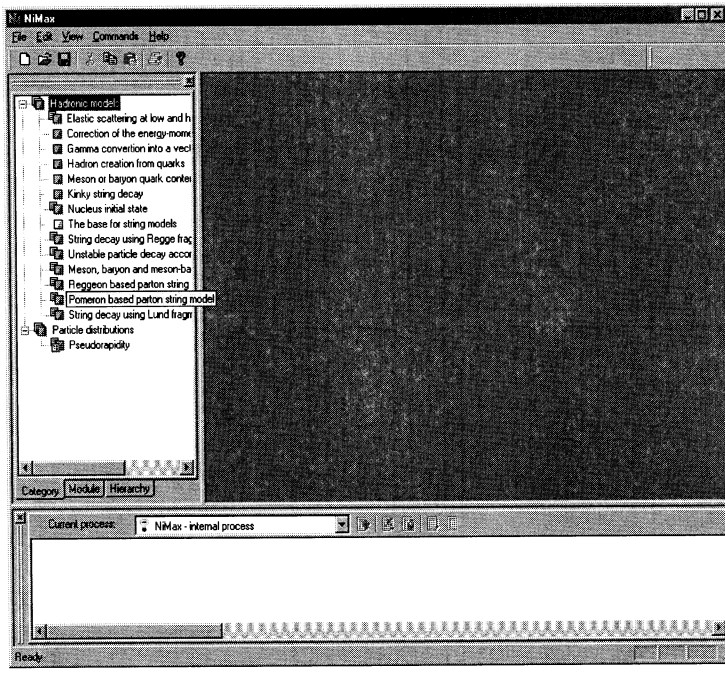

Figure 4.1: Component category view.

a suitable matching map among the matching maps offered by the component developers (see Fig. 4.6).

There is also a possibility to check that a matched data configuration is really exist in an attached data file (see Fig. 4.7) or it will be produced by other collaborated components that belong to the same component net (see the next section).

4.3 Assembling and edition of the component nets

The system user can open a component net file for an edition. He or she can create a new component net file as a starting point of the component net assembling. The system provides an environment for the net assembling and

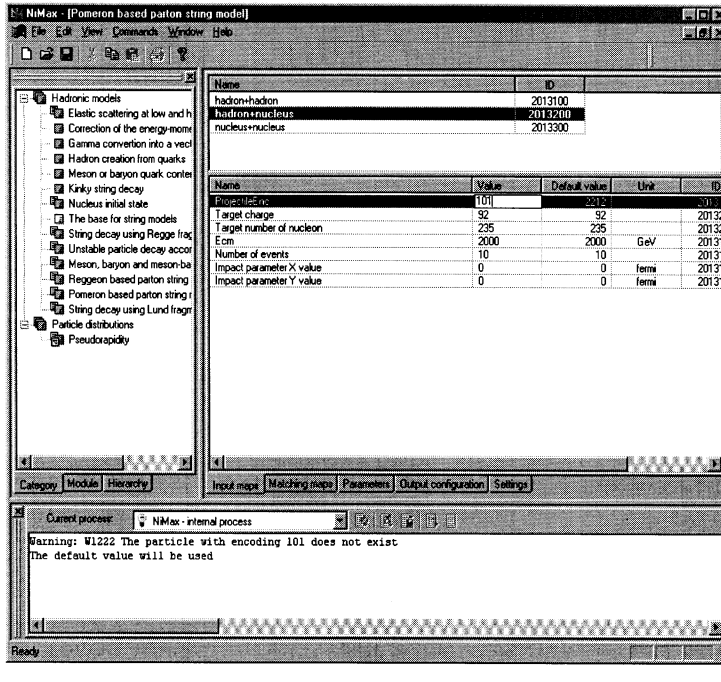

Figure 4.2: An example of the component input map edition.

execution, e.g., a named output file. All available components are displayed and the user can launch several of them in a component net for a collaborative work (see Fig. 4.8).

Then the user should connect them one by another by their output (on the component right sides) and matching (on the component left sides) connectors. Thus, the data stream for the pair of connected components flows from the right side of the component that sends data to the left side of the component that receives the data. Providing the links between component connectors the user is able to reconfigure component outputs and to choose (register) suitable matching maps. The user has also to choose some required entry points, if several of them are found for a particular connection (see Fig. 4.9).

The component connections are marked by dashed lines (see Fig. 4.8). If the component user activates a particular connection, e.g., to reconfigure

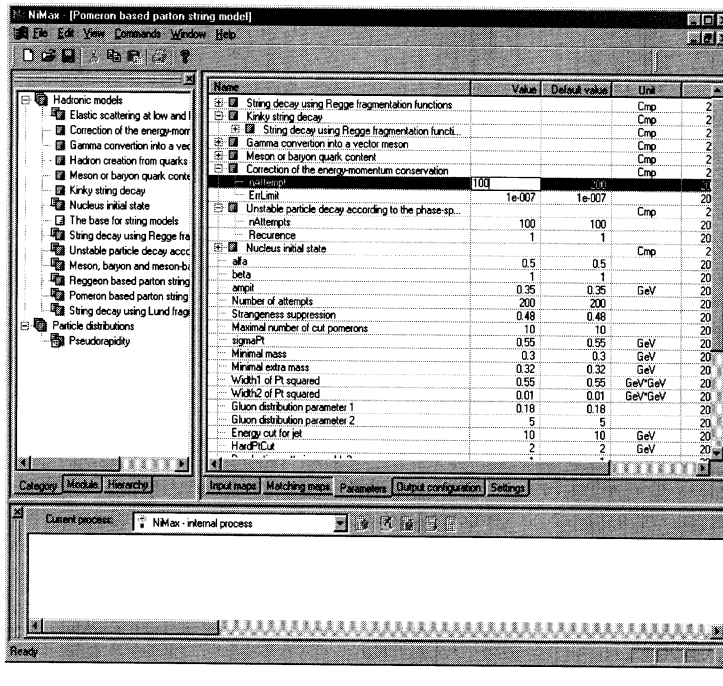

Figure 4.3: An example of the parameter edition of a composite component.

a component output, then a dashed line marks such activated connection with an additional shadowing (see Fig. 4.9). Solid lines point out the correct connections between pairs of components. The start component has a red color and rest (connected) of components have blue colors.

Some components can be removed from a net file as well. For a particular net its user is able to select one group or several groups of connected components that do not include a start component. Thus, such group is not a complete net and cannot be executed, but the user can save such group and use it during the creation or edition of another net. Finally, the net user can save the net file by attaching a particular file name.

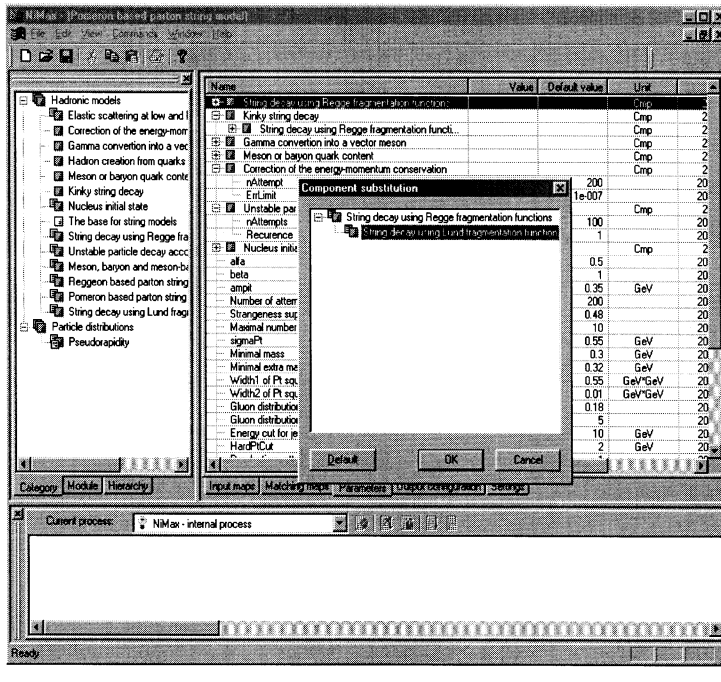

Figure 4.4: An example of the sub-component substitution.

4.4 Control of the component net execution

The component user has a possibility to reset an execution environment for a component net before its execution (see Fig. 4.10), e.g., to change the name of file, where the output data will be written. After that he or she can begin to run the chosen net or even several component nets as separate processes (see Fig. 4.11). The component nets inform about their execution processes during a run session by means of runtime messages. The runtime information is simple text based information. There can be information messages, warning messages and error messages. The information messages are used to tell something about normal execution processes. The warning messages inform the component user about potential errors or other situations, which are able to destroy a normal execution process. The execution process will be continued after the appearance of the warning messages. The error messages

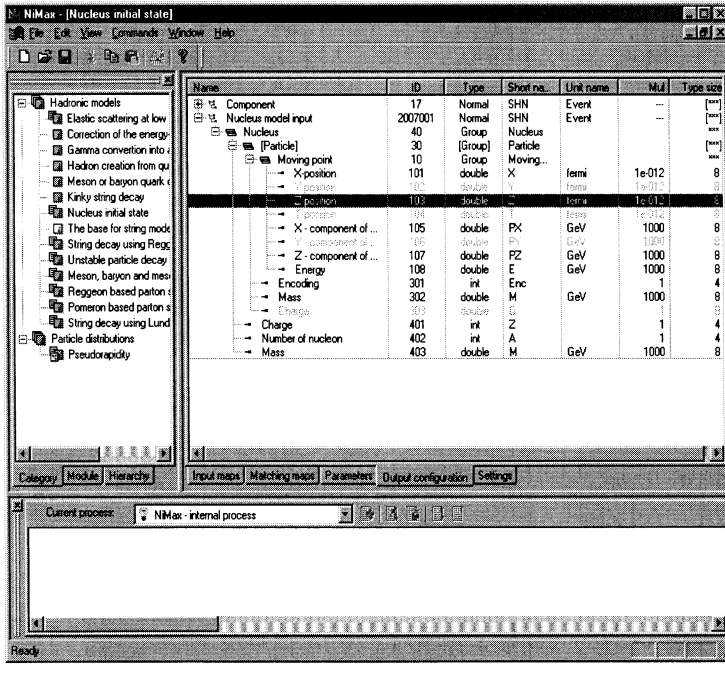

Figure 4.5: An example of the component output configuration.

appear, when an execution process is aborted. The error messages inform the user about a place and type of the error. Each warning or error message has its own unique identifier. It allows the framework to help the users to obtain the full information about the reason of a warning or error.

The user has a possibility to control the runtime information output. Particularly, one can either totally suppress the information messages or suppress the information messages from some component objects only.

As we already told the user can either execute many component nets as different processes at the same time or have several execution processes of one component net, e.g., different component parameters in a use, at the same time as it is shown in Fig. 4.11. The user can obtain the information about current states of the execution processes, select a process and perform some actions on it, e.g., kill a selected process, save its runtime information.

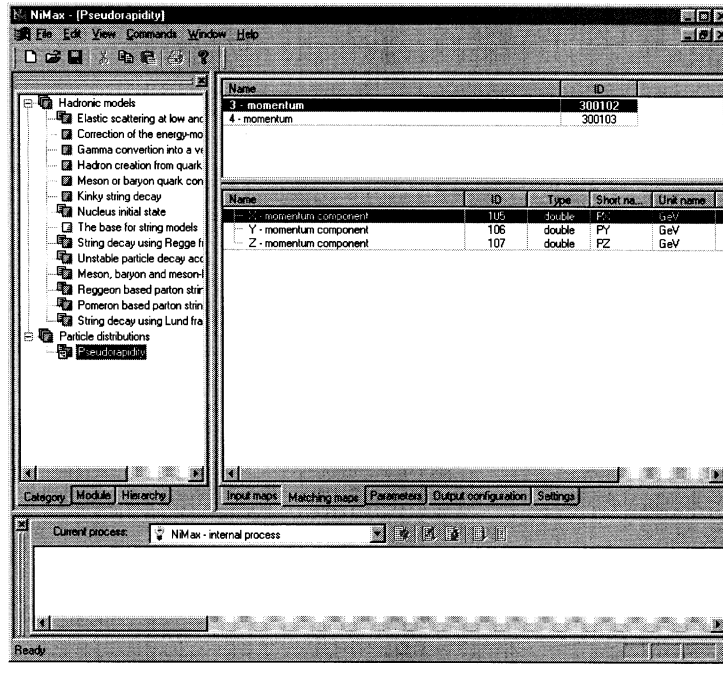

Figure 4.6: An example of the component matching map.

4.5 Working on the data files

The system user can look through the data event configuration (see Fig. 4.12) and data (see Fig. 4.13) written in data files. The whole data events as well as their separate channels can be selected through the data configuration view. We refer it as a structural data selection. After the selection one can perform different operations on data, e.g., copy, cut, protect against component access, etc.

4.6 Working on the histograms and plots

One- and two-dimensional histograms with fixed partitions can be created and visualized as well as two-dimensional plots (see Figs. 4.14 - 4.16). The

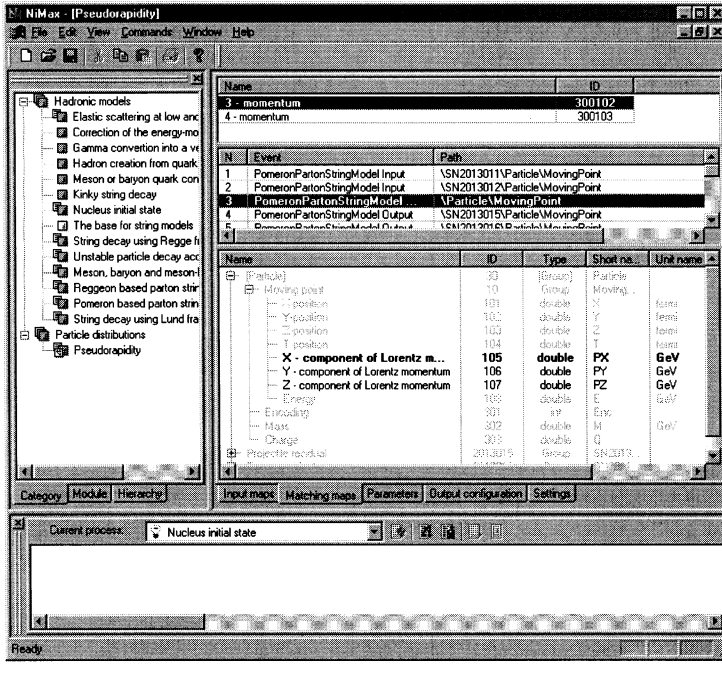

Figure 4.7: An example of the observed matching configuration.

histogram statistics is implemented as bin content statistics, i.e., average and root mean square values are calculated using positions and values of the histogram channels. Bin errors are always computed taking weights into account.

