E1-2001-31

N.S.Amelin*, M.E.Komogorov

NiMax SYSTEM FOR HADRONIC EVENT GENERATORS IN **HEP**

The talk presented at the XV International Seminar
on High Energy Physics Problems, 25 – 29 September, 2000,
Dubna, Russia

*E-mail: Nikolai.Ameline@phys.jyu.fi

1 Introduction

The full description of high-energy hadronic and nuclear interactions from the first principles of quantum chromodynamics (QCD) is rather 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. It makes the task of development and use of the QCD motivated phenomenological models to be very important. Such models are extremely popular for predictions of new phenomena and the analysis of obtained experimental data. Besides, the experimental detectors design, their construction and performance require careful numerical simulations. The Monte Carlo (MC) phenomenological models often referred as the MC event generators are standard tools to perform such simulations [1].

All widely used model codes were written in Fortran. Nevertheless, the most of physics centers have turned now to use the object-oriented languages (C++, Java) to develop physics related software, because object-oriented programming has many advantages compared with traditional procedural coding (see, e.g., [2]).

In this talk we present an object-oriented system to use and develop numerical models. In our opinion it could essentially facilitate model user work and increase productivity of model developer work. We refer our system as the NiMax system. The main idea of this software system is to support component approach to the development and use of numerical models [3], [4]. The component can represent a model of either single physical

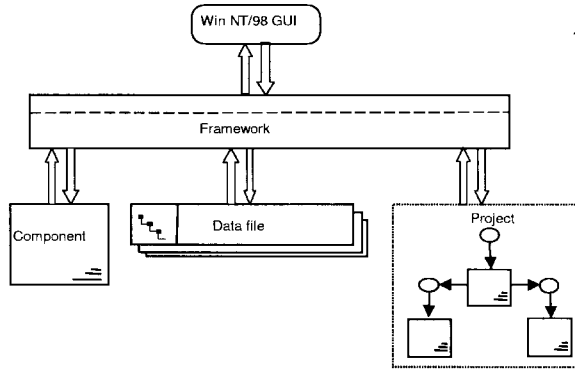

Figure 1: An architectural overview of the NiMax system.

process, e.g., hadron elastic scattering, or very complicated physical phenomenon, e.g., ultra-relativistic heavy-ion collision.

Any model component deals with data. It can generate very large bulk of data and store it on a disc for further analysis and visualization. It can read data from a disc and process it. The collaborative work of several components requires the data exchange. To fulfill these needs we have applied the idea of a data event [3], [4] [5] as a configured portion of data.

We introduce the data file and a component can write and read data events in this file. We suggest the model project concept. The model projects are collections of components that are connected in sequence to process numerical data.

To support the component management and development as well as the data and project management we have developed a large set of C++ classes. They are thought as the object-oriented framework. For our system we distinguish between system users and model developers (advanced users). We assume that a user can interact with our system by means of the user interface without writing and modifying any of source codes.

For users we have developed graphical user interface (GUI). Working on the system and particularly on the GUI we have applied the document - view paradigm (see [6]), when an object holds the data and its views display the data and allows editing.

The NiMax system is written in pure C++ and only the graphical user interface (GUI) is based on the Microsoft foundation classes (MFC) library [6]. Now our system with the GUI is working on the different Windows platforms. However, the NiMax system is portable (on the level of the source code), if it is applied in the command line mode.

A system architectural overview is presented in Fig. 1. Its central part is the framework. The components, data files, projects and GUI are linked together by framework application programming interface (API), i.e., by framework public or protected methods (open arrows).

2 NiMax component

We are considering any component as a set of standardized interfaces between its numerical algorithm and the outside world. An interface includes several methods and some related data.

2.1 Component interfaces

By means of the input interface a user sends a request for a component and provides necessary input data to fulfill this request. A request as well as input data is provided in form of the input map [3], [4]. An input map is based on the list of simple data types and has linear data structure.

