Interoperability and complementarity of simulation tools for beamline design in the OASYS environment

Luca Rebuffi
Manuel Sanchez del Rio

Interoperability and complementarity of simulation tools for beamline design in the OASYS environment

Luca Rebuffi *[a], Manuel Sanchez del Rio[b]

[a]Elettra-Sincrotrone Trieste, Strada Statale 14, Km 163.5, 34149 Basovizza, Trieste, Italy;
[b]European Synchrotron Radiation Facility, 71 Avenue des Martyrs, 38000 Grenoble, France

ABSTRACT

In the next years most of the major synchrotron radiation facilities around the world will upgrade to 4th-generation Diffraction Limited Storage Rings using multi-bend-achromat technology. Moreover, several Free Electron Lasers are ready-to-go or in phase of completion. These events represent a huge challenge for the optics physicists responsible of designing and calculating optical systems capable to exploit the revolutionary characteristics of the new photon beams. Reliable and robust beamline design is nowadays based on sophisticated computer simulations only possible by lumping together different simulation tools. The OASYS (OrAnge SYnchrotron Suite) suite drives several simulation tools providing new mechanisms of interoperability and communication within the same software environment. OASYS has been successfully used during the conceptual design of many beamline and optical designs for the ESRF and Elettra-Sincrotrone Trieste upgrades. Some examples are presented showing comparisons and benchmarking of simulations against calculated and experimental data.

Keywords: optics, simulation, design, integration, oasys

1. INTRODUCTION

In the next few years several synchrotron radiation facilities around the world are, radically upgrading their performances. They evolve to 4th generation diffraction-limited storage rings (DLSR), mostly using the multi-bend-achromat technology [1]. Moreover, several free electron lasers (FEL) are ready-to-go or in phase of completion. These new radiation facilities represent not only a dramatic step ahead in the research perspectives, but also a huge challenge for the optics physicists responsible of the calculations designs and production of the new optical elements optimized to exploit the revolutionary characteristics of the new photon beams. Computer simulation of light sources and optical components is a mandatory step in the design and optimization of synchrotron and FEL radiation beamlines. In the synchrotron world, different codes for numerical simulations are available, implementing different physical approaches. While the real world set up is naturally unique, i.e. a X-ray source and a beamline, the API (application programming interfaces) and the graphical interfaces of the software are always different and typically can not be interchanged. In other words, they do not communicate in an easy way. It is worth noting that, although these API codes have the same goal of simulating X-ray beamlines they tend to be complementary because the physical models used are different: geometrical optics for ray tracing APIs (like SHADOW [2], RAY [3], XRT [4] and McXtrace [5]) and physical optics for wave propagation APIs (like SRW [6] and PHASE [7]). By considering the challenges of optical design for DLSR beamlines, it makes sense to use not only a single but also both of the physical approaches in a compatible mode, with also easy and efficient comparison of their results. This goes in the direction of standardization and definition of a common format and storage for synchrotron radiation data [8]. With this philosophy in mind, we developed in the last few years OASYS (OrAnge Synchrotron Suite) [9], a Python-based, open-source, high-level workflow environment based on Orange [10], a comprehensive, component-based framework for machine learning and data mining. Orange has been customized for synchrotron applications, becoming a graphical environment for simulating X-ray optics [11]. The ultimate goal of OASYS is to drive APIs with the aim to model a virtual experiment, starting from the description of the electron beam, then compute the radiation by the relativistic electron beams, then transport and propagate the photon beam trough the optical system and finally compute the radiation scattered by the sample and its detection.

*luca.rebuffi@elettra.eu; phone +39 375 8544; fax +39 040 9380902; www.elettra.eu

Advances in Computational Methods for X-Ray Optics IV, edited by Oleg Chubar,
Kawal Sawhney, Proc. of SPIE Vol. 10388, 1038808 · © 2017 SPIE
CCC code: 0277-786X/17/$18 · doi: 10.1117/12.2274232

In the initial phase of OASYS development, we first focused our attention on importing python-based APIs into its environment, but recently we dedicated our efforts on its final goal: to define a uniform and exchangeable description of the real world set up that is tailored to the synchrotron world and is still flexible enough to allow particularities for different algorithms and physical approaches. This is implemented by creating an object-oriented framework library, providing the glossary for the definition of light sources and optical components, together with a set of dedicated (so-called) widgets, the active elements of the OASYS graphical user interface (GUI) [11]. This common layer upon OASYS APIs has been called SYNED (SYNchrotron Elements Dictionary), and it is the base for allowing user to create workspaces with several APIs simulating the same beamline without data misalignment or redundancy, thus separating the description of the real world from details of the calculation algorithms and allowing users to easily benchmark their calculations by using different products for the same simulation.

After implementing these common definitions and data structures, we work on the creation of a software platform able to combine different APIs, by exchanging data and calculation results. The final purpose is to optimize running different calculations and making comparisons of the results. The first integration framework library we developed is dedicated to wave optics and it is called WOFRY (Wave Optics FRamework in pYthon), a generalization (or abstraction) of a wavefront propagation software tool, combined with the SYNED definition of a synchrotron radiation beamline.

