Published by Institute of Physics Publishing for SISSA/ISAS
Received: *November 24, 2003*
Accepted: *February 23, 2004*

SHERPA 1.α, a proof-of-concept version

Tanju Gleisberg,[a] Stefan Höche,[a] Frank Krauss,[ab] Andreas Schälicke,[a] Steffen Schumann[a] and Jan-Christopher Winter[a]

[a] *Institut für Theoretische Physik, TU Dresden*
01062 Dresden, Germany
[b] *Theory Division, CERN*
CH-1211 Geneva 23, Switzerland
E-mail: tanju@theory.phy.tu-dresden.de, hoeche@theory.phy.tu-dresden.de, krauss@theory.phy.tu-dresden.de, dreas@theory.phy.tu-dresden.de, steffen@theory.phy.tu-dresden.de, winter@theory.phy.tu-dresden.de

Abstract: The new multipurpose event-generation framework SHERPA, acronym for Simulation for High-Energy Reactions of PArticles, 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.

Keywords: LEP HERA and SLC Physics, QCD, Phenomenological Models, Hadronic Colliders.

Contents

1. 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

JHEP02(2004)056

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.

These shortcomings necessitate other 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` [1] or `Herwig` [2], 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` [3] and `Herwig++` [4], 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 [5, 6]).

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` [7]. 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 `S`imulation for `H`igh-`E`nergy `R`eactions of `PA`rticles, 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:

1. *Modularity:* `SHERPA` only provides the framework for event generation. The physics issues related to the various phases of event generation are handled by specific, physics-

oriented modules. These modules, however, rely on a number of service modules that incorporate basic organisational, mathematical or physics tools, or information concerning the physics environment.

2. *Separation of interface and implementation:* Within SHERPA, the specific physics modules are interfaced through corresponding (handler) classes, which are sufficiently abstract to support an easy inclusion of other modules with similar tasks.

3. *Bottom-to-top approach:* Before the interfaces (abstract handlers) are implemented, the corresponding physics module has been programmed and tested. This is especially true for modules like AMEGIC++ [8], providing a full-fledged matrix-element generator for the evaluation of multi-particle production cross sections, or APACIC++ [9], hosting a parton shower module. In general, these modules can be used as stand-alone codes. They also can be implemented into other event-generation frameworks with minor modifications only, as long as some of the underlying mathematical and physics tools are supplemented as well.

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 section 2 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, sections 3 and 4, 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. (section 5) and for heavy-particle decays such as, e.g., top-quark decays, (section 6) into SHERPA. Matrix elements are also needed for the simulation of multiple hard parton interactions in hadronic collisions. Hence, in section 7 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 salient feature 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 [10]). In section 8, 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 section 9 with a discussion of issues related to soft QCD, e.g. hadronisation, beam jets, etc.. Finally, in section 10, conclusions will be drawn and a further outlook will be given.

JHEP02(2004)056

2. Overall event-generation framework

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.[1] 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 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

[1] 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. section 3. Clearly, all other modules rely on these definitions.

JHEP02(2004)056

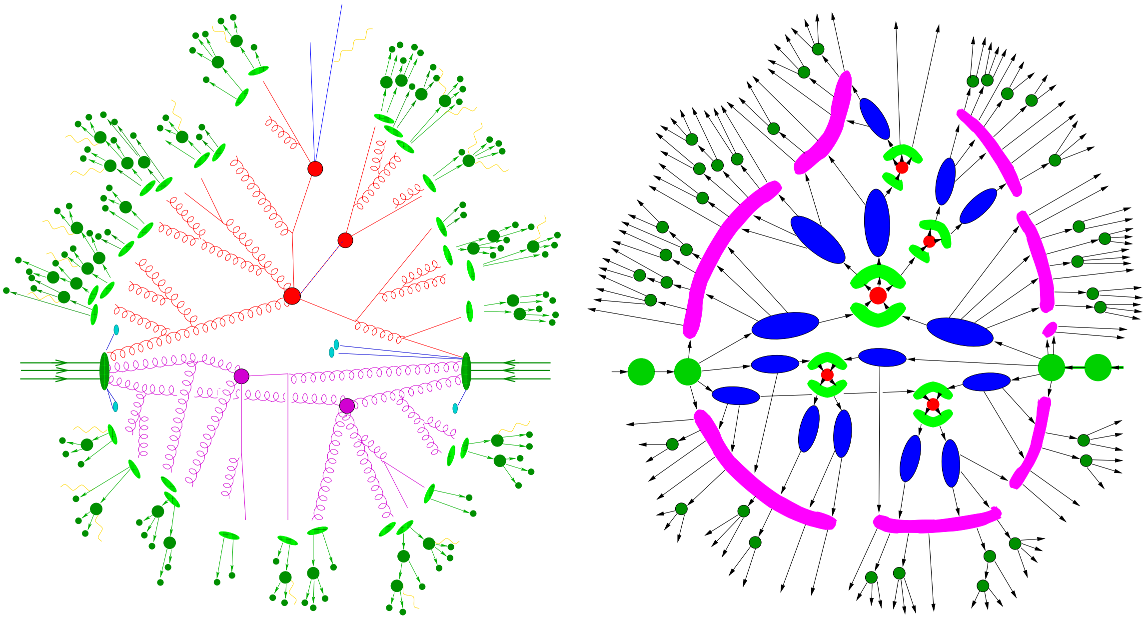

Figure 1: Pictorial representation of the event record. In the left picture, a hadron-hadron collision is exhibited. Clearly, apart from the hard signal subprocess followed by hard decays of two heavy unstable particles, it also contains two more hard parton interactions, all of them shown as thick blobs. The partons are dressed with secondary radiation as well, before the parton ensemble is transformed into primary hadrons which then decay further. On the right this is translated into the language of `Blob`s. Here, each hard matrix-element `Blob` (red) is equipped with merging `Blob`s (green) in the initial and final state which define initial conditions for the parton shower. All extra partons emitted during the shower evolution are combined in individual shower `Blob`s (blue). In the hadronisation `Blob`s (magenta) colour singlet chains formed by incoming partons are translated into primary hadrons which might decay further. Each such hadron decay is represented by an extra `Blob`.

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_Handler`s acting on the expanding event record. This process of event generation is formulated in terms of particles connecting generalised vertices, coined blobs. These `Blob`s 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 `Blob`s, cf. figure 1. An event thus can be represented as a list of `Blob`s, which in turn

JHEP02(2004)056

forms SHERPA's event record.[2] The Event_Phase_Handlers act on this list, by either modifying the Blobs themselves or by adding new Blobs or by subtracting unwanted ones. For event generation, the list of Event_Phase_Handlers is tried on the list of Blobs until no more action is possible, i.e. until none of the individual Event_Phase_Handlers finds an active Blob it can deal with. To illustrate this, consider the following simple example:

- First of all, a yet unspecified blob of the type *"Signal Process"* is added to the so far empty Blob list. Iterating with the list of Event_Phase_Handlers the Signal_-Processes phase deals with the single unspecified active Blob, inserting a number of incoming and outgoing partons through the Matrix_Element_Handler.