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 can edit model parameters and switches. The input interface and tuning interface are the same interfaces from the developer's point of view (the same classes in use [3], the same data structures in use). But the separation of these interfaces makes user work to be more convenient.

The result of component execution is obtained by means of the output interface. This interface assists either to write component output data in the data file or to send component output data for other components.

A component can also read necessary input data from the data file or receive it from another component by means of the matching interface. It allows a component to select data according to the matching configuration [5]. A particular matching configuration is realized as a matching map.

The enumerated standard interfaces are not unique. We can imagine a component that needs an external data interface to receive the data having external format. By adding more standard component interfaces we can extend the application area of the NiMax system.

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

Each component interface is complemented by at least one view. These views help a component user to perform many actions on the component state (see the graphical user interface section).

2.2 Component development

Besides standardized interfaces and views the system components have other common elements. Each component has its own component factory to create the component objects and includes the component static information, which is needed for component object creation. Any component has its unique identifier. The knowledge of a component's identifier helps us to obtain full information about the component. Particularly, the composite component aggregating (see below) other components should include their proxies [3].

The observation of many common component elements allowed us to develop the component frames [3], [4] with aim to produce the so-called skeleton components simply by editing of a particular component frame. Such frame can be thought as the component template that is supporting a correct programming style. Thus, a component developer

needs to design only the component application algorithm. To help a component developer we created the component wizard similar as the Microsoft Visual C++ 6.0 class wizard [6]. The component wizard is the code generator that produces a component skeleton by means of the dialog windows.

A component developer can extend functionality of ready component applying the inheritance mechanism [3], [5] that is supported by our system. The interfaces of a base component and derived component are joined as well as their public APIs.

A new component can be developed by the aggregation of existing components [5]. A composite component can include several aggregated components. A particular component can include the tree of aggregated components. The tuning and output interfaces of aggregated components are simply joined to the relevant aggregating component interfaces.

There are several important features of the suggested aggregation mechanism. A component user is able to see the composite component structure and has access to any sub-components by means of the GUI. There are no limitations to create an efficient component code, e.g., as compared with the standard C++ coding for the developer, which is working on the creation of a component by aggregation of other components. The APIs of aggregated components are called directly. In this case the component coding is even simplified, e.g., no efforts are required to create and destroy the aggregated component objects. The inheritance mechanism provides a possibility of the runtime substitution of sub-components in the aggregating components.

2.3 Component documentation

Each component should be accompanied with its documentation. The component documentation includes the names of component authors, descriptions of component applicability, its input maps, parameters, sub-components in use, numerical algorithm, etc. The component documentation is realized as a set of html files.

The writing of the component documentation is very important part of the component development process. We offer the documentation frames and documentation wizard (for the Windows platform only) for the component developers. The component documentation frames allow a developer to present information with the standard appearance. They are similar as the component frames discussed below and developed in the complement of model component frames.

2.4 Component packaging

Several implemented components can share common data, functions and classes that are related to a particular application domain. Such common software facilitates the work of a component developer and increases its productivity. It allows more efficient use of the computer resources by a developer. We suggest packaging of built components with their related software into the modules [5] that are referred as application domain modules. We consider these modules as the developer's units. For a component developer it is more natural and more efficient to work on the component code within some component related software environment than on an isolated component code. We have also created the module frames that are similar as the component frames. Working on the module concept we assumed that they should be self-sufficient on the level of source code, i.e., no

external methods, no external implementations, etc, are required to compile and execute module components. We consider these modules as the units for distribution. By compilations they are prepared as dynamically linked (shared) libraries (DLLs) for our system users. However, the module independence does not forbid sub-component substitution, if aggregated sub-component and its alternative sub-component belong to the different DLLs.

The large set of the hadronic model components have been already implemented and included into the hadronic model modules [7]. It allows their users to perform simulation of the particle and nuclear interactions at wide high-energy range and for large set of projectiles and targets.

3 NiMax data model

This data model was designed to support the data management. Particularly, it allows us to organize independent component collaboration by transferring of complicated data structures between those components. It includes several important concepts: the data event, data file and data transfer classes.