All histogram objects can store bin values and errors as the values of different types and the system user has a possibility to make a choice among these types.

For any one-dimensional histogram object the user can work on its table views as well as on its graphical views (see Fig. 4.14). Two-dimensional histograms have their table views and cell graphical views (see Fig. 4.15). The plots are presented by their table and scatter plot graphical views (see Fig. 4.16).

The system user can apply different operations on the selected data of any view. Selection methods are different for the different views. For example,

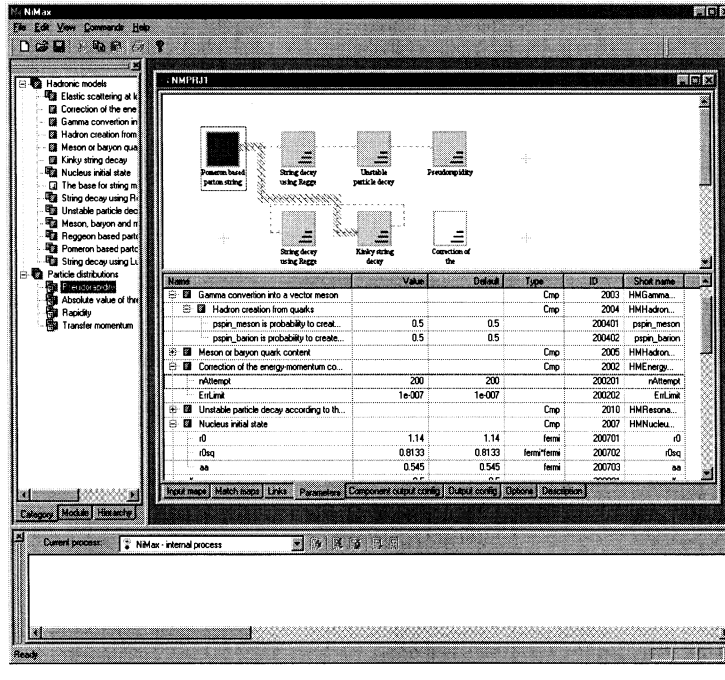

Figure 4.8: An example of the component net assembling.

the user may use a mouse to select the content of bins for a table view of a histogram object and for the graphical views we offer a brush tool.

We would like to make a remark concerning histogramming facilities that provides by the NiMax system. Many of the facilities of the HBOOK package [2] (de-facto a histogramming standard for high energy community) as well as a number of useful extensions are offered for the NiMax users, if they are dealing with the histogram pre-defined data events. It is important to understand that the histogramming facilities are offered through the histogram views. Thus, some of the HBOOK functionalities such as a histogram rebinning and histogram fitting are not discussed here. These facilities are connected with new object creations that are a component privilege. We have to develop a special component or several components for this purpose.

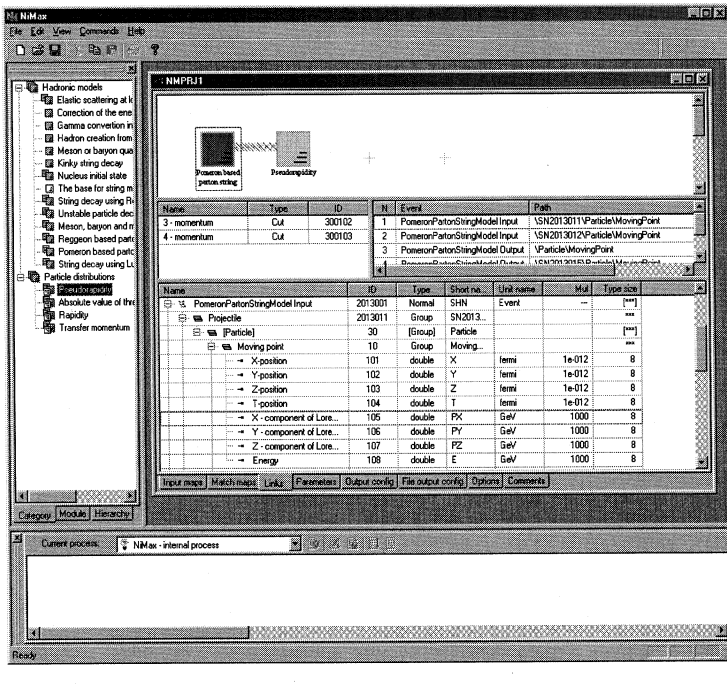

Figure 4.9: An example of the net configuration tuning.

4.7 Obtaining a help

Help sub-system has been developed for the Windows platforms by applying the HTMLHelp Workshop [3]. Such sub-system is needed to help system users and potential model developers. The help sub-system from a hadronic model developer point of view can be considered as a platform to explain how to develop model codes. There are two types of the help: the system help, i.e., help information to assist in the system usage, and the component help (see Fig. 4.17). The help topics related to the NiMax system describe how to handle components, how to process data, etc. The help topics related to a particular component include the information about the component parameters, its input maps, its output events, etc (see Fig. 4.18).

The navigation through the help documentation and search of the needed information can be done using contents and keywords.

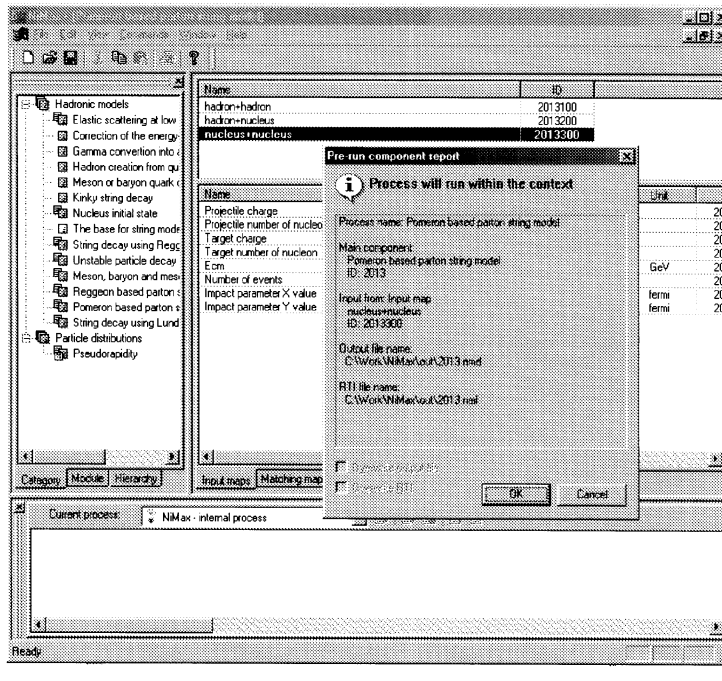

Figure 4.10: Reset of the net execution environment.

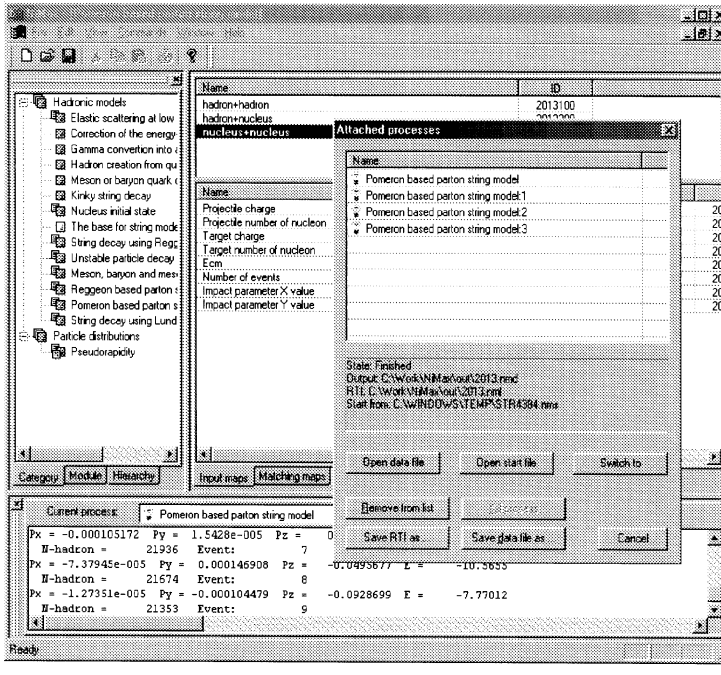

Figure 4.11: Several nets are executed as separate processes.

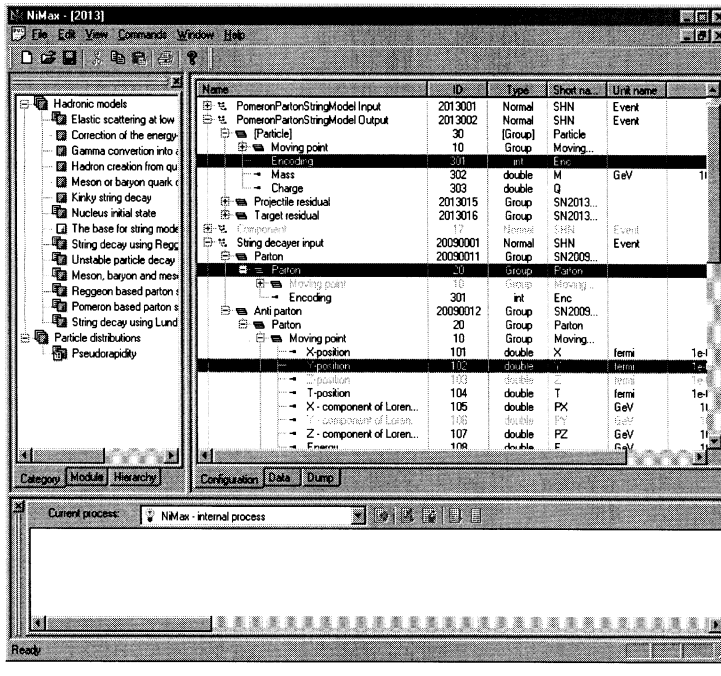

Figure 4.12: Data selection by means of the data configuration view.

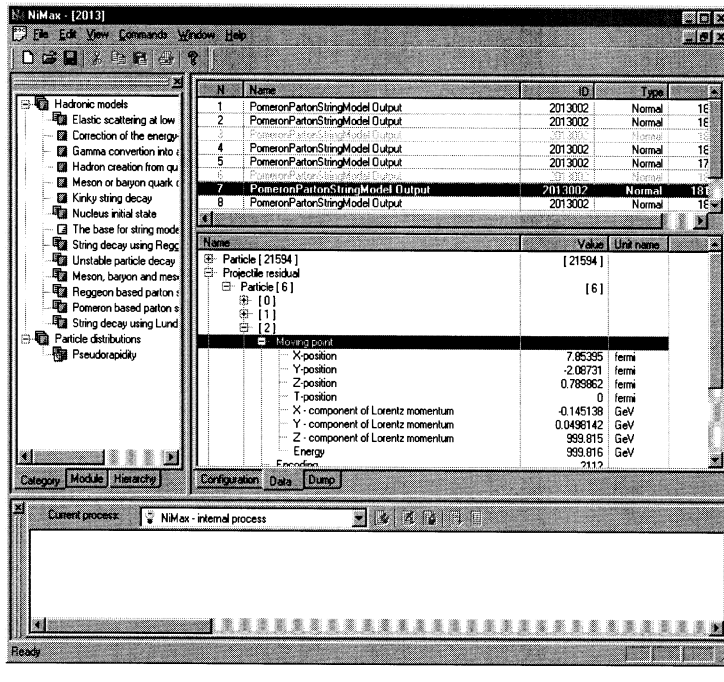

Figure 4.13: An example of the selected data.

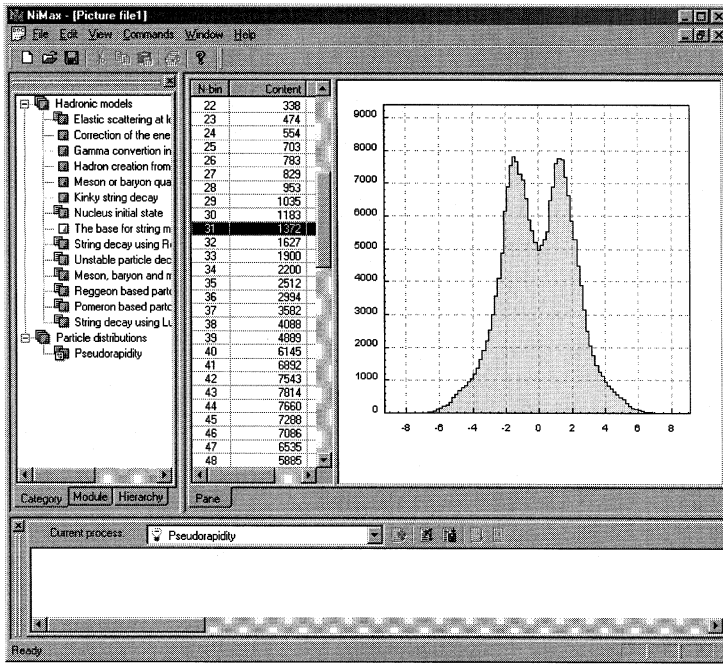

Figure 4.14: An example of the one-dimensional histogram views.

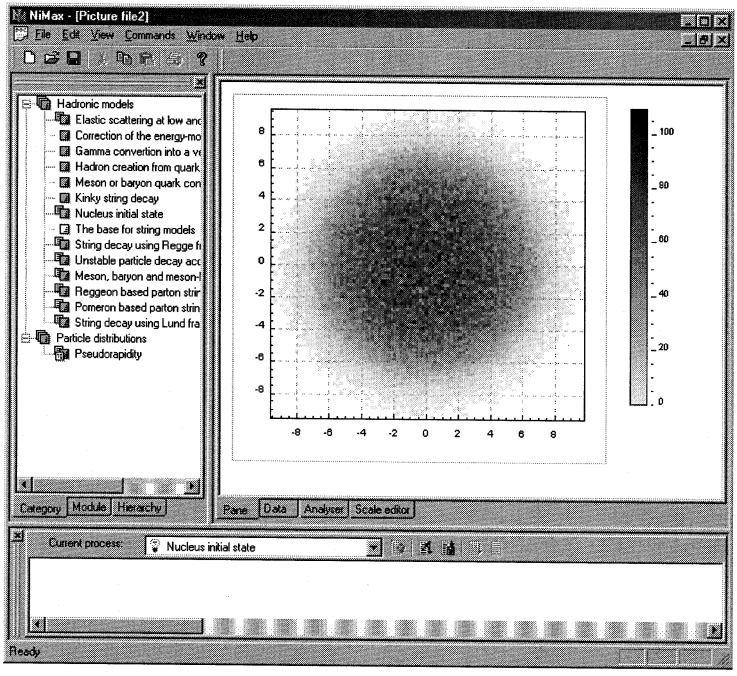

Figure 4.15: An example of the two-dimensional histogram view.