- In the next iteration of the Event_Phase_Handlers, the Jet_Evolution phase steps over this Blob and adds parton showers to it. To this end, some *"ME PS Interface"* Blobs are added as well as some Blobs for the initial- and final-state parton shower, signified by the types *"IS Shower"* and *"FS Shower"*, respectively. Assuming that an e^+e^- annihilation into hadrons is simulated, the *"IS Shower"* Blobs have one incoming and one outgoing electron each, and, maybe, some outgoing photons as well. The *"Signal Process"* as well as the *"ME PS Interface"* Blobs are switched to passive by this phase.

- The Hadronisation phase selects out the shower Blobs for the transition of partons into hadrons. First the Beam_Remnant_Handler has to fill *"Beam Remnant"* and *"Bunch"* Blobs. In the toy example, both, however, have a simple structure with one incoming and one outgoing electron each. Now, the Fragmentation_Handler comes into play, adding more blobs of the type *"Fragmentation"* with a number of incoming partons and a number of outgoing primary hadrons. All Blobs apart from the *"Fragmentation"* ones would be switched to passive now, leaving the outgoing primary hadrons to be decayed. These decays would be represented by more Blobs of the type *"Hadron Decay"*.

The structure elucidated above allows for nearly arbitrary mixtures in the composition of an event. For example, through the action of the Jet_Evolution phase the parton shower could in principle alternate with a sequence of hard decays on the parton level, or it could even be invoked in the decay of a heavy hadron.

In figure 2 the Event_Phase_Handlers implemented so far and their connections to various interfaces are exhibited.

[2] SHERPA also provides methods to translate its event record into two other formats, namely into the common-block structure of HepEvt used, e.g. by the Fortran-based event generators, and into HepMC [11], a C++-based event record. In fact SHERPA's event record resembles to some extent the latter one: Blobs of the former play the role of the GenVertex of the latter, and Particle and GenParticle are similar. This makes the translation between these formats a pretty simple task. Other formats, like for instance the LesHouches agreement [12] for the output of parton-level events generated through matrix elements are straightforward to add.

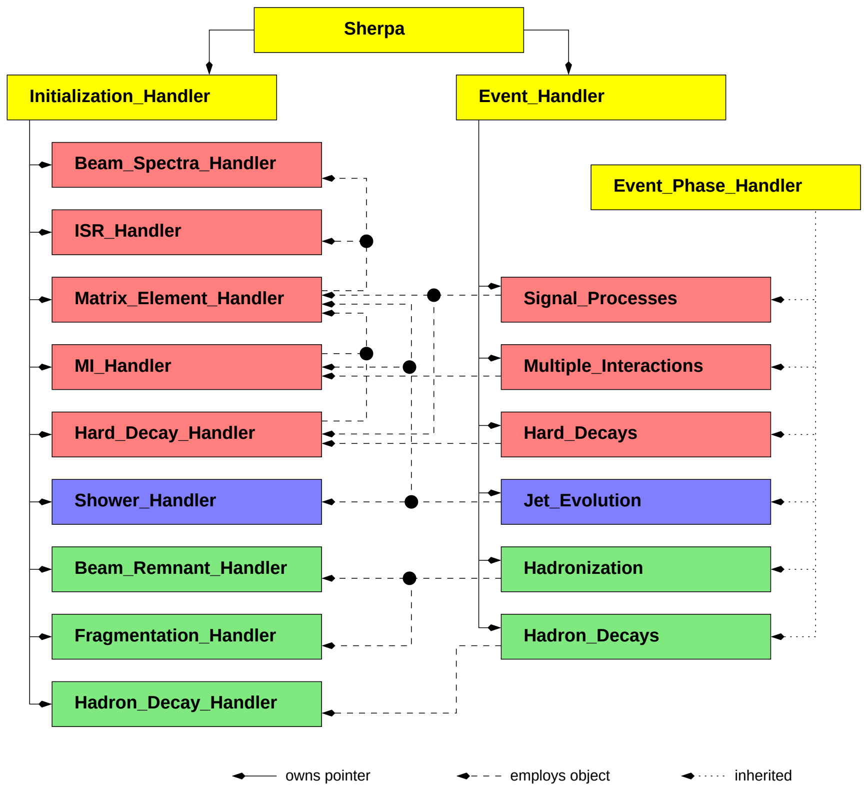

Figure 2: The `Event_Phase_Handlers` and their interfaces, all of which are implemented up to now in `SHERPA`.

3. Tools for event generation

In `SHERPA`, the basic infrastructure for event generation, which is used by other modules, is centralised in a separate package, called `ATOOLS`. It contains management, mathematics, and physics tools.

The organisational tools include, among others, classes to read-in input data, and to provide parameters and objects that must be globally accessible. During the initialisation of the `SHERPA` environment this data-container class is instantiated as a global object, which is filled and accessed by the other modules in due course. Therefore, if a potential user wants to include more objects that are needed in very separate corners of the total framework, he or she would have to include these objects into this class `Run_Parameters`. Of course, the corresponding access methods have to be provided there as well. `SHERPA` offers the possibility to specify a large amount of parameters for a run without recompiling. To enhance the transparency of the read-in procedure and to contribute to its intuitive understanding, the variables might be contained in different, user-specified data files in the following fashion:

```
KEYWORD = Value .
```

Within the code, default values can be given for the parameters connected to the keywords. An example defining, e.g. the physics model, and declaring the Standard Model as the default choice, reads:

```
Data_Read dataread(path,file);
std::string model = dataread.GetValue("MODEL",std::string("SM"));
```

In its instantiation, the dataread-object is given the path and the file name for the read-in procedure.

A second group provides mathematical service classes, including:

- a representation of three- and four-vectors;
- a class for real or complex matrices;
- a representation of Lorentz-transformations (boosts and rotations);
- abstract definitions of functions or grids which can be integrated or inverted;
- a class for simple histograms and operations on them;
- the random number generator.

This group of objects defines the mathematical terms in which SHERPA generates events.

The basic physics terms are also part of the ATOOLS package and cover a wide range of applications. In the following, some of the corresponding basic classes will be briefly described:

- Particles are described by some, in principle, unchangeable characteristics: their quantum numbers, their mass and width, etc.. All these properties are contained in a Flavour object. Within SHERPA, also pseudo-flavours, for instance "jet", are available. Hence, a Flavour object might serve as a container for other Flavours. In SHERPA the particles and their properties are collected in two data files, Particle.dat and Hadron.dat. A typical line in these files looks like:

kf	Mass	Width	3*e	Y	SU(3)	2*Spin	maj	on	stbl	m_on	Name
1	.01	.0	-1	-1	1	1	0	1	1	0	d

  Apart from the mass, width and spin, the electrical charge, the third component of the weak iso-spin, and the ability to participate in strong interactions are defined. In addition, for fermions, the user should provide information whether a specific Flavour describes Majorana particles or not. Also, information has to be provided, whether individual particles should be included at all, whether they are stable or not, and whether their mass should be taken into account in matrix-element calculations.[3] Finally, the particles' names should be defined as well in a form that will show up in the event record.

[3]It should be mentioned here that this mass enters in the phase space and in the propagators. For the Yukawa couplings these masses, if switched on, serve as default value, but can be overwritten during the initialisation of the physics models.

- In some cases, the user might wish to have, e.g., the matrix-element generator(s) to calculate the width of a `Flavour`, thus overwriting the one given in `Particle.dat`. To this end, another data file, by default called `Decays.dat`, might be read-in. Then, for the corresponding particles, decay tables are constructed and evaluated. They are implemented as `Decay_Table` objects.

