[12pt]JHEP3
1 bottomnumber5
@outputbox
epsfig array
Im SU
-.5pt/{}

SHERPA 1.α, a proof-of-concept version CERN-TH/2003-284 Tanju Gleisberg
Stefan Höche
Frank Krauss
Andreas Schälicke
Steffen Schumann
Jan-Christopher Winter

The new multipurpose event-generation framework `SHERPA`, acronym for `S`imulation for `H`igh-`E`nergy `R`eactions of `PA`rticles, is presented. It is entirely written in the object-oriented programming language `C++`. In its current form, it is able to completely simulate electron–positron and unresolved photon–photon collisions at high energies. Also, fully hadronic collisions, such as, e.g., proton–anti-proton, proton–proton, or resolved photon–photon reactions, can be described on the signal level.

Standard Model, Higgs Physics, LEP HERA and SLC Physics, Tevatron and LHC Physics, QCD

document Introduction To a large amount, modern particle physics centres around accelerator experiments, where high-energetic particles are brought to collision. With rising energies, these interactions become more and more violent, leading to an increasing number of particles being produced. To confront the resulting experimental data with theoretical models, a systematic understanding of such multi-particle production processes is of paramount importance. A full, quantum-mechanically correct, treatment is, at the moment, out of reach. There are two reasons for this:

First of all, there only is a limited understanding of the non-perturbative phase of QCD, or, in other words, of how colourless hadrons are built from the coloured quarks and gluons. This is especially true for phenomena such as hadronisation or for questions related to the impact of the partonic substructure of the colliding hadrons on the pattern of multiple interactions. In all such cases, phenomenological models for the transition from hadrons to partons or vice versa have to be applied with parameters to be fitted. This clearly puts a constraint on a conceptual understanding of high-energy particle production processes. On the other hand, even considering the, in principle, well-understood perturbative phase of scattering processes alone, there are limits on our technical abilities to calculate all amplitudes that contribute to a given process. This is due to the fact that even at the tree-level the number of Feynman diagrams grows factorially with the number of particles involved. Moreover, at higher orders of the perturbative evolution new difficulties arise, which are connected for instance with the evaluation of multi-leg loop integrals.

This failure necessitates other, approximate solutions, such as simulation programs. These event generators decompose the full scattering process into a sequence of different stages, which are usually characterised by different energy scales. The past and current success of event generators, like `Pythia` Pythia or `Herwig` Herwig, in describing a full wealth of various data justifies this decomposition intrinsic to all such programs. As a by-product, the decomposition of events into distinguishable, more or less independent phases opens a path to test the underlying assumptions on the dynamics of particle interactions at the corresponding scales. These assumptions, in turn, can be modified and new models can be included on all scales. This property turns event generators into the perfect tool to bridge the gap between experimental data and theoretical predictions. It renders them indispensable for the analyses and planning of current and future experiments.

To meet the new challenges posed by the new experiments, for instance Tevatron at Fermilab and especially LHC at CERN, the traditional event generators `Pythia` and `Herwig`, so far programmed in `Fortran`, are currently being re-written in the modern, object-oriented programming language `C++`. Their new versions will be called `Pythia7` Pythia7 and `Herwig++` Herwig++, respectively. The decision to re-write them from scratch is based on two reasons:

First, new features and models concerning the simulation of particle physics at the shifting energy frontier need to be included. In fact this still is an on-going issue also for the `Fortran` versions (see for instance Moretti:2002eu,Sjostrand:2002ip).

Furthermore, and maybe more importantly, there is a wide-spread belief that the old `Fortran` codes cannot

easily be maintained or extended. On top of that, the software paradigm of the new experiments has already shifted to object-orientation, more specifically, to C++ as programming language. On the other hand, by the virtue of being decomposed into nearly independent phases, the simulation of high-energy particle reactions lends itself to modularisation and, thus, to an object-oriented programming style. In this respect it is also natural to further disentangle management and physics issues in event generation. In fact, both Pythia7 and Herwig++ will fully rely on the same management structure, called ThePEG ThePEG. It includes items such as the event record, mathematical functions, management functionalities, etc.. Using this common event-generation framework, Pythia7 and Herwig++ will just provide their respective, different modules for physics simulation, for instance the implementations of their hadronisation models.
In addition to these two re-writes of their older, Fortran-based counterparts, in the past few years a new event generator, called SHERPA, acronym for Simulation for High-Energy Reactions of PArticles, has been developed independently. From the beginning, it entirely has been written in C++, mainly due to the same reasons already named above. A number of paradigms have been the guiding principles in the construction of this code: enumerate
M odularity:
The goal of this publication is to give a brief status report of SHERPA's first α-version. It already incorporates enough functionality to make SHERPA a useful tool for a number of physics applications.
The outline of this paper is as follows: in Sec. Overall the overall generation framework is briefly introduced. This basically amounts to a discussion of how the framework and its physics modules are initialised, and how these modules are handed over to the actual event generation. Then, in the next two sections, Secs. Tools and Setup, general tools for event generation, including for instance the event record, are presented as well as those modules that specify the physics environment (such as the physics model, beam spectra, or parton distribution functions), in which the simulation is performed. In the following, the implementation of some of the event phases reflecting different physics features will be briefly highlighted. The discussion is commenced with describing the inclusion of hard matrix elements for jet production etc. (Sec. ME) and for heavy-particle decays such as, e.g., top-quark decays, (Sec. DEC) into SHERPA. Matrix elements are also needed for the simulation of multiple hard parton interactions in hadronic collisions. Hence, in Sec. MI a brief outlook will be given on how SHERPA will describe such phenomena. In all cases mentioned above, the matrix elements may give rise to configurations of jets to be fragmented by the subsequent parton shower. A cornerstone of SHERPA is the implementation of an algorithm, which merges matrix elements and parton showers respecting the next-to leading logarithmic accuracy of the parton shower (for details on this algorithm, see CKKW). In Sec. PS, questions related to the inclusion of this algorithm and the interplay with the parton shower inside the SHERPA framework are discussed. The quick tour through the event phases will be finished in Sec. Soft with a discussion of issues related to soft QCD, e.g. hadronisation, beam jets, etc.. Finally, in Sec. Finish, conclusions will be drawn and a further outlook will be given.