3.1 Data event

The data event [5] is a portion of data that only consists of values of simple data types (int, float, double, etc). Any data event has its definition. The definition includes unique identifier and describes data event configuration.

The data event definition is a tree of data channel definitions. The data channel definition includes channel identifier, its type, its name and more information can be added. A data event is not a C++ object, i.e., it is not an instance of the definite class. The data of a data event can be placed into memory or stored on disc according to the data event definition.

We have suggested [5] the concept of the pre-defined data events and channels. These are pre-fabricated for definite actions on them. The pre-defined channels and pre-defined group of channels have fixed configurations (similarity with the C++ objects) and system user is not able to change their configurations. To deal with such data events the framework needs to know only their identifiers, e.g., the framework knows (due to the unique identifier) 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 data events representing component states. The framework knows how to display and execute such pre-defined data events.

3.2 Data file

A component can write and read data in the data file. The data file has its header, which is needed to identify the file and facilitate the navigation through it. The data file is separated into two parts: the data event configuration part and the data part. The presence of the data event configurations in the data file allows a system user or a component to perform very flexible selection of data. We refer it as the structural data selection. The data event configuration part has its own header and includes the list

of event definitions with their unique identifiers. The event configuration header keeps some statistical information about the data events. The data part consists from the data records, where the tree-structured data are written. Each data record has its own header, where the information to help for navigation through the data is stored.

For system users we have created several views for the data file: the data configuration view, data view and view of statistical information about the written data.

3.3 Data transfer classes

We have created the data transfer class (DTC) library [3], [4] with aim to increase the productivity of hadronic component developers. Many of the implemented components can be considered as generators or converters of the DTC objects. These objects are exchanged between components in the case of component aggregation. The most of the DTC objects are counterparts of real high-energy physics objects (particle, parton, nucleus, etc).

Within our system we have no strict recommendations how to build such classes. The most important thing is that any object of a data transfer class should support persistence, i.e., this class needs methods to write to the data file and read from the data file its object states. The DTCs help a component developer to design component output as well as an output of the history of physical event generation.

The DTC library has a hierarchical structure, because it is very convenient to have a base classes that either factorize common properties of all objects of this hierarchy or collect the most of operations needed to manage these objects. These classes help to implement universal numerical algorithms. The degree of universality of an algorithm is determined by the amount of needed input-output information. The most universal algorithms are those that use, as input and output, the objects of node data transfer class.

By introducing of the data transfer term we would stress that set of classes providing described functionalities can be developed in different application domains.

4 NiMax project

A component could receive the data events produced and sent by other components. Thus, several components can be assembled into an event-oriented project to perform a collaborative work.

4.1 Matching of data configurations

The produced data events have tree structures that are described by data event configurations. The framework analyses event configurations and searches the necessary data configuration for a component. We refer this process as the matching of a data configuration [5]. If the necessary configuration is found we refer this situation as observation of an entry point into the component input data.

The necessary data configurations for a given component are described by the component matching configurations or matching maps. A component developer can offer several matching maps. It provides a room for a component user to tune the component by registering of a chosen matching maps as the default one.

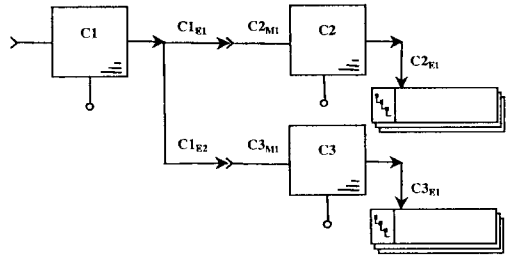

Figure 2: An event-oriented project.

It is important to stress that ability of a component to recognize necessary data obtained from the data file or from other components requires from the component developer neither to learn or use the system or component classes (objects) nor to have a knowledge about the source (the data file, other components) of events that component will receive. The component developer does not need to learn the configurations of data events, which can be received by the component. He or she has to know how to create the configurations of the required input data.