- The particles, which finally show up in the generated event, are represented through a class `Particle`. In addition to the data objects specifying its properties, the `Particle`s are characterised by their four-momenta, by the vertices (`Blob`s) in which they are created or end, and by the flow of quantum numbers associated with them, such as colour.

In addition to the classes outlined above, the `ATOOLS` package includes classes which define some physics observables or which can be used to select events. These `Selector` classes are also needed for the integration over the phase space of the final state in hard subprocesses. One of them is providing a definition of jets according to the $k_\perp$- (or Durham-) algorithm [13] in various collision types. It is of special importance for the `SHERPA` package, since it is used for the merging procedure of matrix elements and the parton shower, an unique feature of `SHERPA`.[4]

4. Physics set-up

In this section those packages are presented that define the overall physics set-up. Clearly, this contains the specification of the physics model, in which cross sections are calculated or events are generated. Such a physics model defines the set of particles in it as well as most of their properties, including their mutual interactions. Equally important is a declaration of which type of process is discussed. Basically this amounts to a definition of incoming beams and their structures, both in terms of their respective energy spread and in terms of their eventual partonic substructure, which can be parametrised by parton distribution functions. In the following, therefore, the packages `MODEL`, `BEAM`, and `PDF` are briefly introduced. Within `SHERPA` they define the physics model, the structure of the incoming beams and the eventual inner structure of the colliding particles, respectively.

The package `MODEL` encapsulates abstract structures to specify arbitrary parameter sets of physical models, e.g. coupling constants, Yukawa masses, decay widths, etc.. For a certain physical model, for instance the Standard Model or its minimal supersymmetric extension, all parameters are represented by a `Model` object derived from the abstract base class `Model_Base`. This base class and its explicit instances mainly serve as containers and handle the input and the access to the parameters. The main ingredients of this class are lists of four standard parameter types:

- `ScalarNumber` for integer constants,

[4] In fact, recently first attempts into this direction within the framework of `Pythia` and `Herwig` have been reported [14]. The main difference with respect to `SHERPA`, however, is that in `SHERPA` the merging is achieved in one package for all types of processes.

JHEP02(2004)056

- ScalarConstant for floating point (double precision) constants,
- ScalarFunction for real single-parameter functions, derived from the abstract class ATOOLS::Function_Base, and
- ComplexMatrix for a matrix of complex floating point (double precision) constants.

Examples of parameters, which could be contained in the lists, are the number of extra dimensions, α in the Thomson limit, the running strong coupling constant α_s, and the CKM-matrix, respectively. Each parameter is mapped on a name string, which is used for all references on the parameter. A code example for the insertion of such a pair of name and parameter into the list of scalar constants reads

```
p_constants->insert(std::make_pair(std::string("ALPHAQED(0)"),
                                    1./137.03599976));
```

To access parameters, the class Model_Base defines a function for each parameter type, for instance the constant "ALPHAQED(0)" can be re-obtained through a call of

```
ScalarConstant("ALPHAQED(0)");
```

There are two typical situations for setting the parameters of a certain model. First, they can be simply read-in from a file, which by default is called Model.dat. As a second possibility, Model_Base is equipped with a pointer to a Spectrum_Generator_Base object. This object provides an abstract interface to external spectrum generators with methods to read-in input parameters, to deduce the particle spectrum and to calculate the other parameters of this model. So far, interfaces to the Fortran codes Hdecay [15] and Isajet [16] have been constructed. They are instances of the abstract base class Spectrum_Generator_Base and they are called Hdecay_Fortran_Interface and Isajet_-Fortran_Interface, respectively. To include more of these generators, a user would have to derive such an interface class and provide methods to read-in the input parameter set, to calculate the other parameters and to modify the particle spectrum accordingly. It should be noted that for the inclusion of new particles, also the class Flavour would have to be extended correspondingly.[5]

Within SHERPA the original beams of a specific collider are treated in two different stages in order to extract the partonic initial states for the hard interactions. In the first step, the incoming beams at a certain energy, the nominal energy of the collider, are transfered into bunches of interacting particles, which have an energy distribution, and whose momenta are distributed collinearly w.r.t. the original beams. Two options are currently implemented: the beams can either be monochromatic, and therefore need no extra treatment, or, for the case of an electron collider, Laser backscattering off the electrons is supported. This mode leads to photon bunches with a certain energy and polarisation distribution. In a second step, possible substructures of the bunch particles

[5]Using the new accord on a generic interface structure for spectrum generators, [17], the task to inherit new instances of the Spectrum_Generator_Base will be substantially alleviated.

are taken into account, as well as ordinary initial state radiation. This task is achieved by means of parton distribution functions (PDFs) or simple structure functions for the case of electron ISR.

As an illustrative example, consider the case of resolved photon interactions at an electron collider. As stated above, by Laser backscattering the incoming electrons can be "transformed" into photons distributed in energy and polarisation depending on the parameters chosen for the incoming electron beam and the Laser. This corresponds to the first step. In the second step, these photons have a partonic substructure described by an appropriate photon PDF defining the probability to find a certain parton flavour at the scale Q^2 and the energy fraction x inside the photon.

The first stage is hosted in the module `BEAM`, housing all classes that are employed to generate beam spectra. The handler class to access different beam-manipulation strategies is `Beam_Spectra_Handler`. Before coming into full effect during integration or event generation, this handler initialises a suitable treatment (`Beam_Bases`) for both beams and uses them to generate corresponding weights, i.e. energy distributions. At the moment, all outgoing bunch particles are still collinear to the incoming beams, but this is going to change in the future, by adding transversal boosts to the kinematics. Up to now two types of `Beam_Bases` are supported: `Monochromatic` beams, and the generation of photon beams via `Laser_Backscattering`. For the latter one the parametrisation of [18] is supplied in addition to a simple theoretical ansatz. To flatten out the peaks in the energy distribution of the produced photons, additional phase-space mappings have been introduced, which are located in the module `PHASIC++` and come to action as further channels in a multi-channel phase-space sampling [19] also implemented there. For more details, cf. section 5. To implement any new beam treatment, such as, e.g., Beamstrahlung, a corresponding instance of the class `Beam_Base` has to be provided. In addition, the construction of extra phase-space mappings might become mandatory.

The second stage, i.e. the handling of initial state radiation or partonic substructures, is located in the `PDF` module. The handler class steering the selection of PDFs or structure functions of bunch particles is the `PDF_Handler`, instantiating a suitable `PDF_Base` object and returning a pointer to it. So far, a structure function for electrons (that can handle charged leptons in general), a photon PDF and various proton structure functions are available. The list of proton PDFs covers: a `C++` version of MRST99 [20],[6] the `Fortran` CTEQ6 PDF [21], and the set of `LHAPDF`s [22]. The two `Fortran` pieces are encapsulated by the two classes `CTEQ6_Fortran_Interface` and `LHAPDF_Fortran_Interface`. For the case of photon bunches, the only structure function implemented is the GRV (LO) parton density [23], again framed by a C++ class, `GRVph_Fortran_Interface`. Having selected and initialised all required PDFs the `PDF_Base` objects are handed over to the `ISR_Handler` via pointers to two `ISR_Base` objects. If no ISR treatment is necessary for a beam the `ISR_Base` is instantiated as an `Intact` object, else a `Structure_Function` object is instantiated, which possesses a pointer to the corresponding `PDF_Base`. At first glance this construction looks quite over-engineered, however, it allows for a straightforward implementation of

[6]The mentioned C++ version has been written by Jeppe Andersen.