Overall event-generation frameworkOverall In SHERPA, the various tasks related to event generation are encapsulated in a number of specific modules. From a structural point of view, the set-up of the event-generation framework condenses into the problem to define the rules for the interplay of these modules and to implement them. The flexibility to do so is increased by a separation of the interfaces defining this interplay from the specific modules – the implementations of physics tasks Of course, this abstraction is to some extent limited by a kind of linguistic problem: in the implementation of the physics tasks, a choice has to be made on the terms in which the tasks are formulated. As a simple example consider four-momenta, clearly a basic ingredient of event generators. In ThePEG, the choice has been made to represent them as five-vectors, where the fifth component denotes the mass related to the four-momentum; in contrast, in SHERPA the representation is in terms of plain four-vectors. To use ThePEG modules within SHERPA requires a translation, which in SHERPA would be performed through the interface classes. The objects defining the terms in which physics tasks are implemented inside SHERPA are accumulated in a namespace ATOOLS, cf. Sec. Tools. Clearly, all other modules rely on these definitions. How this is realized within SHERPA can be exemplified by the hard matrix elements:
There are two implementations, which can be used to generate hard partonic subprocesses. One of them is restricted to a list of analytically known $2 \to 2$ processes, the other one is the multipurpose parton-level generator AMEGIC++. However different they are, in the framework of event generation they have to calculate

on these definitions. .

total cross sections for the hard subprocesses and they must provide single weighted or unweighted events. In SHERPA, these functionalities of both modules are accessible through an interface, the Matrix_Element_-Handler. It naturally lives up to the intrinsic differences in these physics implementations. Without knowing any details about the realization of hard matrix elements in the modules, they can be plugged anywhere into the event-generation framework by means of this abstract handler class. To add another module concerned with hard partonic subprocesses, on the level of SHERPA one would just have to extend the corresponding methods of the Matrix_Element_Handler accordingly. This reflects a typical object-oriented design principle. In general, such abstract handler classes encapsulate the specific physics implementations and are used to interface them with each other. Further examples that have been implemented so far include the Beam_Spectra_Handler, the ISR_Handler, the Hard_Decay_Handler, the Shower_Handler, the Beam_Remnant_Handler and the Fragmentation_Handler. They will be described in the forthcoming sections.
In many cases the underlying physics modules will require some initialisation before they can be used during event generation. Again, this can be exemplified by the hard matrix elements. In this case the initialisation basically consists of tasks like the set-up of matrix elements and phase-space integrators, and of the evaluation of total cross sections. They define the relative contributions of individual sub-processes in the overall composition of the hard process part inside the events. It is clear that such tasks have to be performed in an initialisation phase of an event-generation run. During this phase, SHERPA initialises the various physics modules selected by the user through the abstract handlers responsible for them. The specific set-up of a selected module will depend on external, run-specific parameters, which are read-in from corresponding data files and managed by the same handler class. The initialisation sequence of these handlers and their physics modules is organised by a SHERPA-internal Initialization_Handler, which also owns the pointers to the handlers. To add new handlers for completely new physics features, therefore, necessitates to modify and extend this Initialization_Handler.
Having initialised the interfaces to the physics modules, the SHERPA framework is ready for event generation. As already stated before, the individual events are decomposed into separate phases. This decomposition is reflected by SHERPA's program structure in the following way: an Event_Handler object manages the generation of one single event by having a list of various Event_Phase_Handlers acting on the expanding event record. This process of event generation is formulated in terms of particles connecting generalised vertices, coined blobs. These Blobs in turn reflect the space-time structure of the event, each of them has a list of incoming and outgoing particles. In other words, the blobs are the nodes, the particles are the connecting lines of a network. For a pictorial example, confronting a simple hadron–hadron event with its representation through Blobs, cf. Fig. blobs. figure[h] tabularcc [width=7cm]scetch$_1.eps[width = 8cm]scetch_2.eps$