4.2 Component collaboration

The component collaboration is their interaction through standard output and matching interfaces [5].

We distinguish two different situations for the component collaboration. The first one is that a component writes its output data events in the data file and another component reads these data from the data file. We consider this situation as the component collaboration through the data file bus having in mind a hardware analogy. For this type of interaction the component objects are completely isolated from each other.

The second situation 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 the data event bus. For this type of interaction the components are executed inside a common process.

A set of components that collaborates through the standard output and matching interfaces by sending and receiving data event messages and connected in sequence to process numerical data is referred as an event-oriented project [5].

The Fig. 2 demonstrates the example of a project. Here, the component $C1$ produces two events having configurations: $C1_{E1}$ and $C1_{E2}$, then components $C2$ and $C3$ receive data of those events that are selected in the accordance with the matching configurations $C2_{M1}$ and $C3_{M1}$, respectively. These components produce new data events configured as $C2_{E1}$ and $C3_{E1}$, respectively. The last events are written in the data file.

We consider only the projects, whose component execution order is defined by data flows, i.e., we consider only the so-called pipeline projects. For pipeline projects a system user does not need to write and compile the event control procedure. Any project includes a component that is executed at first. We refer this component as the start or main component. It reads its input by the input or matching interface. Thus, during a project

assembling a system user has to outline the participating components, point out a start component and provide information about pairs of collaborated (connected) components. Any component or sub-component from the project could write its output in the data file.

Any project is the NiMax system file. It consists of two parts. The first part can be considered as a project definition part. The second part includes the event control procedure. This file is constructed on the basis of the information obtained from a user through the project view.

Very often a project consists of only one component and, if we are performing a component execution, we always deal with a project file.

5 NiMax framework

The full description of the framework and other classes that belong to the NiMax system requires a separate paper or manual. Here, we would like to explain shortly the framework functionality offered by the control and navigation methods.

5.1 Control methods

There are many methods to control the component life cycle. In spite of the fact that the component life cycle consists mostly of internal framework processes, which are hidden from a system user we would like to give an idea about it. There are many possibilities for a user to influence the component life cycle. It includes several phases: the component instance creation phase, the edition phase, the execution phase and the destruction phase.

Before to start component object creation procedure the framework creates an environment for the component object. The context of this environment defines the optional variables, which are set to default values, and the output and input files, if they will be used. A user is able to modify these optional variables by means of the user interface, e. g., a user can either set own default parameters, input and matching maps or suppress some data event output or suppress the runtime information output. In the case of composite component, the aggregating component instance is created first. Then framework creates aggregated component instances. In order to create any component object the framework needs to know only the component's identifier. It uses the aggregated component's identifiers to look for their factories. If a factory is not found, the framework tries to find an alternative component according to the component proxy definitions and the component hierarchy. A user is able to control this process changing the context object and enabling or disabling the component substitution.

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

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

The framework helps the system users to execute the project files and controls these processes. The framework allows a user to execute several projects as different processes at the same time or can have several execution processes of one project, e.g., different component parameters in use, at the same time.

5.2 Navigation methods

The framework helps to fulfill the component librarian functions. Particularly, it allows a user to see total list of the components included into the system and to register the required component. This component view shows static information, because the component objects are not created yet. The framework allows a user to look through the project files as well.

The tree structure is heavily used in our system, e.g., the tree structure of the composite components and the tree structure of the data events. Thus the methods to navigate through a composite component and through the data file are very similar.

Using file navigation methods a model developer is also able to write the adapter or driver tools to transform the format of the data written down to system data file into the data formats, which are acceptable for the external packages.

Besides the runtime information and the different information messages, which can appear during the component life cycle a system user is offered more detailed help information. Any object of our framework such as component, parameter, error message, etc has the unique identifier. It opens a possibility to bind these identifiers with the HTML files describing the objects. Thus, a user can get help from *inside* the code by means of a unique identifier, e.g., by warning and error message identifiers the framework finds and opens the HTML files related to the detailed descriptions of these messages.