JHEP02(2004)056

possible multi-parton structure functions, which one would possibly like to use to correctly account for multiple interactions. To efficiently sample initial state radiation or parton distributions, and similar to the beam treatment, qualified phase-space mappings have been constructed, taking into account the peak structure of the corresponding distributions. It is also worth noting that the PDFs are handed over to the `Shower_Handler` in order to facilitate the backward evolution of initial-state parton showers, see section 8.

5. Matrix elements and phase space integration

In the `SHERPA` framework, hard matrix elements occur in different phases of event generation, i.e. in the generation of the (hardest) signal process, in the decay of heavy unstable particles, or during the simulation of multiple parton interactions. This is reflected by the appearance of different `Event_Phase_Handlers` during event generation. In fact, event generation starts with an empty list of blobs. The first blob to be filled by the `Signal_Processes` event phase is, obviously, for the partonic signal process. This event phase, like the other ones, such as `Hard_Decays` and `Multiple_Interactions`, owns a pointer to an appropriate handler for the matrix elements.

As briefly mentioned before, `SHERPA` currently incorporates two modules concerned with matrix elements for hard partonic subprocesses. These modules are interfaced through the `Matrix_Element_Handler`, which in turn possesses public methods for the set-up of the calculation framework (physics model, beam spectra, PDFs, construction of suitable, process- and framework-dependent integration channels), for the evaluation of total cross sections, and for the generation of single events. These tasks as well as some management issues (number and flavour of partons, etc.) look very similar on an abstract level, and in fact, the corresponding methods just call their counterparts in the specific matrix element realisation. There is one difference, however, in these modules. The analytically known $2 \to 2$ processes incorporated in the module `EXTRA_XS` provide the colour structure of individual parton configurations through specific methods. `SetColours` defines this structure in terms of the external four-momenta, whereas `Colours` returns the colour structure. In `AMEGIC++` things are not so easy. In fact, in `SHERPA` the colour structure of an n-parton configuration is reconstructed by backward clustering, which is guided by the individual Feynman diagrams, cf. section 8. This algorithm allows, in principle, to reconstruct colour flows for any multi-parton configuration in a leading-log large-N_c scheme for any parton level generator. The only ingredient that has to be delivered by the parton-level generators is a representation of Feynman diagrams in terms of binary trees. Therefore, `AMEGIC++` provides methods to access the amplitudes. This difference is also reflected in the `Matrix_Element_Handler`. It allows to either directly access the class responsible for the hard $2 \to 2$ subprocesses in the case of `EXTRA_XS` or to extract individual Feynman diagrams from `AMEGIC++`.

The library `EXTRA_XS` supplies a list of simple $2 \to 2$ processes at leading order and their analytically known differential cross sections. Thus it allows for a fast evaluation of such processes. At present it includes all $2 \to 2$ QCD and Drell-Yan processes with massless partons. Furthermore, it is employed for the determination of the initial colour configuration for the parton shower during event generation. When `AMEGIC++` is used as signal generator, this applies after an appropriate backward clustering, cf. section 8.

JHEP02(2004)056

Within EXTRA_XS each process object is inherited from the base class XS_Base, which contains the basic ingredients for a $2 \to 2$ signal generator. This amounts to methods providing the particle types, the total and differential cross section of the process, and to methods that allow the generation of single parton-level events and the determination of their colour structure. In the set-up of such an XS_Base the overall physics model, the beam spectra and the ISR strategy have to be handed over as well. The latter information is employed to select adequate initial state channels for the phase-space integration (see below). Since only $2 \to 2$ processes are taken into account within EXTRA_XS, its final state part boils down to simple hard wired S-, T- and U-channel integrators. According to its specific purpose, an XS_Base object may either correspond to a single $2 \to 2$ process represented by an instance of the class Single_XS or to a set of processes represented by the container class XS_Group. However, if a user wants to set up his own process, he or she has to derive it from Single_XS and to define all its process-specific properties, such as the colour structure of the particles involved, the differential cross section or the final state channels. The overall interface from EXTRA_XS to the SHERPA framework is the special XS_Group called Simple_XSecs, which can be accessed through the Matrix_Element_Handler and serves as a signal generator. This class also contains methods to read-in user-defined run-specific subprocesses and to select and initialise the corresponding XS_Bases.

AMEGIC++ is SHERPA's preferred multipurpose matrix-element generator concerned with the production and evaluation of matrix elements for hard processes in particle collisions at the tree-level. A manual for the current version 2.0 is in preparation, superseding an older one, [8]. This new version now also covers the full Minimal Supersymmetric Standard Model (MSSM) [24, 25] and the ADD model [26] of large extra dimensions; for details concerning the implementation of the latter one, see [27].

In its instantiation, AMEGIC++ is equipped with pointers to a Model_Base object, to a Beam_Spectra_Handler and to an ISR_Handler object. The first one supplies all coupling constants and model specific parameters that allow AMEGIC++ to construct a list of all available Feynman rules, i.e. vertices, for the chosen physical model. They are represented through objects of the type Single_Vertex, which possess pointers to a Lorentz_Function and a Color_Function object accounting for the intrinsic Lorentz and SU(3) colour structure of the vertex. This is nicely exemplified by the triple gluon vertex:

```
Kabbala kcpl0 = -g3;
Kabbala kcpl1 = kcpl0;

for (short int i=0;i<3;i++)
  vertex[vanz].in[i] = Flavour(kf::gluon);

vertex[vanz].cpl[0]        = kcpl0.Value();
vertex[vanz].cpl[1]        = kcpl1.Value();
vertex[vanz].cpl[2]        = 0.;
vertex[vanz].cpl[3]        = 0.;
vertex[vanz].Str           = (kcpl0*PR+kcpl1*PL).String();
```

```
vertex[vanz].ncf           = 1;
vertex[vanz].Color         = new Color_Function(cf::F);
vertex[vanz].Color->SetParticleArg(0,2,1);
vertex[vanz].Color->SetStringArg('0','2','1');

vertex[vanz].nlf           = 1;
vertex[vanz].Lorentz       = new Lorentz_Function(lf::Gauge3);
vertex[vanz].Lorentz->SetParticleArg(0,1,2);

vertex[vanz].on            = 1;
vanz++;
```

To extend AMEGIC++ and incorporate new interaction models, a potential user would just have to derive a corresponding class from the Interaction_Model_Base class and to fill it with suitable vertices.

Having specified a process or a group of processes to be evaluated, AMEGIC++ then constructs all Feynman diagrams by matching the set of vertices onto topologies generated beforehand. These amplitudes are translated into helicity amplitudes, which are subject of various manipulations, all aiming at a reduction of the calculational cost of the entire computation. As a further step AMEGIC++ analyses all individual Feynman diagrams and, according to their phase-space singularities, it automatically generates appropriate phase-space mappings for the integration over the final state. For more details on the multi-channel integration, see below. The integration channels as well as the helicity amplitudes are stored as library files that have to be compiled once and are linked to the main program. The by far most convincing features of the AMEGIC++ module are its robustness and flexibility. The package offers the evaluation of arbitrary processes[7] in the Standard Model, and in two of its extensions, the MSSM and the ADD model.