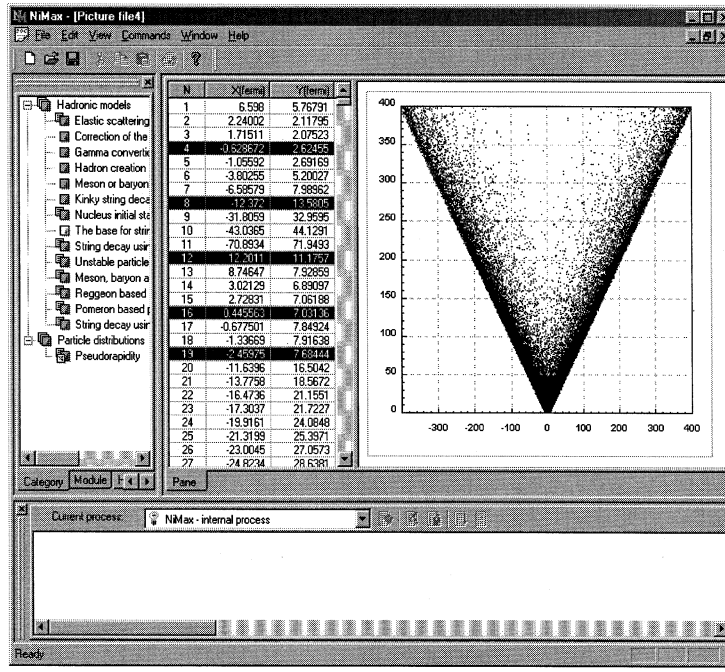

Figure 4.16: An example of the two-dimensional plot views.

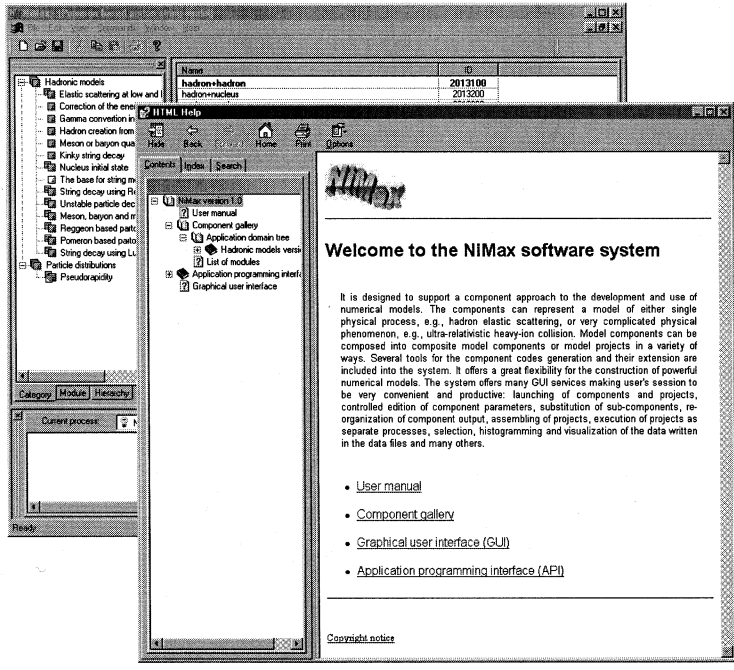

Figure 4.17: NiMax help window.

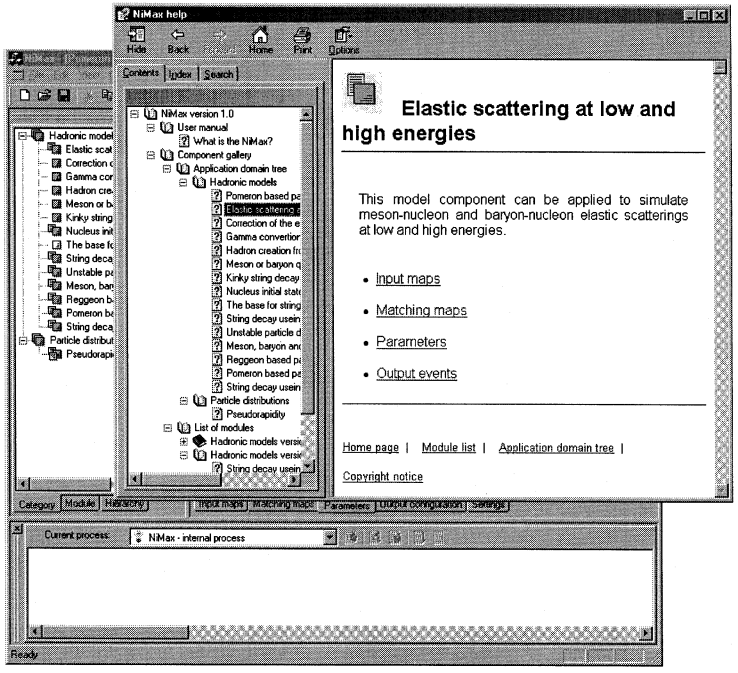

Figure 4.18: An example of the component description.

Bibliography

[1] Amelin N. and Komogorov M., NIMAX: A New Approach to Develop Hadronic Event Generators in HEP. PHYS. P&N LETTERS, 2000, No. 3 [100]-2000, p. 35-47.

[2] HBOOK Statistical Analysis and Histograming Reference Manual, CERN Program Library Long Writeups Y250, CERN Geneva, Switzerland, 1998.

[3] Kruglinski D. J., Shepherd G., Wingo S. - Programming Microsoft Visual C++, Fifth Edition, Microsoft Press, 1998.

Chapter 5

Summary and acknowledgments

5.1 Summary

We have described a new approach to the development and use of numerical phenomenological models in high-energy physics. It is the component approach, when a complex numerical model is composed from simpler model components that are included into application modules.

We have formalized a standard component or a component model by the separation of a component interface part and component algorithm part. Any component is thought as a set of standard interfaces between its numerical algorithm and the outside world.

We have described a unit of configured data: the data event. A component may produce data events, write them in a data file for further analysis and visualization, read data events from a data file and receive data events from other components.

Within the approach we offer a possibility to compose several components into a composite component or component net as well as different tools to develop a new specific components. It provides a great flexibility for the construction of powerful and realistic numerical models.

Each particular component can be tuned and different implementations of the component algorithm can be interchanged at runtime enabling a model user to obtain needed model properties without redesigning of a model and writing the model code.

To support this approach we have developed the NiMax software system. The NiMax system is an object-oriented software written in C++. It is original and unique system. Its central part is a framework that controls

all the system's internal processes and provides an interaction between the graphical user interface (GUI) and rest of parts.

The framework offers many GUI services to have a convenient and productive user session. For example, the user is able to launch required components and component nets. He or she is able to perform a controlled edition of component and sub-component parameters, substitution of sub-components within a composite component, re-organization of component and sub-component outputs. The user can assemble and edit different component nets and execute them as separate processes. The user is able to perform structural, analytical and graphical selections of the produced data that have been written in the data files. He or she can build and visualize histograms and plots.

The NiMax system offers many tools to make a productive work of the component developers. It includes different wizards of the component codes and component documentation. It includes libraries of the application data types and data transfer classes. These tools and libraries facilitate the development of interface and algorithm parts of the components. Working on them we assume that the model component developers should work on component numerical algorithms and each component numerical algorithm can be developed independently from other component numerical algorithms. Component functionalities are extensible by the inheritance mechanism and component codes are re-usable by the component aggregation.

Several concepts of our system have no analogies among the existing software systems. One of them is the component aggregation mechanism. It allows C++ developer to write a large size component code by re-using of ready components without coding limitations. For the outside world an aggregating component looks like a single component. However, its user has the access to any sub-component and is able to modify the algorithm of the composite component by a sub-component substitution. Another one is the matching of data event configuration. It allows us to organize a collaboration of independent components by transferring of complicated data structures between these components. The component developer is not required to learn or use the system or component classes with the aim to support component collaboration. The matching of a data event configuration offers a possibility of a structural selection of the data that is written in the data file.

An additional idea of pre-defined data events and pre-defined data channels facilitates data visualization and data analysis and opens a possibility for an independent extension of the existing user interface and for the development of special (for different application domains) user interfaces.

The format of the developed data file allows storage of a large bulk of the differently structured data and fast navigation through these data. The

data file, data file views and data file control and navigation methods may be thought as an independent data file system for the processing of large date volumes and having its own applications.

The html based help sub-system is a part of the NiMax system. Help topics include descriptions that related to our system and implemented components. The navigation through the help documentation can be done using contents or by keywords. The help sub-system can be use for an information exchange in a separate from the NiMax system.

We have applied the NiMax system for a particular class of the numerical models: the Monte Carlo event generators for high-energy physics. With help of the NiMax system we have developed hadronic modules. They include many MC model components that allow their user to perform a simulation of the particle and nuclear interactions at wide high-energy range and for a large set of projectiles and targets.

The NiMax system can be used in different application domains. First of all, it is suitable for the applications based on a use of complicated numerical models that generate data. We have started the development of application modules to perform computer simulations of the radiation transport in different media. Besides of the numerical simulations it can be applied for the structural (based on the data event configurations), analytical (based on the use of numerical models) and graphical (based on the data visualizations) analysis of the data obtained from different sources.

5.2 Acknowledgments

We are thankful for our colleagues from the Laboratory of High Energies of the Joint Institute for Nuclear Research, Dubna for collaborations. We would like to thank our colleagues from the Physics Department of Jyväskylä Universtity, Finland for their interest in the NiMax system and numerous discussions about the system features.

Chapter 6

Hadronic modules

6.1 Particle structure components

6.1.1 Particle properties

Particle properties have been tabulated. These tables keep the properties of light quarks and antiquarks (u,d and s) as well as the properties of diquarks and antidiquarks that are consisting of light quarks and antiquarks. We have tabulated the properties of a gluon, photon and leptons ($e^{\pm}$, $\mu^{\pm}$ and $\nu_{e,\mu}$). We have also tabulated the properties of $J^P = \frac{1}{2}^+, \frac{3}{2}^+$ baryons and baryon resonances as well as J^{PC} mesons states: (0^{-+}),i.e., π,K,η and η' and (1^{--}),i.e., ρ,K^*,ω and ϕ. J, P and C denote a spin, parity and charge conjugation, respectively. The full baryon-antibaryon symmetry is assumed.

In the case of a resonance production the resonance mass m is determined according to the Breit-Wigner distribution:

$$F(m) = 2\pi \frac{\Gamma(m_R)}{(m_R - m)^2 + \Gamma(m_R)^2/4}, \tag{6.1}$$

The particle properties, pole masses m_R and total decay widths $\Gamma(m_R)$ are taken from the Review of Particle Properties [1], [2], [3], [4].

6.1.2 Baryon and meson splitting

To perform the simulation of a string excitation we need to split baryon (antibaryon) and choose valence quark (antiquark) and diquark (antidiquark). In the case of a meson we split it into valence quark and antiquark (for neutral mixed mesons this splitting is performed according to the quark mixing probabilities). We determine the probabilities of the baryon (antibaryon) state (quark (antiquark) and diquark (antidiquark) with given spin and isospin)

Baryon type	Quark content
p	$1/3uu_{11}d + 1/6ud_{11}u + 1/2ud_{00}u$
n	$1/6ud_{11}d + 1/2ud_{00}d + 1/3dd_{11}u$
Σ^+	$1/3uu_{11}s + 1/6us_{11}u + 1/2us_{00}u$
Σ^0	$1/3ud_{11}s + 1/12us_{11}d + 1/4us_{00}d + 1/12ds_{11}u + 1/4ds_{00}u$
Σ^-	$1/3dd_{11}s + 1/6ds_{11}d + 1/2ds_{00}d$
Ξ^-	$1/6ds_{11}s + 1/2ds_{00}s + 1/3ss_{11}d$
Ξ^0	$1/6us_{11}s + 1/2us_{00}s + 1/3ss_{11}u$
Λ^0	$1/3ud_{00}s + 1/4us_{11}d + 1/12us_{00}d + 1/4ds_{11}u + 1/12ds_{00}u$
Δ^{++}	$uu_{11}u$
Δ^+	$1/3uu_{11}d + 2/3ud_{11}u$
Δ^0	$2/3ud_{11}d + 1/3dd_{11}u$
Δ^-	$dd_{11}d$
Σ^{*+}	$1/3uu_{11}s + 2/3us_{11}u$
Σ^{*0}	$1/3ud_{11}s + 1/3us_{11}d + 1/3ds_{11}u$
Σ^{*-}	$1/3dd_{11}s + 2/3ds_{11}d$
Ξ^{*0}	$1/3us_{11}s + 2/3ss_{11}u$
Ξ^{*-}	$2/3ds_{11}s + 1/3ss_{11}d$
Ω^-	$ss_{11}s$

Table 6.1: Baryon quark contents. Diquark indices indicate spin-isospin states.

from $SU(6)$ symmetric baryon wave functions. These probabilities are given below in the Table 6.1.

We can also use these probabilities to sample baryon or baryon resonance in the course of a string fragmentation, assuming that a valence diquark (antidiquark) keeps its spin and isospin values during the reaction.

6.1.3 Gamma conversion

To calculate high-energy interaction of the real or virtual photon with a nucleon or a nucleus, we convert it into a vector meson or meson type system [5].

We consider the following kinematics variables for γ-nucleon scattering: the Bjorken's x variable is defined as $x = Q^2/2m\nu$ with Q^2, ν and m are photon virtuality, photon energy and nucleon mass, respectively. The total energy squared of the γ-nucleon system is given by $s = Q^2(1-x)/x + m^2$. We restrict our consideration to the range of small x-values and Q^2 is much less than s.