Existing OASYS API like ShadowOui [11] and the wavefront propagation tool WISE [12] and XOPPY (XOP [13] in PYthon) has been refactored to be compatible with SYNED. The SRW OASYS prototype API has been implemented according to both WOFRY and SYNED frameworks. UML [14] class diagrams, implementation details and examples of both framework are shown in the following paragraphs.

2. THE SYNED PACKAGE

The class diagram of the main entities of SYNED is shown in Figure 1. The “Beamline” is the main entity, designed as the container of all the relevant information. “Beamline” is modeled as an aggregate entity containing a “Light Source” and a collection of “Beamline Element” entities, defined by an “Optical Element” and its positioning in the beamline or “Element Coordinates”.

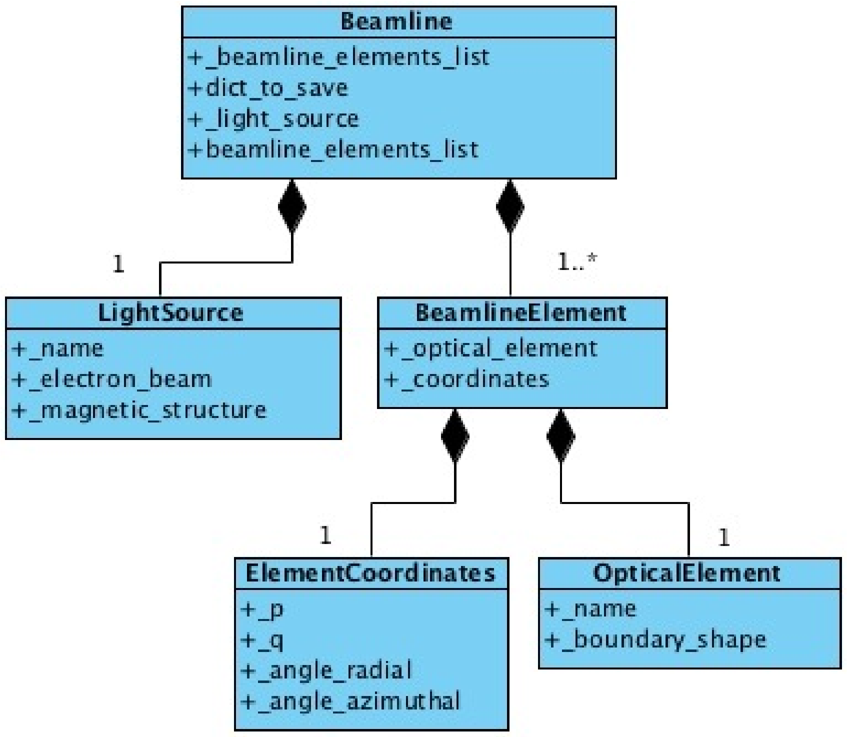


Figure 1. Class Diagram of the Beamline, the main entities of SYNED library.

The scope of this basic structure of objects is to represent the general layout of a synchrotron beamline, as in a blueprint, where no details of the light source or of the optical elements are present, but only their general characteristics and their position in space are shown.

The "Light Source" definition follows the SRW [6] objects layout, being composed by an "Electron Beam", containing all the specification of the light facility electron beam (like energy, current, spatial and angular parameters, etc.) and by a "Magnetic Structure", that contains the parameters defining the magnetic field of the photon source. The "Magnetic Structure" entity is specialized by subclasses describing the real magnetic structures: Bending Magnet, Undulator and Wiggler (both Insertion Devices), as visible in Figure 2.

"Optical elements" are described according to their nature: "Mirror", "Crystal", "Grating", absorbers ("Beam Stopper", "Filter", "Slit") and Ideal Elements ("Screen" and "Ideal Lens"). Compound Optical elements are not yet included, but, in principle, they are a collection of "Optical Elements", plus additional specification describing relative positions, periodicities, etc.

All "Optical Elements" (see Figure 3) contain the attribute "Boundary Shape", an entity describing the shape of their contour (e.g. rectangle or circular), while a specialized group of objects contains the attribute "Surface Shape", an entity containing the parameters able to define the shape of their surface (e.g., ellipsoidal, toroidal). In Figure 4 is visible the class diagram of the Shape class hierarchy, containing all the specialized shapes available to be associated to an Optical Element as a contour or a surface shape description.

The SYNED code repository can be found online in Ref. [15].