The tools for phase-space integrations, i.e. simple integration channels, building blocks for complex phase-space mappings and the full set of multi-channel integration [19] routines are hosted in the package PHASIC++. It is used by AMEGIC++ as well as by the simple matrix elements located in the EXTRA_XS package. If needed, it can be adjusted in a straightforward fashion for usage by any other matrix element generator. The only thing, one would have to do, is to provide information about or to directly construct the channels for the final state part. Both strategies are realized by EXTRA_XS and by AMEGIC++, respectively. In the latter case, the class responsible for the construction of the full final-state multi-channel integrator is the Phase_Space_Generator, individual channels are constructed by the Channel_Generator through a mapping of the Feynman diagrams onto the Channel_Elements supplemented by PHASIC++.

Apart from the matrix-element-specific final-state channels, during the phase-space integration one might have to sample over all initial-state configurations. Within SHERPA initial states on the parton level are constructed from the incoming beams in two steps.

[7] AMEGIC++ has proved to work for up to six final state particles [28].

JHEP02(2004)056

First, the beam particles might be transformed into other particles (such as electrons into photons through Laser backscattering) or may experience some smearing (such as electrons through Beamstrahlung). The resulting particles, which may or may not have an energy distribution, might have a resolved partonic substructure parametrised by PDFs or they might experience additional initial state radiation, which can also be parametrised by a PDF-like structure. To guarantee optimal integration performance, one has to analyse the emerging energy distributions in each of the two steps and flatten them out. This results in up to two more multi-channel mappings, one for the beam centre-of-mass system, and one for the parton-level centre-of-mass system. Both systems currently are defined through the boost relative to their ancestors and by their respective centre-of-mass energy squared. In the near future, also transversal boosts of the subsystems will be included. This, however, is a straightforward extension of existing code.

6. Decays of unstable particles

Decays of heavy unstable particles during the generation of an event are treated by a specific `Event_Phase_Handler` called `Hard_Decays`. This handler owns, not surprisingly, an interface to matrix elements responsible for the description of such decays on the parton level. Again, this interface, the `Hard_Decay_Handler`, is separated from the physics implementation, namely the matrix elements. Currently, only the matrix elements of `AMEGIC++` are accessible through this interface.

At the moment, heavy unstable particles are produced by hard matrix elements only, i.e. through the actions of the following event phases: `Signal_Processes`, `Hard_Processes` and `Multiple_Interactions`. While processing each of these phases, it is checked whether unstable particles emerge. If this is the case, their respective decay channel and the effective mass of this decay are determined. The decay channel is selected by invoking the `Hard_Decay_Handler`, which provides a mapping of particles to decay tables and the corresponding matrix elements for each decay channel. Hence, a pointer to this interface is a member of all the event phases above. The effective mass is distributed according to a Breit-Wigner function, the method for this resides in the `Particle` object itself. Fixing the decay channel before the mass is determined ensures that the correct, initialised branching ratios are recovered. In principle, this also allows for using tree-level decay kinematics as supplemented by, e.g., `AMEGIC++` together with higher order branching ratios.[8] After all masses are fixed, the four-momenta of all particles emerging in the corresponding hard subprocess (all particles leaving the blob) are shifted to their new mass-shell accordingly. This induces some minimal modifications of the energy-momentum relations of the particles and might affect the mutual respective angles. However, the four-momentum of the total system stays fixed. Eventually, after some jet evolution took place, the unstable particles are decayed, maybe giving rise to more unstable particles or new jets and, thus, triggering more actions of the `Hard_Decays` or `Jet_Evolution` phase.

[8] Such a procedure might seem somewhat inconsistent. However, using loop-corrections for, say, two-body decays, basically amounts to a specific choice of scale of the coupling constant(s) involved. In this sense, inconsistencies are due to different choices of scale, which could be fixed and compensated for in the corresponding vertices.

JHEP02(2004)056

At the moment, the procedure outlined above is being implemented and tested. In its current, minimal form two issues have not been tackled:

- In principle, attaching secondary radiation to hard decays leads to multi-scale parton showers [29], which act in the following way: In a first step the parton shower evolves the parton configuration down to scales comparable to the width of the decaying particles. Then, these particles decay, eventually starting an initial and a final state parton shower, which have to be matched with the preceeding one. Finally, the emerging parton ensemble is evolved down to the next decay or the infrared scale. An implementation of this procedure is straightforward in the `SHERPA` framework.

- Furthermore, spin correlations in the fashion of [30] should be added. The underlying idea is as follows. When decays of heavy unstable particles are treated in the way outlined above, implicitly some narrow width approximation has been used. In fact, this inherent assumption only allows to cut the propagators of the unstable particles.[9] Under the narrow width approximation, one can decompose the propagator into a sum over physical polarisation states. The polarisations of a number of outgoing particles produced in one interaction, however, are correlated, and this correlation propagates to a correlation in the kinematical distribution of the decay products.

7. Multiple interactions

Multiple interactions are handled within the `SHERPA` framework by the `Event_Phase_Handler` called `Multiple_Interactions`. Given a `Blob` list, which already contains the signal process, it adds one by one hard $2 \to 2$ subprocesses, according to an ordering in the transverse momentum $p_\perp$ of the outgoing particles. The initial conditions for this sequence of parton interactions are determined by the signal process. However, it might happen that the signal process contains more than two outgoing particles and, thus, the definition of $p_\perp$ is ambiguous. Then, the backward clustering already employed to create an interface from the signal process to the parton shower (see section 8) defines the corresponding $2 \to 2$ process. The sequence of further partonic $2 \to 2$ interactions results in new `Blob`s, each of which experiences its own shower evolution through the action of the `Jet_Evolution` event phase.

To create the additional hard subprocesses, the `Multiple_Interactions` phase employs a `MI_Handler`, the interface to the new module `AMISIC++`. This module is concerned with the generation of hard underlying events similar to how this is simulated in `Pythia` [31]. There, the hard underlying event is assumed to be a mostly incoherent sum of individual scattering processes. Right now, `AMISIC++` is restricted to hard QCD processes and therefore employs the library of `EXTRA_XS`, (see section 5). To account for a fast performance, however, `AMISIC++` does neither evaluate matrix elements on-line nor uses a veto algorithm

[9] In other words, if the decaying particles' width becomes large, all processes, i.e. also the "continuum" or background, contributing to the same final state have to be taken into account.

as proposed in [31]. Instead it pre-calculates and tabulates the appropriate differential cross sections and stores them to disk in the initialisation phase. This data may then also serve for future runs.

It should be noted here that AMISIC++ is in the process of full implementation and of careful tests only. Furthermore, the description of the soft underlying event is still lacking in Multiple_Interactions.

8. The interface to fragmentation

Having produced a number of partons in hard subprocesses — either the signal process, hard particle decays, or multiple hard partonic interactions — these coloured objects have to be transformed into colourless hadrons. The gap between the varying scales of these hard interactions and some universal scale connected to hadronisation is bridged by parton showers. Invoking the parton shower fills in further parton radiation and guarantees the universality of the scale, where the phenomenological hadronisation model sets in, and of its parameters.