6 NiMax graphical user interface

After starting of the NiMax system, the main window will appear. This window includes many views. These views allow a user to work on model components, projects and data files as well as to obtain necessary help.

6.1 Component and project navigations

The available model components and model projects are displayed for a system user. There are three different views of the total component list for a user: by categories (default view) to see the components from different application categories, by modules (DLLs) to see the DLL component contents, by hierarchies to see the component relations. The component views show also the component types that are defined by presence or absence of particular standard interfaces [5]. The icons mark the component types. A file browser and selector that offer access to the projects by the open command from the file menu fulfill the project navigation.

6.2 Working on component states

By means of the component interface views the framework offers for a component user a possibility to change a component state. A user can see, register (instead of the default input map) and edit component input maps as well as see and edit component parameters. A user always has a possibility to use the default parameter values. A user has also access to any parameter of any sub-component inside the composite component.

A sub-component can be substituted by another sub-component, if they have 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., using the user interface. The component user can perform substitution inside any sub-component. It is a way to change and update the application algorithm of a composite component.

If a component is expected to produce output, the component user has a possibility to reconfigure the output. The total events or only some selected channels can be disabled for the output. In the case of a composite component, the output reconfiguration can be performed for any sub-component.

If a component is expected to receive data either from another component or from the data file, the component user has a possibility to choose and register suitable matching map. There is also a possibility to check that matched data configuration will be produced by another collaborated component or it really exists in the attached data file.

6.3 Project assembling and edition

A user can open a model project file for edition. He or she can create a new model project file as a starting point of the model project assembling. The system provides an environment, e.g., the name of output file, for the project assembling and execution.

All available components are displayed and a user can launch several of them in a project for collaborative work. Then a user should connect them one by another by their output and matching connectors. The projects are arranged so the data stream flows from left to right across the screen. Providing the links between the component connectors a user is able to reconfigure component outputs and to choose the suitable matching maps. Some components can be removed from the project file as well. Finally, a user can save the project file by giving a particular name.

6.4 Project execution control

A project user has a possibility to reset the execution environment for a project before its execution.

A user can obtain the information about current states of the execution processes, select a process and perform some actions on it, e.g., kill selected process, save its runtime information.

By means of the runtime messages the projects inform about their execution processes during run session. There can be information messages, warning messages and error messages. The information messages are used to tell something during normal execution process. The warning messages inform a user about potential errors or other situations, which are able to destroy normal execution process. The execution process will be continued after the appearance of the warning messages. The error messages appear, when the execution process is aborted. The error messages inform a user about place and type of the error. A user has the possibility to control the runtime information output.

6.5 Working on data files

There is a possibility to visualize data file content by means of the data file views. The data events as well as their channels can be selected through the data configuration

view. After selection one can perform different operations on data, e.g., copy, cut, protect against component access, etc.

6.6 Working on histograms and plots

One- and two-dimensional histograms with fixed partitions and two-dimensional plots can be created and visualized. Histogram statistics is implemented as bin content statistics. The average and root mean square values are calculated as well. Bin errors are always computed taking weights into account. All histogram objects can store bin values and errors as different types and user has a possibility to make a choice among these types. For any histogram object a user can work on its table view. One-dimensional histogram has also graphical view. Two-dimensional histograms have the cell graphical view. The plots are presented by the table and scatter plot graphical views.

A user can apply different operations on the selected data of any view. The selection methods are different for different views. For example, a user can use mouse to select content of bins for table view of a histogram object and for graphical views we offer the brush tool.

6.7 Obtaining a help

The help sub-system based on the HTML has been developed for the Windows platforms [6]. There are two types of the help: system help, i.e., help information to assist in the system usage, and the component help. Help topics related to the NiMax system describe how to handle a component, how to process data, etc. Help topics related to the particular component include the information about component parameters, its numerical algorithm, etc. Navigation through the help can be done using contents, keywords or by F1 button.