The Generalized Vector Dominance Model (GVDM) [6] assumes that a virtual photon fluctuates into intermediate $q\bar{q}$-state V of mass M that may subsequently interact with a nucleon N. Thus, the total photon-nucleon cross section can be expressed by the relation [5]:

$$\sigma_{\gamma N}(s,Q^2) = 4\pi\alpha_{em}\int_{M_0^2}^{M_1^2} dM^2 D(M^2)\left(\frac{M^2}{M^2+Q^2}\right)^2\left(1+\epsilon\frac{Q^2}{M^2}\right)\sigma_{VN}(s,Q^2), \tag{6.2}$$

where the integration over M^2 is performed between $M_0^2 = 4m_\pi^2$ and $M^2 = s$. Here $\alpha_{em} = e^2/4\pi = 1/137$ and $q\bar{q}$-system mass squared distribution is given by

$$D(M^2) = \frac{R_{e^+e^-}(M^2)}{12\pi^2 M^2} \tag{6.3}$$

and

$$R_{e^+e^-}(M^2) = \frac{\sigma_{e^+e^-\to hadrons}(M^2)}{\sigma_{e^+e^-\to\mu^+\mu^-}(M^2)} \approx 3\Sigma_f e_f^2, \tag{6.4}$$

where e_f^2 is the squared charge of a quark with the flavor f. ϵ is the ratio between the fluxes of longitudinally and transversally polarized photons.

The Monte Carlo procedure of a gamma conversion can be outlined as follows:

- At given s and virtuality Q^2 sample the mass M^2 for a hadron $q\bar{q}$ fluctuation according to the Eq. (6.3);
- Sample its flavor according to the statistical weights: $\omega_{u\bar{u}} = 1/2$, $\omega_{d\bar{d}} = 1/4$ and $\omega_{s\bar{s}} = 1/4$ that are derived from the Eq. (6.4).

Bibliography

[1] Particle-Data-Group, Phys. Rev. **D45** 1342 (1992).

[2] Particle-Data-Group, Phys. Rev. **D54** 1 (1996).

[3] Particle-Data-Group 98, Eur. Phys. J. **C3** 1 (1998).

[4] Particle-Data-Group 2000, Eur. Phys. J. **C15** 1 (2000).

[5] Piller G., Ratzka W., Weise W., Z. Phys. **A352** 427 (1995).

[6] Bauer T. H., Spital R. D., Yennie D. R., Rev. Mod. Phys. **50** 261 (1978).

6.2 Hadronic cross section components

The hadron interaction cross sections are functions of the types of colliding particles and their c. m. energy $\sqrt{s_{i,j}} = \sqrt{(p_i + p_j)^2}$, where $p_i = (E_i, \mathbf{p_i})$ and $p_j = (E_j, \mathbf{p_j})$ denote four momenta of the projectile particle i with mass $m_i = \sqrt{E_i^2 - \mathbf{p_i}^2}$ and target particle j with mass $m_j = \sqrt{E_j^2 - \mathbf{p_j}^2}$, respectively. The hadron interaction cross sections can either be tabulated or parameterized according to algebraic functions. The hadron interaction cross sections can be estimated from the additive quark model and Regge theory. We would note that some unknown cross sections can be extracted from other cross sections via general principles, such as the isotopic invariance and detailed balance [1].

6.2.1 Additive quark model

In the additive quark model (AQM) cross section depends only on the quark content of colliding hadrons [2], [3], [4]:

$$\sigma_{tot}^{AQM} = 40(\frac{2}{3})^{n_M}(1 - 0.4x_1^s)(1 - 0.4x_2^s) \tag{6.5}$$

and

$$\sigma_{el}^{AQM} = 0.039\sigma_{tot}^{\frac{2}{3}}, \tag{6.6}$$

where n_M is the number of colliding mesons and x_i^s is the ratio of strange quarks to non-strange quarks in the i-th hadron. Thus, we can use the AQM to obtain unknown cross sections by the scaling of known cross sections.

6.2.2 Pomeron eikonal model

The hadron-hadron cross sections at given c.m. energy squared s can be calculated from the interaction probabilities $p_{ij}(b_{ij}^2, s)$ by integration over impact parameter b. Particularly, the total and inelastic cross sections are calculated according to

$$\sigma_{tot} = 2\pi \int_0^\infty bdbp_{ij}^{tot}(b_{ij}^2, s) = \sigma_P f(\frac{z}{2}) \tag{6.7}$$

and

$$\sigma_{in} = 2\pi \int_0^\infty bdbp_{ij}^{in}(b_{ij}^2, s) = \sigma_P f(z), \tag{6.8}$$

where

$$\sigma_P = 4\pi z(s)\lambda(s), \tag{6.9}$$

and

$$f(z) = \sum_{\nu=1}^{\infty} \frac{(-z)^{\nu-1}}{\nu\nu!}. \tag{6.10}$$

In the Regge-Gribov approach [5], [6] the probability for an inelastic collision of hadrons i and j as a function of the squared impact parameter difference $b_{ij}^2 = (\vec{b}_i - \vec{b}_j)^2$ and s is given by

$$p_{ij}(\vec{b}_i - \vec{b}_j, s) = c^{-1}[1 - \exp\{-2u(b_{ij}^2, s)\}] = \sum_{n=1}^{\infty} p_{ij}^{(n)}(\vec{b}_i - \vec{b}_j, s), \tag{6.11}$$

where

$$p_{ij}^{(n)}(\vec{b}_i - \vec{b}_j, s) = c^{-1} \exp\{-2u(b_{ij}^2, s)\} \frac{[2u(b_{ij}^2, s)]^n}{n!}. \tag{6.12}$$

is the probability to find n cut Pomerons or the probability for $2n$ strings produced in the inelastic hadron-hadron collision, if we assume that each cut Pomeron can be substituted by two strings. These probabilities are defined in terms of the amplitude (eikonal) of hadron-hadron elastic scattering with Pomeron exchange:

$$u(b_{ij}^2, s) = \frac{z(s)}{2} \exp[-b_{ij}^2/4\lambda(s)]. \tag{6.13}$$

The quantities $z(s)$ and $\lambda(s)$ are expressed through the parameters of Pomeron trajectory $\alpha_P' = 0.25\ GeV^{-2}$ and $\alpha_P(0) = 1.0808$, and the parameters of Pomeron-hadron vertex R_P^2 and γ_P:

$$z(s) = \frac{2c\gamma_P}{\lambda(s)} (s/s_0)^{\alpha_P(0)-1} \tag{6.14}$$

$$\lambda(s) = R_P^2 + \alpha_P' \ln(s/s_0), \tag{6.15}$$

respectively, where s_0 is the dimensional parameter.

In the equations (6.11,6.12) the so-called shower enhancement coefficient c was introduced to determine the contribution of diffractive dissociation [6]. Thus, the probability for diffractive dissociation in hadron collision can be computed as

$$p_{ij}^d(\vec{b}_i - \vec{b}_j, s) = \frac{c-1}{c}[p_{ij}^{tot}(\vec{b}_i - \vec{b}_j, s) - p_{ij}(\vec{b}_i - \vec{b}_j, s)], \tag{6.16}$$

where

$$p_{ij}^{tot}(\vec{b}_i - \vec{b}_j, s) = (2/c)[1 - \exp\{-u(b_{ij}^2, s)\}]. \tag{6.17}$$

The Pomeron vertex parameters are found from a global fit of the total, elastic, differential elastic and diffractive cross sections of the hadron-nucleon interaction at different energies.

For the nucleon-nucleon, pion-nucleon and kaon-nucleon collisions we have obtained the next values of the Pomeron vertex parameters and shower enhancement coefficients: $R_P^{2N} = 3.56\ GeV^{-2}$, $\gamma_P^N = 3.96\ GeV^{-2}$, $s_0^N = 3.0\ GeV^2$, $s_0^N = 1.4$; $R_P^{2\pi} = 2.36\ GeV^{-2}$, $\gamma_P^\pi = 2.17\ GeV^{-2}$, $s_0^\pi = 1.5\ GeV^2$, $c^\pi = 1.6$; $R_P^{2K} = 1.96\ GeV^{-2}$, $\gamma_P^K = 1.92\ GeV^{-2}$, $s_0^K = 2.3\ GeV^2$, $c^\pi = 1.8$. These values can be used to describe properly the total, elastic and diffractive cross sections.

6.2.3 Single diffraction cross section

The proton-proton single diffraction cross section was parameterized in paper [7]

$$\sigma_{SD}(s) = (0.68 \pm 0.05)\left(1 + \frac{36 \pm 8}{s}\right)\ln(0.6 + 0.1s), \tag{6.18}$$

where s is the total c.m. energy squared.

6.2.4 Baryon annihilation cross section

To calculate the baryon-antibaryon annihilation cross sections we follow the approach [4]. In this approach the same initial energy dependence is used for all baryon-antibaryon cross sections. The different baryon quark contents are taken into account by scaling factor obtained from the AQM [2], [3]:

$$\sigma_{\bar{B}B}(\sqrt{s}) = \frac{\sigma_{BB}^{AQM}}{\sigma_{NN}^{AQM}}\sigma_{\bar{N}N}(\sqrt{s}), \tag{6.19}$$

where the antiproton-proton annihilation cross section $\sigma_{\bar{p}p}^{ann}$ is parametrised [8] by

$$\sigma_{\bar{p}p}^{ann}(s) = \sigma_0\frac{s_0}{s}\left[\frac{A^2 s_0}{(s - s_0)^2 + A^2 s_0} + B\right] \tag{6.20}$$

and the antiproton-neutron cross annihilation cross section is treated identically. The parameter values are $\sigma_0 = 120$ mb, $s_0 = 4m_N^2$, $A = 50$ MeV and $B = 0.6$.

Bibliography

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

[2] Levin E. M., Frankfurt L. L., Pisma ZhETP **3** 105 (1965).

[3] Lipkin H. J., Sheck F., Phys. Rev. Lett. **16** 71 (1966).

[4] Bass S. A. *et al.*, Prog. Part. Nucl. Phys. **V. 41** 225 (1998).

[5] Abramovskii V. A., Gribov V. N., Kancheli O. V., Sov. J. Nucl. Phys. **18** (1974) 308.

[6] Baker M. and Ter–Martirosyan K. A., Phys. Rep. **28C** (1976) 1.

[7] Goulianos K., Phys. Rep. **101** 169 (1983).

[8] Koch P., Dover C. B., Phys. Rev. **C40** 145 (1989).

6.3 Particle decay components

6.3.1 Particle decay simulation

Particle decay channels are sampled using the partial decay widths $\Gamma(m)$, which depend on the masses m of particles. All particles are considered as non-polarized particles. If a resonance is among of the outgoing particles, its mass is determined according to the Breit-Wigner mass distribution. All pole masses and partial decay widths are taken from the Review of Particle Properties [1]. If an exit channel contains two, three or four particles, then the respective N-body phase-space is taken into account during the sampling of particle momenta.

The Kopylov's algorithm [2] is used to sample product particle 4-momenta. We would stress that the Kopylov's algorithm gives a possibility to take into account resonance decay matrix elements.

6.3.2 N-body decay algorithm

We can write the n particles state probability as

$$dW = \prod_{i=1}^{n} \frac{d^3\mathbf{p}_i}{2\omega_i} \delta(\sum_{i=1}^{n} p_i - P_n)|M|^2, \tag{6.21}$$

where $P_n = (\mathbf{P}_n, E_n)$ is the decaying particle 4-momentum and $M(p_1, p_2, ..., p_n)$ is the reaction amplitude, which depends on the 4-momenta p_i of products.

We know the sum of produced particle masses

$$\mu_n = \sum_{i=1}^{n} m_i \tag{6.22}$$

and c.m. system kinetic energy

$$T_n = M_n - \mu_n, \tag{6.23}$$

where $M_n = \sqrt{E_n^2 - P_n^2}$ is the invariant mass of the system.

Performing integration of the probability expressed by the (6.21), one can obtain different particle distributions. The integration over all set of variables S can shortly be written as follows

$$W = \int_D \Phi(S) dS, \tag{6.24}$$

where $\Phi(S)$ can be considered as the weight of n-particles production event.

The Kopylov's algorithm is based on the choice of a set of variables S_1: T_k, η_k and ϕ_k instead of particle 4-momenta to perform the integration of dW. The integration range D_1 over these new variables is simple. Angular variables are inside $1 \geq \eta_k \geq -1$ and $2\pi\phi_k \geq 0$, where $k = n, n-1, ..., 2$, and kinetic energies of the systems with some numbers of particles are ordered according to $T_n \geq T_{n-1} \geq ... \geq T_2 \geq T_1 = 0$. Further substitution of the variables T_k, η_k and ϕ_k by the set variables S_2: ξ_k, γ_k and β_k that are limited between 0 and 1, creates an unit hypercube from the integration range D_2:

$$W \sim \int_0^1 ... \int_0^1 \prod_2^n d\beta_k d\gamma_k \prod_2^{n-1} d\xi_k \xi_k^{\frac{3k-5}{2}} \sqrt{1-\xi_k} |M|^2. \tag{6.25}$$

Two sets of variables S_1 and S_2 are related to each other:

$$\begin{gathered} T_k = T_{k+1}\xi_k, k = n-1, ..., 2 \\ \eta_k = 2\beta_k - 1, k = n, ..., 2 \\ \phi_k = 2\pi\gamma_k, k = n, ..., 2. \end{gathered} \tag{6.26}$$

The event weight $\Phi(S_1)$ [2] is

$$\Phi(S_1) = \frac{\pi^{\frac{3}{2}(n-1)} T_n^{\frac{3n-5}{2}}}{2M_n \Gamma(\frac{3}{2}(n-1))} \prod_{k=2}^{n} \frac{\hat{p}_k}{\sqrt{T_k - T_{k-1}}} |M|^2, \tag{6.27}$$

where $\Gamma(x)$ is the gamma function and $\hat{p}_k$ is the 4-momentum of k-th particle in the rest frame, 0_k, where $T_k = 0$.

As one can see from the Eq. (6.25) the variables ξ_k, γ_k and β_k can be considered as random numbers distributed between 0 and 1. Thus, to generate particle momenta we can apply the next recursive procedure [2].