Within the SHERPA framework, such additional emission in general happens during an event phase called Jet_Evolution. This event phase adds blobs describing radiation of secondary partons to the list of blobs constituting the event. To this end, all parton configurations in blobs for signal processes, hard decays, or for multiple parton interactions have to be analysed and modified by parton showers. The Jet_Evolution, thus, owns pointers to all corresponding Matrix_Element_Handlers for the definition of colour configurations and other starting conditions of the parton shower and to a Shower_Handler. This object provides public methods that allow to initialise and perform showers and to insert the resulting shower blobs into the event record. In principle, one can think of using different shower realisations, for instance a dipole cascade as in Ariadne [32], an angular ordered shower as in Herwig [2, 33], or a virtuality ordered shower as in Pythia [1]. So far, in SHERPA a virtuality-ordered shower has been implemented through a separate module called APACIC++ [9]. This module also includes the functionality needed for the merging of parton showers and matrix elements in the fashion of [10], i.e. a veto on jets at the parton level. The implementation of other approaches that model multiple emission of secondary partons will not substantially change the interface Shower_Handler.

From the brief description above, it is clear that the matrix elements and the parton showers might act on different objects. In the case realized so far, i.e. in the case of APACIC++, the parton shower is formulated in terms of trees and knots; for a shower described in the fashion of Ariadne one could imagine that dipole objects are the basic terms. Hence, in the case of APACIC++ being the parton shower generator the Jet_Evolution would have to administer the translation of partons to knots, i.e. the definition of a primordial tree structure representing a parton configuration. This is done through suitable interfaces. The specific instantiation of the abstract base class Perturbative_Interface depends on the form of the matrix elements and their functionality inside the Matrix_Element_Handler, and on the Shower_Handler itself. The application of these interfaces is mandatory for the Jet_Evolution and results in some "merging blobs" around the blob of the hard subprocess under consideration. These merging blobs are needed for the sake of four-momentum conservation, since secondary emission a posteriori gives a virtual mass to the primary on-shell partons, which has to be balanced by shifting the four-momenta of the primary parton ensemble. All of these interfaces are part of the `SHERPA` framework itself rather than of the individual modules (such as `AMEGIC++` etc.). Due to the merging algorithm, this interface needs to supply the possibility to calculate Sudakov weights, and to accept or reject parton configurations according to them. It is clear that a rejection necessitates a new parton configuration and, therefore, results in a new event to be supplied by the `Matrix_Element_Handler`. Correspondingly, a new `Blob` is filled by the `Signal_Processes` event phase. However, since at the moment only two specific matrix element generators are available, cf. section 5, only two realisations of the `Perturbative_Interface` exist, namely `SimpleXS_Apacic_Interface` and `Amegic_Apacic_Interface`.

The former is very simple, since the library of $2 \to 2$ subprocesses is used such that additional jets are the result of the simulation of the radiation activity through the parton showers. Therefore, in this case, no veto on extra jets has to be performed inside a shower and consequently no Sudakov form factor has to be applied. Furthermore, the colour structure of the partons as well as the hard scale of the subprocess can be obtained directly from the `XS_Base`s inside `EXTRA_XS` through simple access methods made available to the `SimpleXS_Apacic_Interface`. The starting conditions for the shower are obtained in quite a straightforward fashion. The initial virtualities for the shower evolution are given by the scale of the hard subprocess, which is connected to the maximal momentum transfer along coloured lines. The maximal opening angle of the next emission for each parton is obtained from the angles w.r.t. to the colour connected partons in the hard $2 \to 2$ process. The parton shower is then simply initialised by filling this information into the trees of `APACIC++`.

When using `AMEGIC++` or any other matrix element generator involving $2 \to n$ processes with $n > 2$ the situation is more complicated. In such cases, the $2 \to n$ configuration is reduced to a "core" $2 \to 2$ process through the $k_\perp$-cluster algorithm. To keep track of allowed and disallowed clusterings, i.e. of actual Feynman rules, the clustering follows the Feynman diagrams of the corresponding matrix element. They are obtained through the `Matrix_Element_Handler`. For each clustering, a Sudakov form factor is evaluated and attached as an extra weight (for details see [10]), which finally results in an overall weight for this specific parton-level event. In case it is accepted, the initial colour structure is fixed by the colour structure of the core $2 \to 2$ process, since the parton shower inherently is formulated in the large N_c-approximation. In the clustering procedure the tree structure for the parton shower already has been constructed, and, hence, the colour configuration of the partons has been fixed. It is supplemented with missing information (i.e. the starting virtualities for each parton, opening angles etc.) through the principle that the parton shower evolution of each parton is defined through the node in which it was produced first.

This condenses in the following algorithm: going from inner knots to the outer ones, in each node it is decided by the `Perturbative_Interface` which emerging parton is the harder, i.e. more energetic, one. The winner inherits the starting scale and angle of the decaying mother, the losers starting conditions are defined through the actual node. The starting conditions of the first four partons stem from the core $2 \to 2$ subprocess.

JHEP02(2004)056

As already stated, the interface to the showers and the actual physics implementation are separated. Whereas the interface is located in the `Shower_Handler`, the first physics implementation of a parton shower is encapsulated in the independent module `APACIC++`. It provides a virtuality ordered parton shower, supplemented with angular ordering enforced "by hand", similar to the one realized in `Pythia`. One of the major differences, however, is that in `SHERPA` matrix elements for arbitrary parton configurations are merged consistently with the parton shower. This merging procedure results in constraints on the parton shower, which must not produce any parton emission that would have to be interpreted as the production of an extra jet, since jet production is left to the corresponding matrix elements.

The parton shower in `APACIC++` is organised recursively in terms of binary tree structures, where the emission of an additional parton is understood as a branching process giving rise to another node, a `Knot`, inside the `Tree`.[10] In the evolution of the tree the binary branches are defined through splitting functions, which are represented by objects of similar name, i.e. by derivatives of the base class `Splitting_Function`. These objects contain methods to determine outgoing flavours of a branching process and their kinematics. Since in `APACIC++` the parton shower proceeds through a hit-or-miss method, functions overestimating the integral of a splitting function in certain boundaries and corresponding correction weights are also included. For the incorporation of new branching modes, such as for the simulation of parton showers off supersymmetric particles, just a suitable derivative of the base class has to be added. The sequence of branches within the parton shower is defined through Sudakov form factors. Consequently, such objects are also used by `APACIC++`. For the description of parton showers in the initial state, backward evolution relying on the parton distribution functions usually is employed. Therefore, the corresponding PDFs are handed over to `APACIC++` and used in the space-like showers and Sudakov form factors. Here, it should be briefly mentioned that the Sudakov form factors, in principle, provide only the trees of branching processes. There, each node is supplemented by the scale, where the branching takes place, and the distribution of energies. The corresponding determination of the actual kinematics is separated from the implementation of the Sudakov form factors; it is located in extra classes. However, once the parton shower has terminated, the tree structure is translated back into partons. The interface, i.e. the `Shower_Handler`, will provide blobs with one incoming parton stemming from the hard matrix element, which is identified as the jet's seed, and a number of outgoing partons exhibiting the partonic structure of the jet before hadronisation sets in.