7 Conclusion

We have suggested a new approach to develop, assemble and use numerical models in high-energy physics. It is a component approach, when complex numerical model is composed from more simple components that are self-contained entities.

We have formulated the *standard component* by separating component interface part and component numerical algorithm part. Any component is thought as a set of standard interfaces between its algorithm and the outside world.

We have suggested a unit of configured data: *the data event*. A component can produce data events, write them in the data file for further analysis and visualization, read data events from the data file and receive data events from other components. The developed data file allows the storage of very large bulk of data and fast navigation through these data.

Several components can be composed into a composite model component or a model project in a variety of ways and the new components with their peculiarities can be added. It offers a great flexibility for the construction of a powerful numerical model. Each particular component can be tuned and different implementations of the component algorithm can be interchanged at runtime enabling a model user to obtain the 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 object-oriented system written in C++. Its central part is the framework that controls all the system's internal processes and provides an interaction between the graphical user interface and the rest of parts.

The framework supports many GUI services to have a convenient user session: launching of components and projects, controlled edition of component and sub-component parameters, substitution of sub-components within a composite component, re-organization of component and sub-component outputs, assembling and re-assembling of projects, execution of projects as separate processes, structural, analytical and graphical selections, histogramming and visualization of the data produced and written in the data files and many others.

We have developed different component frames and data transfer class library to facilitate creation and distribution of component codes by model developers. A model developer should work on the component application algorithm and each component application algorithm can be developed independently from other component application algorithms. Our components are also extendible and re-usable by the inheritance and aggregation mechanisms.

We have applied the NiMax system for a particular class of the numerical models: the MC event generators. Many MC model components have already been implemented.

The developed system can be used in different application domains. First of all, it is suitable to develop different applications based on the use of complicated numerical models. Besides of the numerical simulations it can be applied for the structural, analytical (based on the use of numerical models) and graphical selection and analysis of the data obtained from different sources. The data file, data file views and data file control and navigation methods can be thought as an independent data file system having its own applications.

Finally, we would like to acknowledge the Academy of Finland for the financial support under Grant No. 48477.

References

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

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

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

[4] N. Amelin and M. Komogorov, The talk published in Proc. of Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP2000), 7 - 11 February 2000, Padova, Italy, p. 119-123.

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

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

[7] N. Amelin and M. Komogorov, NiMax Hadronics: Model Components, in preparation.

Received by Publishing Department
on February 27, 2001.

Амелин Н.С., Комогоров М.Э. E1-2001-31
Система NiMax для адронных генераторов событий
взаимодействия частиц и ядер в физике высоких энергий

Предложен новый подход к разработке и применению монте-карло генераторов событий взаимодействия частиц и ядер в области физики высоких энергий. В этом компонентном подходе сложная численная модель собирается из стандартных компонентов. Он также открывает возможность организации библиотеки модернизируемых компонентов и обеспечивает большую гибкость при конструировании реалистических численных моделей. Для поддержки данного подхода создано специальное программное обеспечение, написанное на C++, которое получило название NiMax.

Работа выполнена в Лаборатории высоких энергий ОИЯИ.

Препринт Объединенного института ядерных исследований. Дубна, 2001

Amelin N.S., Komogorov M.E. E1-2001-31
NiMax System for Hadronic Event Generators in HEP

We have suggested a new approach to the development and use of Monte Carlo event generators in high-energy physics (HEP). It is a component approach, when a complex numerical model is composed of standard components. Our approach opens a way to organize a library of HEP model components and provides a great flexibility for the construction of very powerful and realistic numerical models. To support this approach we have designed the NiMax software system (framework) written in C++.

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

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

Макет Т.Е.Попеко

Подписано в печать 17.04.2001
Формат 60 × 90/16. Офсетная печать. Уч.-изд. листов 1,52
Тираж 375. Заказ 52607. Цена 1 р. 95 к.

Издательский отдел Объединенного института ядерных исследований
Дубна Московской области