1. Sample ξ_{k-1} according to the distribution

$$F(\xi_k) = \xi_k^{\frac{3k-5}{2}} \sqrt{1-\xi_k}, \tag{6.28}$$

where functions $F(\xi_k)$ have maxima $F_{max} = F((3k-5)/(3k-4))$, and compute $T_{k-1} = T_k \xi_{k-1}$. If $k = 2$, then T_{k-1} should be set to 0 in accordance with the definition of the rest frame 0_{k-1}.

The functions $F(x)$ are sharp functions at large values of k. At the large k we can use another method of the sampling with a tabulation [2]. In this case for uniformly distributed between 0 and 1 random numbers α_{k-1} we have to solve the equation

$$C_{k-1}(\xi_{k-1}) = \alpha_{k-1}, \tag{6.29}$$

where

$$C_k(\xi_k) = \frac{\int_0^{\xi_k} d\xi_k \xi_k^{\frac{3k-5}{2}} \sqrt{1-\xi_k}}{\int_0^1 d\xi_k \xi_k^{\frac{3k-5}{2}} \sqrt{1-\xi_k}} \tag{6.30}$$

and find the value of ξ_{k-1}. $C_k(\xi_k)$ are tabulated smooth functions. A linear interpolation can be applied to find the function values at the intermediate points.

2. Compute the total energy

$$\hat{\omega}_k = (M_k^2 + m_k^2 - M_{k-1}^2)/2M_k \tag{6.31}$$

and absolute value of 3-momentum

$$\hat{p}_k = \sqrt{\hat{\omega}_k^2 - m_k^2} \tag{6.32}$$

for k-th particle in the rest frame, 0_k, where

$$\mu_{k-1} = \mu_k - m_k \tag{6.33}$$

and

$$M_{k-1} = \mu_{k-1} - T_{k-1}. \tag{6.34}$$

3. Sample the direction of k-th momentum $\hat{\mathbf{p}}_k$, i.e., sample the azimuthal and polar angles according to $\phi_k = 2\pi\gamma_k$ and $\cos\theta_k = 2\beta_k - 1$, respectively. The γ_k and β_k are uniformly distributed between 0 and 1 random numbers.

4. Calculate the k-th particle energy

$$\omega_k = \frac{E_k\hat{\omega}_k + \mathbf{P}_k\hat{\mathbf{p}}_k}{M_k} \tag{6.35}$$

and its 3-momentum

$$\mathbf{p}_k = \hat{\mathbf{p}}_k + \hat{\mathbf{P}}_k \frac{\omega_k + \hat{\omega}_k}{E_k + M_k} \tag{6.36}$$

in the observer frame.

5. Calculate the total energy E_{k-1} and 3-momentum $\mathbf{P}_{k-1}$ of the system with $k-1$ particles:

$$E_{k-1} = E_k - \omega_k, \tag{6.37}$$

$$\mathbf{P}_{k-1} = \mathbf{P}_k - \mathbf{p}_k. \tag{6.38}$$

If $k = 2$, then we can check that $m_1 = \sqrt{E_1^2 - P_1^2}$, where $\mathbf{p}_1 = \mathbf{P}_1$ and $\omega_1 = E_1$.

As result of the outlined recursive procedure we know the momenta of all products and can calculate the event weight (6.27).

Finally, if $\Phi(S_1) < \xi\Phi_{max}$, where ξ is a uniformly distributed between 0 and 1 random number and Φ_{max} is a maximal value from the $\Phi(S_1)$ set of values, we will accept the event.

We should note that in the non-relativistic case and for $|M|^2 = 1$ the last step, i.e., the test $\Phi(S_1) < \xi\Phi_{max}$, is not needed [2].

We also should note that the outlined procedure can be used to calculate the relativistic phase space volume W, i.e., to perform an integration over the event weight (6.24) at $|M|^2 = 1$. To perform this integration we need to find an average value of the event weights for N events, where $N \to \infty$. This method to perform the integration is also efficient at $|M|^2 \neq 1$, if $|M|^2$ has no pole behavior. Thus, the Kopylov's algorithm can be used to calculate decay widths and different particle distributions.

It also allows us to simulate the nuclear fragment production as result of the decay of excited nuclei.

Bibliography

[1] Particle-Data-Group, Phys. Rev. **D54** 1 (1996).

[2] Kopylov G. I, The basics of resonance kinematics, Nauka, Moscow, 1970.

6.4 String decay components

A colored string is stretched between flying away partons having color charges: a quark and antiquark or quark and diquark or diquark and antidiquark or antiquark and antidiquark. From knowledge of the parton longitudinal $p_{3i} = p_{zi}$ and transversal $p_{1i} = p_{xi}$, $p_{2i} = p_{yi}$ momenta as well as their energies $p_{0i} = E_i$, where $i = 1, 2$, we can calculate the string (string with parton ends) mass squared:

$$M_S^2 = p^\mu p_\mu = p_0^2 - p_1^2 - p_2^2 - p_3^2, \tag{6.39}$$

where $p_\mu = p_{\mu 1} + p_{\mu 2}$ is the string four momentum and $\mu = 0, 1, 2, 3$.

The string decay model components can be used to simulate decays of the longitudinal $q - \bar{q}$, $qq - \bar{q}\bar{q}$, $q - qq$ and $\bar{q} - \bar{q}\bar{q}$ strings or kinky $q - g - \bar{q}$, $qq - g - \bar{q}\bar{q}$, $q - g - qq$ and $\bar{q} - g - \bar{q}\bar{q}$ strings.

It is convenient to perform a simulation of the string decay in the string rest frame, with string end partons are moving along z-axis. Finally, after the string decay we have to perform a backward Lorentz boost with the velocity $-\beta_\mu$ of hadron 4-momenta and their 4-coordinates and a backward rotation of hadron 3-momenta and 3-coordinates with the matrix R_{kl}^{-1} (see below).

6.4.1 Cluster decay simulation

The procedure to model a cluster (small mass string) decay is suggested in the papers [1], [2]. It is assumed that each cluster with mass M_c decays isotropically in its rest frame into a hadron pair with masses M_1 and M_2 and spins J_1 and J_2 by pulling a quark-antiquark or diquark-antidiquark pair with the quark (diquark) mass m_q. For a hadron, which is containing u, d and s quarks, the $J^P = 0^-, 1^\pm$ meson and $J^P = \frac{1}{2}^+, \frac{3}{2}^+$ baryon states are allowed. A decay channel is selected according to the probability:

$$P_{decay} = P_{flavor}(M_c, m_q) P_{spin}(J_1, J_2) P_{ps}(M_c, M_1, M_2), \tag{6.40}$$

where the flavor factor is

$$P_{flavor}(M_c, m_q) = 1 + \frac{2m_q^2}{M_c^2} \sqrt{1 - \frac{4m_q^2}{M_c^2}}, \tag{6.41}$$

spin factor is

$$P_{spin}(J_1, J_2) = (2J_1 + 1)(2J_2 + 1) \tag{6.42}$$

and phase space factor is

$$P_{ps} = \frac{\sqrt{M_c^4 + M_1^4 + M_2^4 - 2(M_c^2 M_1^2 + M_c^2 M_2^2 + M_1^2 M_2^2)}}{M_c^2}. \tag{6.43}$$

6.4.2 Longitudinal string decay

The decay of a string follows an iterative scheme:

$$string \Rightarrow hadron + new\ string, \tag{6.44}$$

i. e., the string decay into a new string and hadron is modeled by means of the vacuum quark-antiquark (or diquark-antidiquark) pair creation. The values of strangeness suppression and diquark suppression factors are

$$u : d : s : qq = 1 : 1 : 0.35 : 0.1. \tag{6.45}$$

Optionally, a possibility of a diquark breaking during the string fragmentation can be considered.

A hadron is formed randomly on one of the end-point of the string. A quark content of the hadrons determines its species and charges. In the chosen fragmentation scheme we can produce not only the ground states of baryons and mesons, but also their lower excited states. If for baryons, the quark content does not determine whether the state belongs to the lowest octet or to the lowest decuplet, then the octet or decuplet is chosen with the probability that defined by spin and isospin of the diquark (antidiquark). In the case of mesons the multiplet must be also determined before a type of hadron can be assigned. The probability to choose a certain multiplet depends on the spin of the multiplet. In the case of resonances their masses m are determined according to the Breit-Wigner distributions, where all pole masses and total decay widths are taken from the Review of Particle Properties [3].

Zero transverse momentum of the created quark-antiquark (or diquark-antidiquark) pair is obtained by summing of equal and opposite directed transverse momenta of the quark and antiquark. A transverse momentum of the quark is randomly sampled according to the probability (6.85) with the parameter $a = 0.55$ GeV^{-2}. The produced hadron transverse momentum $\mathbf{p_t}$ is a sum of the transverse momenta of its constituents.

The fragmentation function $f^h(z, p_t)$ represents a probability distribution for hadrons with the transverse momenta $\mathbf{p_t}$ to carry a light cone momentum fraction $z = z^{\pm} = (E^h \pm p_z^h)/(E^q \pm p_z^q)$, where E^h and E^q are the hadron and fragmented quark energies, respectively. p_z^h and p_z^q are the hadron and fragmented quark longitudinal momenta, respectively. The values of z are limited between $z_{min}^{\pm}$ and $z_{max}^{\pm}$. $z_{min,max}^{\pm}$ are determined by the hadron mass m_h, constituent transverse masses and available string mass. One of the most commonly used fragmentation function is taken from the LUND model [4]:

$$f^h(z, p_t) \sim \frac{1}{z}(1-z)^a \exp\left[-\frac{b(m_h^2 + p_t^2)}{z}\right]. \tag{6.46}$$

One can use this fragmentation function for the decay of an excited string.

One can also use fragmentation functions are derived in [5]:

$$f_q^h(z, p_t) = [1 + \alpha_q^h(< p_t >)](1 - z)^{\alpha_q^h(<p_t>)}. \tag{6.47}$$

An advantage of these functions as compared to the LUND fragmentation function is that they have a correct three-reggeon behavior at $z \to 1$ [5].

To calculate produced hadron formation times and longitudinal coordinates we consider $(1+1)$-string with the mass M_S and string tension κ that decays into hadrons in its rest frame. The i-th produced hadron has the energy E_i and its longitudinal momentum p_{zi}, respectively. By introducing the light cone variables $p_i^{\pm} = E_i \pm p_{iz}$ and numerating string breaking points consecutively from the right to the left we obtain $p_0^+ = M_S$, $p_i^+ = \kappa(z_{i-1}^+ - z_i^+)$ and $p_i^- = \kappa x_i^-$.

We can identify a hadron formation point coordinate and time as the point in space-time, where quark lines of a quark-antiquark pair that is forming the hadron meet for the first time (the so-called "yo-yo" formation point [4]):

$$t_i = \frac{1}{2\kappa}[M_S - 2\sum_{j=1}^{i-1} p_{zj} + E_i - p_{zi}] \tag{6.48}$$

and coordinate

$$z_i = \frac{1}{2\kappa}[M_S - 2\sum_{j=1}^{i-1} E_j + p_{zi} - E_i]. \tag{6.49}$$

6.4.3 Kinky string decay simulation

For the kinky string decay simulation we have assumed a two steps process [6]:

1. Split the gluon $g \to q_1\bar{q}_1$ and create two longitudinal strings;
2. Decay of the longitudunal strings $q\bar{q}_1 \to h$ and $\bar{q}q_1 \to h$ into hadrons h.

The production of $q_1\bar{q}_1$ is considered similarly (the same sampling of quark flavors and the same p_t-distribution for the quarks) as the production of $q\bar{q}$-pairs during a longitudinal string decay. The $g \to q_1\bar{q}_1$ splitting function is

$$f_g^q(z) = z^2 + (1 - z)^2, \tag{6.50}$$

where $z = \frac{E_q + p_q^z}{E_g + p_g^z}$, have been derived by Altarelli and Parisi [7].

6.4.4 Transformation of a string

The simulation of string decay is considered in the frame of a string rest. The string end partons are moving along z-axis in the opposite directions. We can perform Lorentz transformation of the parton momenta to the c.m. of the string:

$$p_{\mu 1,\mu 2} \rightarrow L_\mu p_{\mu 1,\mu 2}, \tag{6.51}$$

where

$$L_0 = \beta^\nu p_\nu \tag{6.52}$$

and

$$L_k = \beta_k p_0 + \sum_{l=1}^{3} (\delta_{lk} + \frac{\beta_k \beta_l}{1+\beta_0}) p_l. \tag{6.53}$$

β_μ is defined as follows:

$$\beta_\mu = \frac{p_\mu}{M_S}. \tag{6.54}$$

The string orientation respecting to z-axis is determined by two Euler angles α and β, which can be calculated according to

$$\cos\alpha = \frac{p_{32}}{\sqrt{p_{22}^2 + p_{32}^2}} \tag{6.55}$$

and

$$\cos\beta = \frac{\sqrt{p_{22}^2 + p_{32}^2}}{\sqrt{p_{12}^2 + p_{22}^2 + p_{32}^2}}. \tag{6.56}$$

Then, by a string rotation:

$$p_{k1,k2} \rightarrow R_{kl} p_{l1,l2}, \tag{6.57}$$

we can obtain a motion of the constituents along z-axis. Here the matrix R_{kl} is given by

$$R_{kl} = \begin{vmatrix} \cos\beta & -\sin\alpha\cos\beta & -\cos\alpha\sin\beta \\ 0 & \cos\alpha & -\sin\alpha \\ \sin\beta & -\sin\alpha\cos\beta & -\cos\alpha\cos\beta \end{vmatrix} \tag{6.58}$$

and $k, l = 1, 2, 3$.

Bibliography

[1] Gottschalk T. D., Nucl. Phys. **B214** 201 (1983); **B227** 413 (1983).

[2] Weber B. R., Nucl. Phys. **B238** 492 (1984).

[3] Particle-Data-Group, Phys. Rev. **D54** 1 (1996).

[4] Andersson B., Gustafson G., Ingelman G., Sjöstrand T., Phys. Rep. **97** 31 (1983).

[5] Kaidalov A. B., Sov. J. Nucl. Phys. **45** 1452 (1987).

[6] Ali A, Pietarinen E., Kramer G., Willrodt J, Phys. Lett. **B93** 155 (1980).

[7] Altarelli G. and Parisi G., Nucl. Phys. **B126** 298 (1977).

6.5 Elastic scattering components

6.5.1 Hadron elastic scattering

The hadron elastic scattering model is capable to predict final states of the hadron elastic collisions.