9. Hadronisation & soft physics aspects

After the parton shower described above has terminated, one is left with a configuration of coloured partons at some low scale of the order of a few GeV in transverse momentum. These partons, in order to match experiments, have to be translated into white

[10]These trees are the only objects of `APACIC++`, which are handed over to the `Shower_Handler` in order to be filled with partons subject to further emission. This process is triggered by the `Shower_Handler` and managed by the `Hard_Interface`, the class managing the access to `APACIC++`

hadrons. Within SHERPA, this transition occurs in an event phase called Hadronisation. This Event_Phase_Handler contains interfaces to two physics tasks related to this phase.

First of all, extracting a coloured parton from a white initial hadron (such as in collisions involving protons) necessitates to describe the colour structure of its remnant. This is achieved by the Beam_Remnant_Handler.

It is clear that the coloured constituents will be colour connected to other partons in the final state, thus influencing properties of the event at hadron level. The distribution of colour over the hadron remnants is a tricky task, well beyond perturbation theory. This immediately implies that phenomenological models have to be employed. For instance, one could assume that such a model is guided by the attempt to minimise the string length of the colour string spanned by the outgoing partons. Therefore, within SHERPA the beam remnants arising from hadrons are currently handled in a naive approach. Given a list of Blobs, all initiators of initial state showers are extracted and attached to a beam blob, which represents the breakup of the incoming hadron. Beam-remnant partons are added such that the flavour quantum numbers of the hadron are recovered step by step. Colours are distributed in a randomised fashion, where, of course, gluons or quarks carry two or one colour index different from zero, respectively. Again, these indices are distributed such that they add up to a white hadron. The energies of the additional parton remnants are distributed either according to PDFs or to a phenomenological function like the one in [31]. Finally, all particles obtain a mild $k_\perp$-kick according to a gaussian distribution.

The resulting final parton configuration then originates from the perturbative event phases, i.e. from Signal_Processes, Hard_Decays, Multiple_Interactions or Jet_Evolution, or from the beam remnants as described above.[11] The Hadronisation phase has to translate these coloured partons into white hadrons. For this purpose, it employs its Fragmentation_Handler, which provides an interface to phenomenological hadronisation models.

The Fragmentation_Handler first of all sorts the partons into disconnected chains starting with a colour triplet, such as a quark, and ending with a parton in a colour anti-triplet state, such as an anti-quark. Then, within these chains, partons are shifted to their constituent mass-shells, if necessary. Only then, the selected individual hadronisation model is invoked. This mass-shift inside the Fragmentation_Handler guarantees the independence of the perturbative phase, which presumably is formulated in terms of current masses, and the non-perturbative phase with its constituent masses. Especially for cluster-fragmentation models [34] relying on the breakup of massive gluons into constituent quarks this is clearly advantageous. However, at the moment only the Lund string model [35] is implemented as a specific hadronisation model to be used by the Fragmentation_Handler. Its implementation within Pythia is accessible through a special Lund_Fortran_Interface class, which also reads in some of the parameters needed in this model from a corresponding data file. In the near future, also a new version of the cluster-hadronisation model [36] will be made available.

[11] Altogether these partons must form a colour singlet, although, if baryon-number violating sub-processes are implemented, it might be difficult to recover them as singlets in the large N_c-representation inherent to event generation.

JHEP02(2004)056

This model will be added as a new module, AHADIC++, to the overall framework. This module just finished construction and currently is being tested. It performs the transition from partons to primary hadrons in two steps: first of all, the gluons experience a forced decay into colour triplet pairs, which allows to decompose the parton singlet chain into clusters. The clusters are built from one triplet-anti-triplet pair and thus have the quantum numbers of hadrons, including those of baryons. In this step of cluster formation effects of soft colour reconnection are modelled, which is an extension to the previous versions of the cluster model [34]. In the next step, the clusters decay either into lighter ones, or into the primary hadrons. The respective decay mode depends on the cluster mass and on the masses emerging for the resulting four-vectors. The distribution of the decay products' momenta is governed by some universal anisotropic kinematics, the selection of the decay mode thus reflects a constituent-flavour-dependent separation into a cluster and a hadron regime. There, also soft colour reconnection effects are taken into account. In the rare case that a primary cluster already is inside the hadron regime a one-particle transition is enforced. For more details on this model, cf. [36].

In any case, invoking the Fragmentation_Handler results in a number of colour singlet parton chains, each of which enters a new Blob, producing a number of primordial hadrons. These hadrons may or may not decay further; at the moment, the subsequent hadron decays are also handled through the Lund_Fortran_Interface. In the future, however, it is envisioned to have an extra event phase Hadron_Decays and specific interfaces. Each of the hadron decays is then represented by another Blob, allowing to reconstruct displaced vertices etc..

10. Summary and outlook

In this publication a proof-of-concept version of the new event-generation framework SHERPA, Simluation for High-Energy Reactions of PArticles, has been presented in its version $1.\alpha$. Its construction is a still on-going process, which is based on three programming paradigms, namely modularity, the separation of interface and physics implementation and a bottom-to-top approach for the addition of further modules. In its overall structure, SHERPA reflects a typical, event-generator-inherent simulation of full events through disjoint event phases. This lends itself to modularisation and, therefore, SHERPA is entirely written in the object-oriented programming language C++.

So far a number of physics modules have been attached to SHERPA, which allow users to fully simulate electron-positron or unresolved photon-photon collisions at high energies. Also, fully hadronic collisions, such as, e.g., proton-anti-proton or proton-proton reactions, can be simulated. In the description of such events, however, some features, for instance the soft underlying event, are still lacking or basically not tested yet. In all cases considered so far, SHERPA proved to be flexible and to live up for all demands. More tests and the inclusion of further, nearly ready physics modules, such as a new version of the cluster hadronisation, hard decays of unstable heavy particles, or an underlying event model, will be in the focus of future work.

JHEP02(2004)056

SHERPA can be obtained through the downloads section of:

`http://www.physik.tu-dresden.de/~krauss/hep/index.html`

Acknowledgments

F.K. wants to acknowledge financial support by the EC 5th Framework Programme under contract number HPMF-CT-2002-01663. Further financial support by BMBF, DFG and GSI is gratefully acknowledged.

The authors are grateful for fruitful discussions with Stefan Gieseke, Klaus Hamacher, Hendrik Hoeth, Leif Lönnblad, Alberto Ribon, Gerhard Soff, Philip Stevens, and Bryan Webber. Also, the authors owe a great deal to users of SHERPA, which have reported on bugs and shortcomings, in particular Claude Charlot, Alessio Ghezzi, Hendrik Hoeth, Huber Nieto, and Thorsten Wengler. Without all this help such a task would be unsurmountable.

References

[1] T. Sjöstrand, *High-energy physics event generation with PYTHIA 5.7 and JETSET 7.4*, *Comput. Phys. Commun.* **82** (1994) 74;
T. Sjöstrand, L. Lönnblad, S. Mrenna and P. Skands, *PYTHIA 6.3 physics and manual*, hep-ph/0308153.

[2] G. Corcella et al., *HERWIG 6: an event generator for hadron emission reactions with interfering gluons (including supersymmetric processes)*, *J. High Energy Phys.* **01** (2001) 010 [hep-ph/0011363]; *HERWIG 6.5 release note*, hep-ph/0210213.

[3] L. Lönnblad, *Development strategies for PYTHIA version 7*, *Comput. Phys. Commun.* **118** (1999) 213 [hep-ph/9810208];
M. Bertini, L. Lönnblad and T. Sjöstrand, *Pythia version 7-0.0: a proof-of-concept version*, *Comput. Phys. Commun.* **134** (2001) 365 [hep-ph/0006152].

[4] S. Gieseke, A. Ribon, M.H. Seymour, P. Stephens and B. Webber, *HERWIG++ 1.0: an event generator for e^+e^- annihilation*, *J. High Energy Phys.* **02** (2004) 005 [hep-ph/0311208].

[5] S. Moretti, K. Odagiri, P. Richardson, M.H. Seymour and B.R. Webber, *Implementation of supersymmetric processes in the HERWIG event generator*, *J. High Energy Phys.* **04** (2002) 028 [hep-ph/0204123].

[6] T. Sjöstrand and P.Z. Skands, *Baryon number violation and string topologies*, *Nucl. Phys.* **B 659** (2003) 243 [hep-ph/0212264].

[7] *The PEG*, http://www.thep.lu.se/ThePEG/;
L. Lönnblad, *CLHEP: a project for designing a C++ class library for high-energy physics*, *Comput. Phys. Commun.* **84** (1994) 307.

[8] F. Krauss, R. Kuhn and G. Soff, *AMEGIC++ 1.0: a matrix element generator in C++*, *J. High Energy Phys.* **02** (2002) 044 [hep-ph/0109036];
A. Schälicke, F. Krauss, R. Kuhn and G. Soff, *Implementing initial state radiation for lepton induced processes in AMEGIC++*, *J. High Energy Phys.* **12** (2002) 013 [hep-ph/0203259].

[9] R. Kuhn, F. Krauss, B. Ivanyi and G. Soff, *APACIC++ 1.0: a parton cascade in C++*, *Comput. Phys. Commun.* **134** (2001) 223 [hep-ph/0004270].

JHEP02(2004)056

[10] S. Catani, F. Krauss, R. Kuhn and B.R. Webber, *QCD matrix elements + parton showers*, *J. High Energy Phys.* **11** (2001) 063 [hep-ph/0109231];
F. Krauss, *Matrix elements and parton showers in hadronic interactions*, *J. High Energy Phys.* **08** (2002) 015 [hep-ph/0205283].

[11] M. Dobbs and J.B. Hansen, *The HepMC C++ Monte Carlo event record for high energy physics*, *Comput. Phys. Commun.* **134** (2001) 41.

[12] E. Boos et al., *Generic user process interface for event generators*, hep-ph/0109068.

[13] S. Catani, Y.L. Dokshitzer, M. Olsson, G. Turnock and B.R. Webber, *New clustering algorithm for multi - jet cross-sections in e^+e^- annihilation*, *Phys. Lett.* **B 269** (1991) 432;
S. Catani, Y.L. Dokshitzer and B.R. Webber, *The k-perpendicular clustering algorithm for jets in deep inelastic scattering and hadron collisions*, *Phys. Lett.* **B 285** (1992) 291;
S. Catani, Y.L. Dokshitzer, M.H. Seymour and B.R. Webber, *Longitudinally invariant K(t) clustering algorithms for hadron hadron collisions*, *Nucl. Phys.* **B 406** (1993) 187.

[14] S. Mrenna and P. Richardson, *Matching matrix elements and parton showers with HERWIG and PYTHIA*, hep-ph/0312274.

[15] A. Djouadi, J. Kalinowski and M. Spira, *HDECAY: a program for Higgs boson decays in the standard model and its supersymmetric extension*, *Comput. Phys. Commun.* **108** (1998) 56 [hep-ph/9704448].

[16] H. Baer, F.E. Paige, S.D. Protopopescu and X. Tata, *ISAJET 7.48: a Monte Carlo event generator for PP, $\bar{p}p$ and e^+e^- reactions*, hep-ph/0001086.

[17] P. Skands et al., *SUSY les houches accord: interfacing SUSY spectrum calculators, decay packages and event generators*, hep-ph/0311123.

[18] A.F. Zarnecki, *CompAZ: parametrization of the photon collider luminosity spectra*, *Acta Phys. Polon.* **B34** (2003) 2741 [hep-ex/0207021].

[19] F.A. Berends, R. Pittau and R. Kleiss, *All electroweak four fermion processes in electron - positron collisions*, *Nucl. Phys.* **B 424** (1994) 308 [hep-ph/9404313];
R. Kleiss and R. Pittau, *Weight optimization in multichannel Monte Carlo*, *Comput. Phys. Commun.* **83** (1994) 141 [hep-ph/9405257].

[20] A.D. Martin, R.G. Roberts, W.J. Stirling and R.S. Thorne, *Parton distributions and the LHC: W and Z production*, *Eur. Phys. J.* **C 14** (2000) 133 [hep-ph/9907231];

[21] J. Pumplin et al., *New generation of parton distributions with uncertainties from global QCD analysis*, *J. High Energy Phys.* **07** (2002) 012 [hep-ph/0201195].

[22] *The LHAPDF interface*, http://vircol.fnal.gov/.

[23] M. Gluck, E. Reya and A. Vogt, *Photonic parton distributions*, *Phys. Rev.* **D 46** (1992) 1973.

[24] H.E. Haber and G.L. Kane, *The search for supersymmetry: probing physics beyond the standard model*, *Phys. Rept.* **117** (1985) 75.

[25] J. Rosiek, *Complete set of Feynman rules for the minimal supersymmetric extension of the standard model*, *Phys. Rev.* **D 41** (1990) 3464; *Complete set of Feynman rules for the MSSM — erratum*, hep-ph/9511250.

[26] N. Arkani-Hamed, S. Dimopoulos and G.R. Dvali, *The hierarchy problem and new dimensions at a millimeter*, *Phys. Lett.* **B 429** (1998) 263 [hep-ph/9803315].

JHEP02(2004)056

[27] T. Gleisberg et al., *Helicity formalism for spin-2 particles*, *J. High Energy Phys.* **09** (2003) 001 [hep-ph/0306182].

[28] T. Gleisberg, F. Krauss, C.G. Papadopoulos, A. Schälicke and S. Schumann, *Cross sections for multi-particle final states at a linear collider*, hep-ph/0311273.

[29] S. Gieseke, *Event generators: new developments*, hep-ph/0210294.

[30] P. Richardson, *Spin correlations in Monte Carlo simulations*, *J. High Energy Phys.* **11** (2001) 029 [hep-ph/0110108].

[31] T. Sjöstrand and M. van Zijl, *A multiple interaction model for the event structure in hadron collisions*, *Phys. Rev.* **D 36** (1987) 2019.

[32] L. Lönnblad, *ARIADNE version 4: a program for simulation of QCD cascades implementing the color dipole model*, *Comput. Phys. Commun.* **71** (1992) 15.

[33] S. Gieseke, P. Stephens and B. Webber, *New formalism for QCD parton showers*, *J. High Energy Phys.* **12** (2003) 045 [hep-ph/0310083].

[34] B.R. Webber, *A QCD model for jet fragmentation including soft gluon interference*, *Nucl. Phys.* **B 238** (1984) 492.

[35] B. Andersson, *The Lund model*, *Cambridge Monogr. Part. Phys. Nucl. Phys. Cosmol.* **7** (1997) 1.

[36] J.-C. Winter, F. Krauss and G. Soff, *A modified cluster-hadronization model*, hep-ph/0311085.

JHEP02(2004)056