An elastic angular distribution can be obtained from the Pomeron eikonal model [1]. It gives a possibility to sample scattering angles at initial energies $\sqrt{s} > 2$ GeV. At lower energies the sampling of scattering angles are based on a parameterization of experimental data for nucleon-nucleon interactions.

Angular distributions of the two-body hadron elastic scatterings at high energies are well described by

$$\frac{dW(s)}{dt} = B_{el}(s) \exp\left[B_{el}(s)t\right], \tag{6.59}$$

where $t = -2p_{cm}^2(1 - \cos\theta)$ is the four momentum transfer that is expressed through the particle center of masses momentum p_{cm} and particle center of masses scattering angle θ. The values of t are limited in the interval $t_{min} \leq t \leq t_{max}$, where $t_{max} = 0$ and $t_{min} = -4p_{cm}^2$. The energy dependence of the slope parameter $B_{el}(s)$ is defined in the Regge-Gribov approach [1] as follows

$$B_{el}(s) = 2\lambda(s), \tag{6.60}$$

where

$$\lambda(s) = R_P^2 + \alpha_P^{'} \ln(s/s_0). \tag{6.61}$$

As it was explained (see the hadron cross section section) the Pomeron parameters R_P^2, $\alpha_P^{'}$ and s_0 can be found from a global fit of the total, elastic, differential elastic and diffractive cross sections for nucleon-nucleon, pion-nucleon and kaon-nucleon interactions at the different values of s.

To perform elastic scattering simulations we use the Eq. (6.59) to sample t. Then we calculate the scattering angle θ. The Eq. (6.60) is used, when the initial energy $\sqrt{s} > \sqrt{s_{th}}$, where the threshold energy $\sqrt{s_{th}} = 1.6$ GeV for pion-nucleon collisions, $\sqrt{s_{th}} = 2.0$ GeV for kaon-nucleon collisions and $\sqrt{s_{th}} = 2.5$ GeV for nucleon-nucleon collisions, respectively.

To sample $\cos\theta$ at lower energies we use the parameterization:

$$B_{el}(s) = \frac{2.15R_P^2[3.65(\sqrt{s} - m_1 - m_2)]^6}{1 + [3.65(\sqrt{s} - m_1 - m_2)]^6}, \tag{6.62}$$

where m_1 and m_2 are masses of colliding particles. This form of the angular distribution was obtained by a modification of the parameterization [2] to make a smooth transition to the high energy region described by the Regge motivated slopes.

6.5.2 Gluon elastic scattering

The gluon-gluon elastic scattering model is used to simulate hadron jets or kinky strings production in inelastic hadron and nuclear collisions.

The generation of the scattered gluon angles θ for the center of masses of a parton-parton system is based on the QCD gluon-gluon interaction cross section that is

$$\frac{d\sigma_{gg}(\hat{s})}{d\cos\theta} = \frac{9\pi\alpha_s^2(Q^2)}{32s}\frac{(3+\cos^2\theta)^3}{(1-\cos^2\theta)^2} \tag{6.63}$$

calculated in the Born approximation [3], where $\alpha_s(Q^2)$ is the QCD running coupling constant. The value of $\hat{s} = Q^2$ should be large enough to produce gluons with a transverse momentum squared above the chosen cutoff $Q_0^2 = 2\ GeV^2$. Due to this condition we have $-z_0 < \cos\theta < z_0$, where

$$z_0 = \sqrt{1 - \frac{4Q_0^2}{\hat{s}}}. \tag{6.64}$$

A scattering angle can be expressed through the transferred momentum squared $\hat{t}$:

$$\cos\theta = 1 + \frac{2\hat{s}\hat{t}}{[\hat{s} - (m_i + m_j)^2)(\hat{s} - (m_i - m_j)^2)]}, \tag{6.65}$$

where m_i and m_j refer initial parton masses. As a rule, a sampling of $\hat{t}$ is much more simple than sampling of the $\cos\theta$.

6.5.3 Elastic scattering MC procedure

We use the following MC procedure to sample elastic scattering angles $\cos\theta$:

1. Sample transferred momentum squared according to

$$t = \frac{1}{B_{el}(s)}\ln\xi, \tag{6.66}$$

where ξ are random numbers that are uniformly distributed between 0 and 1, $t_{min} \leq t \leq t_{max}$;

2. Calculate

$$\cos\theta = 1 - \frac{t}{2p_{cm}^2}. \tag{6.67}$$

Bibliography

[1] Baker M. and Ter-Martirosyan K. A., Phys. Rep. **28C** 1 (1976).

[2] Cugnon J., Mizutani T., Vandermeulen J., Nucl. Phys. **A352** 505 (1981).

[3] Combrige B. L., Kripfganz J., Ranft J., Phys. Lett. **70B** 234 (1977); Cutler R., Sivers D., Phys. Rev. **D17** 196 (1978).

6.6 Particle annihilation components

The particle annihilation components allow their users to model a special hadron-hadron inelastic process, when an antiquark (antdiquark) from one colliding hadron annihilates with a suitable quark (diquark) from another colliding hadron. In this case the excitation of a baryon (a string with quark and diquark ends) or antibaryon (a string with antiquark and antidiquark ends) or meson (a string with quark and antiquark ends) string is occurred. Particularly, the baryon annihilation model component is able to predict final states (produced hadrons) with zero net baryon number in a baryon-antibaryon collision at different energies. Here, a baryon annihilation process is considered in the diquark-antidiquark annihilation approximation. The produced hadrons belong to the scalar and vector meson nonets and the baryon (antibaryon) octet and decuplet.

6.6.1 String excitation by a quark annihilation

These processes in the Regge theory correspond to the cut reggeon exchange diagrams. The initial energy $\sqrt{s}$ dependences of process cross sections are defined by the intercepts of reggeon exchange trajectories. For example, $\sigma_{\pi^+p\to S(s)} \sim s^{\alpha_\rho(0)-1}$, S denotes a string and $\alpha_\rho(0)$ is the intercept of ρ reggeon trajectory. Thus, $\sigma_{\pi^+p\to S(s)}$ decreases with the energy rise. By an analysis of other similar diagrams we can show that the quark annihilation processes are important at relatively low initial energies.

A simulation of these processes is rather simple. We should randomly (according to the weights that are calculated using hadron wave functions) choose a quark (antiquark) from the projectile and find a suitable (with the same flavor content) partner for the annihilation from the target. The created string four-momentum

$$P = P_1 + P_2 \tag{6.68}$$

is approximated by the total reaction four-momentum.

6.6.2 Quark annihilation weight

To determine statistical weights for the quark annihilation processes followed by a string production and separate them from the processes, when two or more strings can be produced, we can use the Regge motivated expression for the hadronic total cross section:

$$\sigma_{tot}(s) = \sum_i \sigma_i(s), \tag{6.69}$$

where $\sigma_i(s)$ is the cross section of i-th subprocess. It can be either the elastic cross section $\sigma_{el}(s)$ or annihilation cross section with one string production $\sigma_S(s)$ or single diffraction cross section $\sigma_D(s)$ or multi-string production cross section $\sigma_{MS}(s)$.

From experimental knowledge of the vtotal and elastic cross sections we can calculate the inelastic cross section:

$$\sigma_{in}(s) = \sigma_{tot}(s) - \sigma_{el}(s) = \sigma_S(s) + \sigma_D(s) + \sigma_{MS}(s). \tag{6.70}$$

The diffraction and multi-string cross sections are calculated according to the Regge eikonal model as it was described above in the chapter devoted to the hadronic cross sections. Thus, the statistical weight for the quark annihilation process can be calculated as follows

$$W_S(s) = \frac{\sigma_{in}(s) - \sigma_D(s) - \sigma_{MS}(s)}{\sigma^{in}(s)}. \tag{6.71}$$

6.6.3 Baryon annihilation weight

The baryon annihilation cross section $\sigma_{ann}(s)$ is used to determine statistical weights for the diquark annihilation process:

$$W_{ann}(s) = \frac{\sigma_{ann}(s)}{\sigma_{tot}(s)}, \tag{6.72}$$

where $\sigma_{tot}(s)$ is the total baryon-antibaryon interaction cross section as a function of the total c.m. energy squared.

The baryon annihilation model can be improved by adding another competing sub-processes like the string junction annihilation, when three meson strings can be produced (see papers [1], [2]).

Bibliography

[1] Volkovitskii P. E., Sov. J. Nucl. Phys. **44** 729 (1986).

[2] Amelin N. S., Bravina L. V., Sarycheva L. I., Sov. J. Nucl. Phys. **50** 1705 (1989).

6.7 Particle inelastic collision components

Here we describe a composite model component that is referred as the Reggeon based Parton String Model (RPSM). This model is devoted to predict hadron-hadron collisions at high energies. It is originated from the approaches described in papers [1], [2], [3]. The hadrons that produced in the course collisions belong to the scalar and vector meson nonets and the baryon (antibaryon) octet and decuplet.

The model can be applied to describe photon-nucleon inelastic collisions.

The allowed bombarding energy in hadron-nucleon and photon-nucleon collisions should be above 2-pions production threshold.

This model can also be used to predict elastic and annihilation hadron-nucleon collisions described in the previous sections.

6.7.1 Particle inelastic collision algorithm

The RPSM algorithm includes a set of the steps should be performed:

- Create projectile and target particles: assign the initial projectile and target particle types, their coordinates in the impact parameter space and momenta.
- Sample a type of the collision: an elastic, annihilation or inelastic.
- In the cases of elastic and annihilation collisions they are modelled as it was described in the corresponding sections.
- In the case of an inelastic collision sample an impact parameter and define a type of the inelastic collision(a diffractive or non-diffractive). Store the total 4-momentum of the collision participants.
- For the non-diffractive inelastic collisions sample a number of longitudinal and kinky strings can be produced.
- Perform the longitudinal and kinky string excitations.
- Perform the simulation of string decays (see string decay component section).
- Correct energies and momenta of produced particles (see utility component section), if it is necessary.

6.7.2 Sampling of the longitudinal strings

For an each pair of hadrons i and j with the chosen impact parameters $\vec{b}_i$ and $\vec{b}_j$ and total c.m. energy squared s one should check whether they interact inelastically or not using the probability:

$$p_{ij}^{in}(\vec{b}_i - \vec{b}_j, s) = p_{ij}(\vec{b}_i - \vec{b}_j, s) + p_{ij}^{d}(\vec{b}_i - \vec{b}_j, s). \tag{6.73}$$

If the interaction will take place, then one has to decide that it is diffractive or nondiffractive with the probabilities:

$$\frac{p_{ij}^{d}(\vec{b}_i - \vec{b}_j, s)}{p_{ij}^{in}(\vec{b}_i - \vec{b}_j, s)} \tag{6.74}$$

and

$$\frac{p_{ij}(\vec{b}_i - \vec{b}_j, s)}{p_{ij}^{in}(\vec{b}_i - \vec{b}_j, s)}. \tag{6.75}$$

In the Regge-Gribov approach [4] the probability to find n cut Pomerons [5] or probability for $2n$ strings produced in an inelastic collision of the nucleons is given by

$$p_{ij}^{(n)}(\vec{b}_i - \vec{b}_j, s) = c^{-1} \exp\{-2u(b_{ij}^2, s)\} \frac{[2u(b_{ij}^2, s)]^n}{n!}. \tag{6.76}$$

It is assumed that two strings substitute an each cut Pomeron. By summing of the $p_{ij}^{(n)}(\vec{b}_i - \vec{b}_j, s)$ we obtain the non-diffractive probability:

$$p_{ij}(\vec{b}_i - \vec{b}_j, s) = \sum_{n=1}^{\infty} p_{ij}^{(n)}(\vec{b}_i^A - \vec{b}_j^B, s). \tag{6.77}$$

This probability and other hadron interaction probabilities are defined in terms of the amplitude (eikonal) for the nucleon-nucleon elastic scattering with Pomeron exchange $u(b_{ij}^2, s)$. The so-called shower enhancement coefficient c is introduced to determine a contribution of the diffractive dissociation [4] (see hadron cross section section).

6.7.3 Sampling of the kinky strings

To estimate the number of kinky strings produced in hard hadron collisions we assume [6], [7] that each cut Pomeron can be substituted either by the two longitudinal strings as result of soft hadron interaction or by the two kinky strings as result of hard hadron interactions.

At the moment it is not completely clear how to make definite choice between the longitudinal and kinky strings to substitute cut Pomerons. One recipe is based on the eikonal model [8], [7], where we sum the soft and hard eikonal parts:

$$u(b_{ij}^2, s) = u_{soft}(b_{ij}^2, s) + u_{hard}(b_{ij}^2, s). \tag{6.78}$$

The soft eikonal part is defined as

$$u_{soft}(b_{ij}^2, s) = \frac{\gamma_{soft}}{\lambda_{soft}(s)}(s/s_0)^{\Delta_{soft}} \exp[-b_{ij}^2/4\lambda_{soft}(s)]. \tag{6.79}$$

The hard part is calculated according to

$$u_{hard}(b_{ij}^2, s) = \frac{\sigma_{jet}}{8\pi\lambda_{hard}(s)}(s/s_0)^{\Delta_{hard}} \exp[-b_{ij}^2/4\lambda_{hard}(s)]. \tag{6.80}$$

The $\sigma_{jet} = 0.027$ mb and $\Delta_{hard} = 0.47$ have been found [8] from the fit of the two-jets experimental cross section [9]. Then from the global fit of the total and elastic cross sections for pp collisions the values of $\gamma_{soft} = 35.5$ mb, $\Delta_{soft} = 0.07$ and $R_{hard}^2 = R_{soft}^2 = 3.56$ GeV^{-2} have been found.

Thus, we can examine each cut Pomeron and substitute it by two kinky strings with the probability

$$P_{hard}(b_{ij}^2, s) = \frac{u_{hard}(b_{ij}^2, s)}{u_{soft}(b_{ij}^2, s) + u_{hard}(b_{ij}^2, s)}. \tag{6.81}$$

6.7.4 Longitudinal string excitation

Let us consider a collision of two hadrons with their c. m. momenta $P_1 = \{E_1^+, m_1^2/E_1^+, \mathbf{0}\}$ and $P_2 = \{E_2^-, m_2^2/E_2^-, \mathbf{0}\}$, where the light-cone variables $E_{1,2}^{\pm} = E_{1,2} \pm P_{z1,2}$ are defined through hadron energies $E_{1,2} = \sqrt{m_{1,2}^2 + P_{z1,2}^2}$, hadron longitudinal momenta $P_{z1,2}$ and hadron masses $m_{1,2}$, respectively. We assume that two hadrons collide by means of two partons with the momenta $p_1 = \{x^+E_1^+, 0, \mathbf{0}\}$ and $p_2 = \{0, x^-E_2^-, \mathbf{0}\}$, respectively.

Diffractive string excitation

In the diffractive string excitation (Fritiof approach [10]) only momentum can be transferred:

$$\begin{aligned} P_1' &= P_1 + q \\ P_2' &= P_2 - q, \end{aligned} \tag{6.82}$$

where

$$q = \{-q_t^2/(x^-E_2^-), q_t^2/(x^+E_1^+), \mathbf{q_t}\} \tag{6.83}$$

is parton momentum transferred and $\mathbf{q_t}$ is its transverse component.

String excitation by a parton exchange

For this case (the QGSM approach [1]) the parton exchange (rearrangement) and momentum exchange are allowed [1],[2],[3]:

$$\begin{aligned} P_1' &= P_1 - p_1 + p_2 + q \\ P_2' &= P_2 + p_1 - p_2 - q, \end{aligned} \tag{6.84}$$

where $q = \{0, 0, \mathbf{q_t}\}$ is parton momentum transferred.

Transversal and longitudinal momenta generation

The transverse component of a parton momentum transferred is sampled according to the probability

$$P(\mathbf{q_t})d\mathbf{q_t} = \sqrt{\frac{a}{\pi}} \exp\left(-aq_t^2\right) d\mathbf{q_t}, \tag{6.85}$$

where the parameter $a = 0.6$ GeV^{-2}.

The light cone parton quantities x^+ and x^- are generated independently and according to the distribution:

$$u(x) \sim x^{\alpha}(1-x)^{\beta}, \tag{6.86}$$

where $x = x^+ > x_{min}$ or $x = x^- > x_{min}$. The parameters $\alpha = -1$ and $\beta = 0$ are chosen for the FRITIOF approach [10]. In the case of the QGSM approach [1] the parameters α and β are explained below. Masses of the excited strings should satisfy the kinematics constraints:

$$P_1'^{+} P_1'^{-} \geq m_{h1}^2 + q_t^2 \tag{6.87}$$

and

$$P_2'^{+} P_2'^{-} \geq m_{h2}^2 + q_t^2, \tag{6.88}$$

where the hadronic masses m_{h1} and m_{h2} are defined by string quark contents. Thus, a random selection of the values x^+ and x^- is limited by the above constraints.

Parton exchange and hadron structure functions

In the QGSM approach [3] strings (as result of a parton rearrangement) should be spanned not only between valence quarks of the colliding hadrons, but also between valence and sea quarks and between sea quarks. Each participating hadron should be split into a set of partons: valence quark and antiquark for meson or valence quark (antiquark) and diquark (antidiquark)

for baryon (antibaryon) as it was explained in the particle structure component section, and, additionally, into $(n-1)$ sea quark-antiquark pairs (their flavours are selected according to the probability ratios $u:d:s=1:1:0.35$), if a hadron is participating in n inelastic collisions. As an option the possibility to split a hadron into sea diquark-antidiquark pairs is included. Such assumption enhances the antibaryon production in the central rapidity region.

Thus, for each participating hadron we have to generate a set of the light cone variables x_{2n}, where $x_{2n}=x_{2n}^{+}$ or $x_{2n}=x_{2n}^{-}$ according to the distribution:

$$f^h(x_1,x_2,...,x_{2n})=f_0\prod_{i=1}^{2n}u_{q_i}^h(x_i)\delta(1-\sum_{i=1}^{2n}x_i), \tag{6.89}$$

where f_0 is a normalization constant. In the last expression $u_{q_i}^h(x_i)$ denote the hadron structure functions for valence quark (antiquark) q_v, sea quark and antiquark q_s and valence diquark (antidiquark) qq:

$$u_{q_v}^h(x_v)=x_v^{\alpha_v},\ u_{q_s}^h(x_s)=x_s^{\alpha_s},\ u_{qq}^h(x_{qq})=x_{qq}^{\beta_{qq}}, \tag{6.90}$$

where $\alpha_v=-0.5$ and $\alpha_s=-0.5$ [1] for the non-strange quarks (antiquarks) and $\alpha_v=0$ and $\alpha_s=0$ for strange quarks (antiquarks), $\beta_{uu}=1.5$ and $\beta_{ud}=2.5$ in the case of proton (antiproton) and $\beta_{dd}=1.5$ and $\beta_{ud}=2.5$ in the case of neutron (antineutron). Usually x_i is selected between $x_i^{min}\le x_i\le 1$, where the model parameter $x^{min}=0.3/\sqrt{s}$ is a function of initial energy $\sqrt{s}$. It prevents the production of strings with low masses (smaller than hadron masses), while the whole selection procedure should be repeated.

The transversal momenta of partons $\mathbf{q_{it}}$ are generated according to the Gaussian probability Eq. (6.85) with $a=1/4\lambda(s)$, where $\lambda(s)$ is calculated according to the Eq. (6.15), and under the constraint: $\sum_{i=1}^{2n}\mathbf{q_{it}}=0$. Thus, the partons are considered as the off-shell or virtual partons.

6.7.5 Kinky string excitation

Having sampled configuration of the kinky strings we perform kinky string excitations.

It is performed by means of two steps for each pair of kinky strings. At first step we determine the total 4-momentum of each kinky string from the chosen pair. Then we generate 4-momentuma of the gluon-kinks. The first step is fulfilled in the same way as it was described for a pair of the longitudinal strings. To fulfill the second step we assume that kinky strings are produced as the result of $gg\to gg$ hard interactions. Our consideration

of the outgoing gluons (kinks) momenta is based on the two-jets inclusive production cross section:

$$\frac{d\sigma_{gg}}{dx_g^+ dx_g^- d\cos\theta} = f(x_g^+, Q^2) f(x_g^-, Q^2) \frac{d\sigma_{gg}(\hat{s})}{d\cos\theta}, \tag{6.91}$$

where we take $\hat{s} = Q^2 = x_g^+ x_g^- s$ and s is the total center of mass energy squared for the colliding system that is calculated using x_i and q_{ti} and $m_i^2$of the string end partons. The value of s should be large enough to produce gluons with the transverse momentum above the chosen cutoff $Q_0^2 = 2\ GeV^2$. $f(x, Q^2)$ is the momentum fraction distribution of gluons in the hadron. It can be chosen from [11]. The QCD gluon-gluon interaction cross section $\frac{d\sigma_{gg}(\hat{s})}{d\cos\theta}$, where θ is the scattering angle in the center of mass of the parton-parton system and it was calculated in the Born approximation [12]. Simulation of an elastic gluon-gluon scattering is described in the elastic scattering components section.

Thus, the MC procedure to build kinky strings can be outlined as follows:

- Sample x_i, $\mathbf{q_{it}}$ and m_i^2, where $i = 1, 2, ..., 2n$, for the partons, which will be on the $2n$ string ends for both soft and kinky strings.
- For each pair of the kinky strings calculate the total center of mass energy s and sample $x^+ > x_{min}^+$ and $x^- > x_{min}^-$, where $x_{min} = 2Q_0/\sqrt{s}$, using the gluon distribution.
- Sample the center of mass scattering angle θ for outgoing gluons and calculate their 4-momenta.
- For each kinky string recalculate the energies and momenta of parton string ends.

This procedure should be improved taking into account gluon radiation.

Bibliography

[1] Kaidalov A. B., Ter-Martirosyan K. A., Phys. Lett. **B117** 247 (1982).

[2] Capella A., Sukhatme U., Tan C. I., Tran Thanh Van. J., Phys. Rep. **236** 225 (1994).

[3] Amelin N. S., Bravina L. B., Sov. J. Nucl. Phys. **51** 221 (1990).

[4] Baker M. and Ter–Martirosyan K. A., Phys. Rep. **28C** 1 (1976).

[5] Abramovskii V. A., Gribov V. N., Kancheli O. V., Sov. J. Nucl. Phys. **18** 308 (1974).

[6] Amelin N. S., Stöcker H, Greiner W., Armesto N., Braun M. A., Pajares C., Phys. Rev. **C52** 362 (1995).

[7] Werner K., Drescher H. J., Furler E., Hladik M., Ostapchenko S., in Proc.: 3rd Inter. Conf. on Physics and Astrophysiks of Quark-Gluon Plasma, Jaipur, India, March 17-21, 1997.

[8] Ranft J, Capella A., Tran Thanh Van, Phys. Lett. **B320** 346 (1994).

[9] UA1 Collab., Albajar C. *et al.*, Preprint CERN-EP/88-29.

[10] Andersson B., Gustafson G., Nielsson-Almquist, Nucl. Phys. **281** 289 (1987).

[11] Capella A., Kaidalov A., Merino C., Tran Thanh Van J., Phys. Lett. **B343** 403 (1995).

[12] Combrige B. L. , Kripfganz J. , Ranft J. , Phys. Lett. **70B** 234 (1977); Cutler R., Sivers D., Phys. Rev. **D17** 196 (1978).

6.8 Nuclear model components

6.8.1 Nuclear properties

We have tabulated the experimental values [1] of nuclear masses and binding energies. There is a possibility to calculate nuclear masses as well as binding energies applying the liquid drop model formulas.

Bethe-Weizsäcker's formula

In this approach [2] the nucleus binding energy $B(A,Z)$ is given by the sum of volume, surface and Coulomb energy terms, respectively:

$$B(A,Z) = \\ = -0.01587A + 0.01834A^{2/3} + 0.09286\left(Z - \frac{A}{2}\right)^2 + 0.00071\frac{Z^2}{A^{1/3}}, \tag{6.92}$$

where A is the number of nuclear nucleons and Z is the number of nuclear protons.

Nucleon potential

For i-th nucleon with radius r_i this potential is defined by a sum of the Fermi-potential

$$T^F(r_i) = \frac{[p^F(r_i)]^2}{2m_i}, \tag{6.93}$$

binding energy $B(A,Z)$ and Coulomb (for protons only) potential $V_C(Z,r_i)$:

$$V(A,Z,r_i) = T^F(r_i) + B(A,Z) + V_C(Z,r_i). \tag{6.94}$$

The Fermi-momentum in the local Thomas-Fermi approximation [3] depends on proton or neutron density $\rho_{Z,N}(r)$ as follows

$$p^F(r) = \hbar(3\pi^2\rho_{Z,N})^{1/3}, \tag{6.95}$$

were $\hbar = 197.327$ MeVfm.

The Coulomb barrier (in MeV) is given by

$$V_C(A,Z) = C\frac{Z}{r_0(1+A^{1/3})}, \tag{6.96}$$

where $C = 1.44$ MeVfm and $r_0 = 1.3$ fm.

6.8.2 Nucleus initial state simulation

The initialization of colliding nucleus, consisting of A nucleons and Z protons ($N = A - Z$ neutrons) with coordinates $\mathbf{r}_i$ and momenta $\mathbf{p}_i$, where $i = 1, 2, ..., A$, should be performed in the case of nuclear collision simulation by different models (see, e.g., the inelastic nuclear collision component section).

The MC initialization procedure can be outlined as follows:

1. Nucleon radii r_i are selected randomly at the rest of nucleus according to proton or neutron density $\rho(r_i)$. The normalized nucleon density for heavy nuclei with $A > 16$ [4] is

$$\rho(r_i) = \frac{\rho_0}{1 + \exp\left[(r_i - R)/a\right]} \tag{6.97}$$

where

$$\rho_0 \approx \frac{3}{4\pi R^3}\left(1 + \frac{a^2\pi^2}{R^2}\right)^{-1}. \tag{6.98}$$

Here $R = r_0 A^{1/3}$ fm and $r_0 = 1.16(1 - 1.16A^{-2/3})$ fm and $a \approx 0.545$ fm. The normalized density of light nuclei with $A < 17$ is given by the harmonic oscillator shell model [5], i. e.,

$$\rho(r_i) = (\pi R^2)^{-3/2} \exp\left(-r_i^2/R^2\right), \tag{6.99}$$

where $R^2 = 2/3 < r^2 >= 0.8133A^{2/3}$ fm^2 or

$$\rho(r_i) = \frac{4}{\pi^{3/2}R^3}\left[1 + \frac{A-4}{6}\left(\frac{r_i}{R}\right)^2\right]\exp\left(-r_i^2/R^2\right), \tag{6.100}$$

where

$$R^2 = \left(\frac{5}{2} - \frac{4}{A}\right)^{-1}(< r_{ch}^2 >_A - < r_{ch}^2 >_p). \tag{6.101}$$

The mean charge radii squared for nucleus $< r_{ch}^2 >_A$ and proton $< r_{ch}^2 >_p$ are taken from the lepton-nucleus scattering experiments [6]. To take into account the nucleon repulsive core it is assumed that inter-nucleon distance $d > 0.8$ fm;

2. The nuclear radius with counting of diffuseness is determined from the condition:

$$\frac{\rho(R)}{\rho(0)} = 0.01; \tag{6.102}$$

3. The momenta of the nucleons are randomly chosen between 0 and $p_{max}^F(r)$ with equal probabilities, where $p_{max}^F(r)$ is calculated according to the Eq. (6.95).

4. To obtain coordinate and momentum vector components, it is assumed that nucleons are distributed isotropically in the configuration and momentum spaces;

5. The nucleus must be centered in configuration space around $\mathbf{0}$, *i. e.* $\sum_i \mathbf{r}_i = \mathbf{0}$ and it must be at rest, i. e., $\sum_i \mathbf{p_i} = \mathbf{0}$ and $\sum_i \mathbf{r_i} \times \mathbf{p_i} = \mathbf{0}$. To provide the first two conditions we can perform shifts of nucleon coordinates $\mathbf{r'_i} = \mathbf{r_i} - 1/A \sum_i \mathbf{r_i}$ and momenta $\mathbf{p'_i} = \mathbf{p_i} - 1/A \sum_i \mathbf{p_i}$.

6. There is a possibility to unpack a nucleus into *free* streaming nucleons having effective masses by computing the energy per nucleon as $e = E/A = m_N + B(A,Z)/A$, where m_N is the nucleon mass and nucleus binding energy $B(A,Z)$ given by the Bethe-Weizsäcker formula (Eq. (6.92)), and assigning the effective mass $m_i^{eff} = \sqrt{(E/A)^2 - p_i^{2\prime}}$ for each nucleon.

Bibliography

[1] Audi G., Wapstra A. H., Nucl. Phys. **A595** V. 4 409 (1995).

[2] Bohr A., Mottelson B. R., Nuclear Structure, W. A. Benjamin, New York, Vol. 1, 1969.

[3] DeShalit A., Feshbach H., Theoretical Nuclear Physics, Vol. 1: Nuclear Structure, Wyley, 1974.

[4] Grypeos M. E., Lalazissis G. A., Massen S. E., Panos C. P., J. Phys. **G17** 1093 (1991).

[5] Elton L. R. B., Nuclear Sizes, Oxford University Press, Oxford, 1961.

[6] Barrett R. C. and Jackson D. F., Nuclear Sizes and Structure. Oxford University, New York, 1977.

6.9 Inelastic nuclear collision components

Here we describe a composite model component that is referred as the Pomeron based Parton String Model (PPSM). This model is devoted to predict inelastic collisions of a large variety of hadron (the same types as final state hadrons) and nuclear projectiles with nucleon and nuclear targets. The produced hadrons belong to the scalar and vector meson nonets and the baryon (antibaryon) octet and decuplet. The allowed bombarding energies in a hadron-nucleon collision should be above 2-pions production threshold. In the case of hadron-nucleus or nucleus-nucleus collisions the initial energies $\sqrt{s} > 5$ AGeV are recommended. This model is also able to predict final states (produced hadrons) in photon-nucleon and photon-nucleus inelastic collisions at the initial energies $\sqrt{s} > 5$ AGeV.

6.9.1 Nuclear inelastic collision algorithm

The PPSM algorithm includes a following set of the steps should be performed:

- Create two vectors (a projectile and target) of particles. Assign the initial projectile and target particle types, their coordinates and momenta. In the case of a hadron-nucleus (or nucleus-nucleus) interaction one should perform the target nucleus (or projectile and target nuclei) initial state simulation (see the nuclear model component section) and sample an impact parameter;
- Sample collision participants and separated them into a diffractive and non-diffractive. Store the total interaction four momentum for the participants;
- For each non-diffractive inelastic collisions sample the numbers of longitudinal and kinky string can be produced;
- Excite and re-excite colliding particles in the case of diffractive collisions and create diffractive longitudinal strings, if particles are not participate in the further soft or hard collisions;
- Perform the longitudinal and kinky string excitations;
- Perform the simulation of string decays (see the string decay component section);
- Correct energies and momenta of produced particles (see the utility components section), if it is necessary.

6.9.2 Initial state simulation

An impact parameter value ($0 \leq b \leq R_p + R_t$) is randomly selected according to the distribution:

$$P(\mathbf{b}) = b. \tag{6.103}$$

Then one should update the transversal components of projectile coordinates:

$$r_{xi} \rightarrow r_{xi} + b_x \tag{6.104}$$

and

$$r_{yi} \rightarrow r_{yi} + b_y. \tag{6.105}$$

For the target nucleus, if a projectile is centered on the origin of coordinate system, one should perform a shift of nucleon longitudinal coordinates:

$$r_{zi} \rightarrow r_{zi} + \Delta r_z/\gamma_i, \tag{6.106}$$

where $\Delta r_z = R_p + 1.5\ fm + R_t + 1.5\ fm$ can be taken.

The chosen working frames, where an interaction process is simulated, are the c. m. frame of colliding particles and particle equal velocities frame, respectively, in the case of particle and nuclear interactions.

In the case of a fast moving nucleus with the initial momentum per nucleon $\mathbf{P_0} = \{0, 0, P_{z0}\}$ one should perform Lorentz transformation of the nucleon longitudinal momenta

$$p_{zi} \rightarrow \gamma_i(p_{zi} - \beta_i e_i) \tag{6.107}$$

and the nucleon energies

$$e_i \rightarrow \gamma_i(e_i - \beta_i p_{zi}), \tag{6.108}$$

where β_i is defined as

$$\beta_i = \frac{P_{z0}}{\sqrt{P_{z0}^2 + m_i^{eff2}}} \tag{6.109}$$

and γ_i is given by

$$\gamma_i = \frac{1}{\sqrt{1 - \beta_i^2}}. \tag{6.110}$$

The m_i^{eff} denotes an effective i-th nucleon mass.

For a fast moving nucleus one needs also to perform its Lorentz contraction, i. e.,

$$\mathbf{r_i'} = \mathbf{r_i} - \frac{\gamma}{1+\gamma}\mathbf{v}(\mathbf{v}\mathbf{r_i}), \tag{6.111}$$

where $\mathbf{v}$ is the nuclear velocity and $\gamma = 1/\sqrt{1-v^2}$. The nucleus initialization can be performed at once, even if one needs to simulate multiple nuclear interactions, when the same nucleus is participating. The nucleus is randomly rotated in coordinate and momentum spaces before repeated interaction. To perform this rotation three Euler angles: $0 < \theta_1 < 2\pi$, $0 < \theta_2 < 2\pi$ and $0 < \theta_3 < 2\pi$ are selected (with the uniform probabilities) randomly. Then the nucleon coordinates $\mathbf{r_i} = \{r_{xi}, r_{yi}, r_{zi}\}$ and momenta $\mathbf{p_i} = \{p_{xi}, p_{yi}, p_{zi}\}$ are rotated sequentially:

$$\mathbf{r_i} \to \mathbf{r_i}R_1 \to \mathbf{r_i}R_2 \to \mathbf{r_i}R_3, \tag{6.112}$$

and

$$\mathbf{p_i} \to \mathbf{p_i}R_1 \to \mathbf{p_i}R_2 \to \mathbf{p_i}R_3, \tag{6.113}$$

where the rotation matrices are determined as follows

$$R_1 = \begin{vmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \tag{6.114}$$

and

$$R_2 = \begin{vmatrix} \cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 1 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 \end{vmatrix} \tag{6.115}$$

and

$$R_3 = \begin{vmatrix} \cos\theta_3 & -\sin\theta_3 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{vmatrix} \tag{6.116}$$

6.9.3 Nuclear collision participants

The ultra-relativistic inelastic hadron-nucleus and nucleus-nucleus collisions are considered as independent hadron-nucleon and nucleon-nucleon inelastic collisions. It was shown long time ago [1] for the hadron-nucleus collision that such picture can be obtained starting from the Regge-Gribov approach [2], where the hadron-nucleus elastic scattering amplitude is a result of reggeon exchanges between initial hadron and nucleons from a target-nucleus. This result can be extended for the nucleus-nucleus collision case and leads to a simple and efficient MC procedure [3] to define interaction cross sections and number of the nucleons participating in the inelastic nucleus-nucleus collision:

1. We randomly distribute A nucleons from the projectile-nucleus and B nucleons from the target-nucleus inside the impact parameter plane according to the weight functions $T([\vec{b}_i^A])$ and $T([\vec{b}_j^B])$, respectively. These functions represent probability densities to find sets of nucleon impact parameters $[\vec{b}_i^A]$ and $[\vec{b}_j^B]$, where $i = 1, 2, ..., A$ and $j = 1, 2, ..., B$, respectively.

2. For each pair of nucleons i and j with chosen impact parameters $\vec{b}_i^A$ and $\vec{b}_j^B$ we check whether they interact inelastic or not using the probability $p_{ij}(\vec{b}_i^A - \vec{b}_j^B, s)$, where $s_{ij} = (p_i + p_j)^2$ is the total c.m. energy squared of the given nucleons with the 4-momenta p_i and p_j, respectively.

The described MC procedure is based on the probability $P(\vec{B}, s)$ of a such configuration, when several pairs of nucleons from the projectile and target nuclei interact inelastically and the rest of the nucleons do not participate in collisions:

$$P(\vec{B}, s) = \left\langle \prod_{i,j=1} p_{ij}(\vec{b}_i^A - \vec{b}_j^B, s) \prod_{k,l=1} [1 - p_{kl}(\vec{b}_k^A - \vec{b}_k^B, s)] \right\rangle, \qquad (6.117)$$

where $\vec{B}$ is an impact parameter for the colliding nuclei and s is total c.m. nucleon-nucleon energy squared. To simplify notations, we assume, that all interacting nucleon pairs have the same s. The last equation can be rewritten more explicitly as follows

$$\begin{aligned} P(\vec{B}, s) = \int \prod_{i,j=1} p_{ij}(\vec{b}_i^A - \vec{b}_j^B, s) \prod_{k,l=1} [1 - p_{kl}(\vec{b}_k^A - \vec{b}_l^B, s)] \times \\ \times T_A(\vec{b}_1^A) T_A(\vec{b}_2^A) ... T_A(\vec{B} - \vec{b}_B^B) d\vec{b}_1^A d\vec{b}_2^A ... d\vec{b}_B^B. \end{aligned} \qquad (6.118)$$

The probability for an inelastic collision of the nucleons i and j as a function of the squared impact parameter difference $b_{ij}^2 = (\vec{b}_i^A - \vec{b}_j^B)^2$ and s can be calculated within the Regge-Gribov approach [4], [2] (see the hadron cross section section).

We would stress that in the Pomeron eikonal model [2] we count only Pomeron exchanges for the hadron elastic scattering amplitude. With such amplitude we are not able to predict properly the experimental values of hadron interaction cross sections and also the corresponding interaction probabilities in the whole energy range under consideration. We will strongly underestimate low energy parts of the cross sections and the corresponding interaction probabilities.

At this energy region neglecting by the real part of a hadron scattering amplitude we can correct the eikonal variables (see the hadron cross section

section) expressing them through experimental values of the total $\sigma_{tot}(s)$, elastic $\sigma_{el}(s)$ and single diffraction $\sigma_d(s)$ cross sections:

$$\frac{z}{2} = \frac{\sigma_{tot}(s)}{4\pi B_{el}(s)} \tag{6.119}$$

$$\lambda(s) = 2B_{el}(s), \tag{6.120}$$

where

$$B_{el}(s) = \frac{\sigma_{tot}^2(s)}{8\pi[\sigma_{el}(s) + \sigma_d(s)]}. \tag{6.121}$$

Bibliography

[1] Capella A. and Krzywicki A., Phys. Rev. **D18** (1978) 4120.

[2] Baker M. and Ter–Martirosyan K. A., Phys. Rep. **28C** (1976) 1.

[3] Amelin N. S., Gudima K. K., Toneev V. D., Sov. J. Nucl. Phys. **51** (1990) 327; Amelin N. S., JINR Report **P2-86-56** (1986).

[4] Abramovskii V. A., Gribov V. N., Kancheli O. V., Sov. J. Nucl. Phys. **18** (1974) 308.

6.10 Utility components

We have developed a set of the components to fulfill utility functions in the course of particle and nuclear collisions modeling. As a rule these components deal with particle 4-momenta [1].

6.10.1 Energy-momentum correction

Sometimes we need to correct the energy and momentum of each produced particle to guarantee the energy-momentum conservation law for a colliding system. The correction procedure consists of three steps:

- We boost 4-momenta of the produced particles into their center of mass frame using the total velocity of produced particles.
- We scale 3-momentum of each produced particle until their summary mass is close to the effective mass of the colliding system. The scaling procedure is controlled by a difference of a scale factor from the unit.
- We perform the reverse boost of 4-momenta of the produced particles into the initial colliding frame using the velocity of the colliding system.

6.10.2 Particle distributions

As result of a particle or nuclear collision simulation we obtain the values of particle 3-momenta (p_x, p_y, p_z) , their energies (E), their quantum numbers, i.e., the particle types. This information allows us a possibility to build different particle distributions. The particle rapidity $y = -0.5 \ln \frac{E+p_z}{E-p_z}$ distribution is calculated according to the equation:

$$\frac{d\bar{n}}{dy} = \frac{1}{N_{evt}} \sum_{k=1}^{N_{evt}} \sum_{j=1}^{n^k} \delta(y_j^k - y), \tag{6.122}$$

where $\bar{n}$ is the mean multiplicity of particles of the required type, N_{evt} is the number of generated collision events, n^k is the number of particles of required type in k-th event. Similarly we can build other particle distributions, e.g., transversal momentum $\vec{p}_t$ distribution:

$$\frac{d\bar{n}}{d^2p_t} = \frac{1}{N_{evt}} \sum_{k=1}^{N_{evt}} \sum_{j=1}^{n^k} \delta^2(p_{tj}^k - p_t). \tag{6.123}$$

The particle multiplicity n distribution $P(n)$ is calculated according to

$$P(n) = \frac{1}{N_{evt}} \sum_{i=1}^{N_{evt}} \delta(n^i - n), \tag{6.124}$$

where n^i is the multiplicity of required particles in i-th event.

Bibliography

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

Received by Publishing Department
on August 23, 2001.

Амелин Н. С., Комогоров М. Э.
Д11-2001-175

Компонентно-ориентированный подход
к разработке и использованию численных моделей
в физике высоких энергий

Обсуждаются основные концепции компонентного подхода к разработке и применению численных моделей в физике высоких энергий. Данный подход реализован в виде системы программ NiMax. Обсуждаемые концепции иллюстрируются многочисленными примерами работы пользователя системы. В приложении описаны физика и численные алгоритмы компонентов для моделирования событий взаимодействия частиц и ядер при высоких энергиях. Эти компоненты входят в состав прикладных модулей адронных моделей, созданных в рамках данной системы. Данное описание может рассматриваться как предварительная версия руководства для пользователей моделирующих компонентов, работающих с системой NiMax.

Работа выполнена в Лаборатории высоких энергий ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна, 2001

Amelin N. S., Komogorov M. E.
D11-2001-175

Component-Oriented Approach to the Development
and Use of Numerical Models in High Energy Physics

We discuss the main concepts of a component approach to the development and use of numerical models in high energy physics. This approach is realized as the NiMax software system. The discussed concepts are illustrated by numerous examples of the system user session. In appendix chapter we describe physics and numerical algorithms of the model components to perform simulation of hadronic and nuclear collisions at high energies. These components are members of hadronic application modules that have been developed with the help of the NiMax system. Given report is served as an early release of the NiMax manual in the main for model component users.

The investigation has been performed at the Laboratory of High Energies, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna, 2001

Макет *Т. Е. Попеко*

ЛР № 020579 от 23.06.97.
Подписано в печать 23.01.2002.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 7,0. Уч.-изд. л. 9,78. Тираж 170 экз. Заказ № 53003.

Издательский отдел Объединенного института ядерных исследований
141980, г. Дубна, Московская обл., ул. Жолио-Кюри, 6.