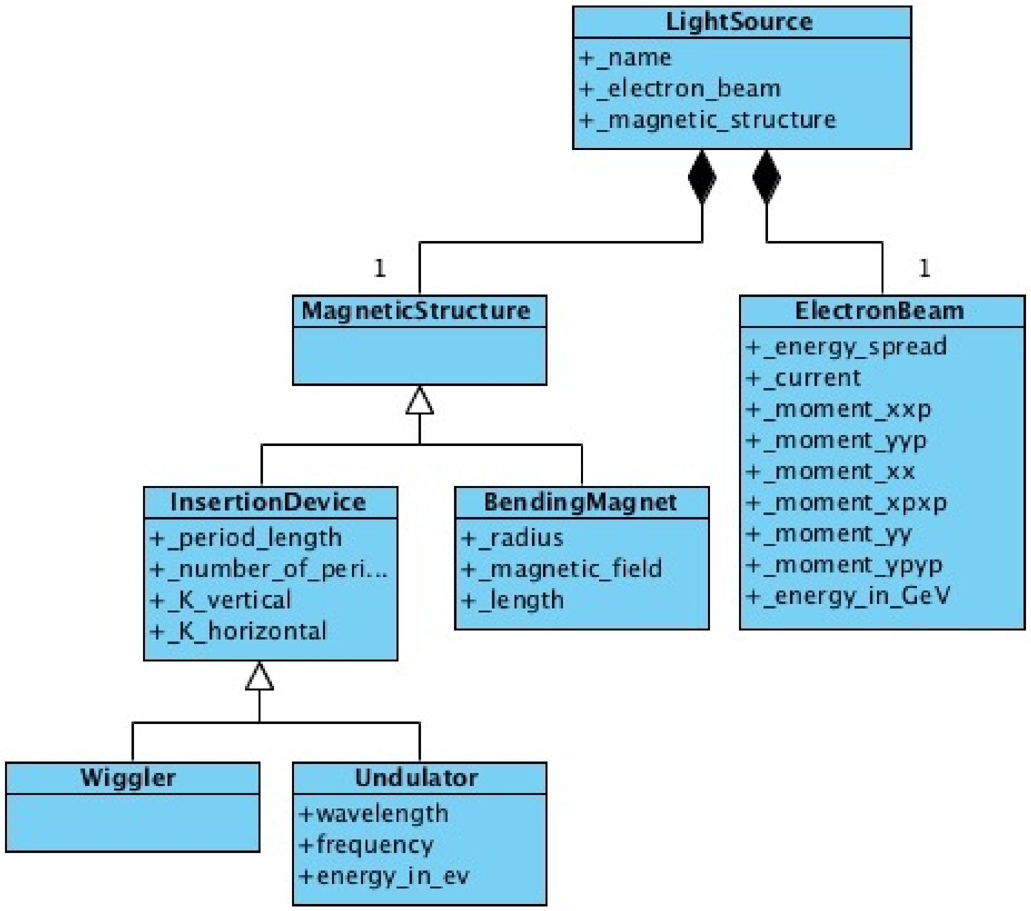


Figure 2. Class Diagram of Light Source entity of SYNED library.

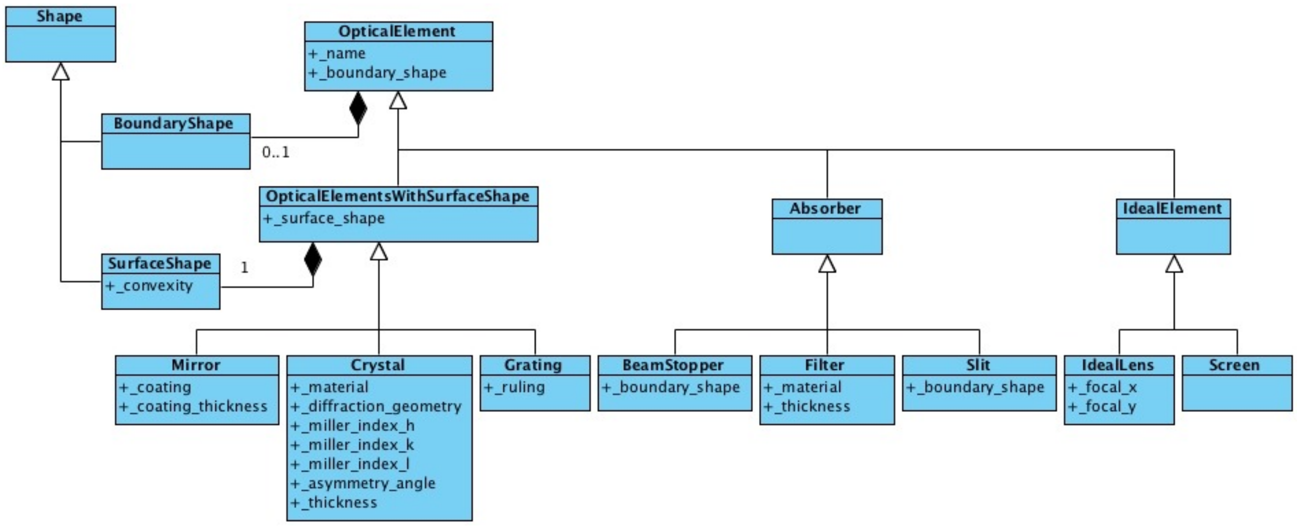


Figure 3. Class Diagram of the Optical Element entity in the SYNED library.

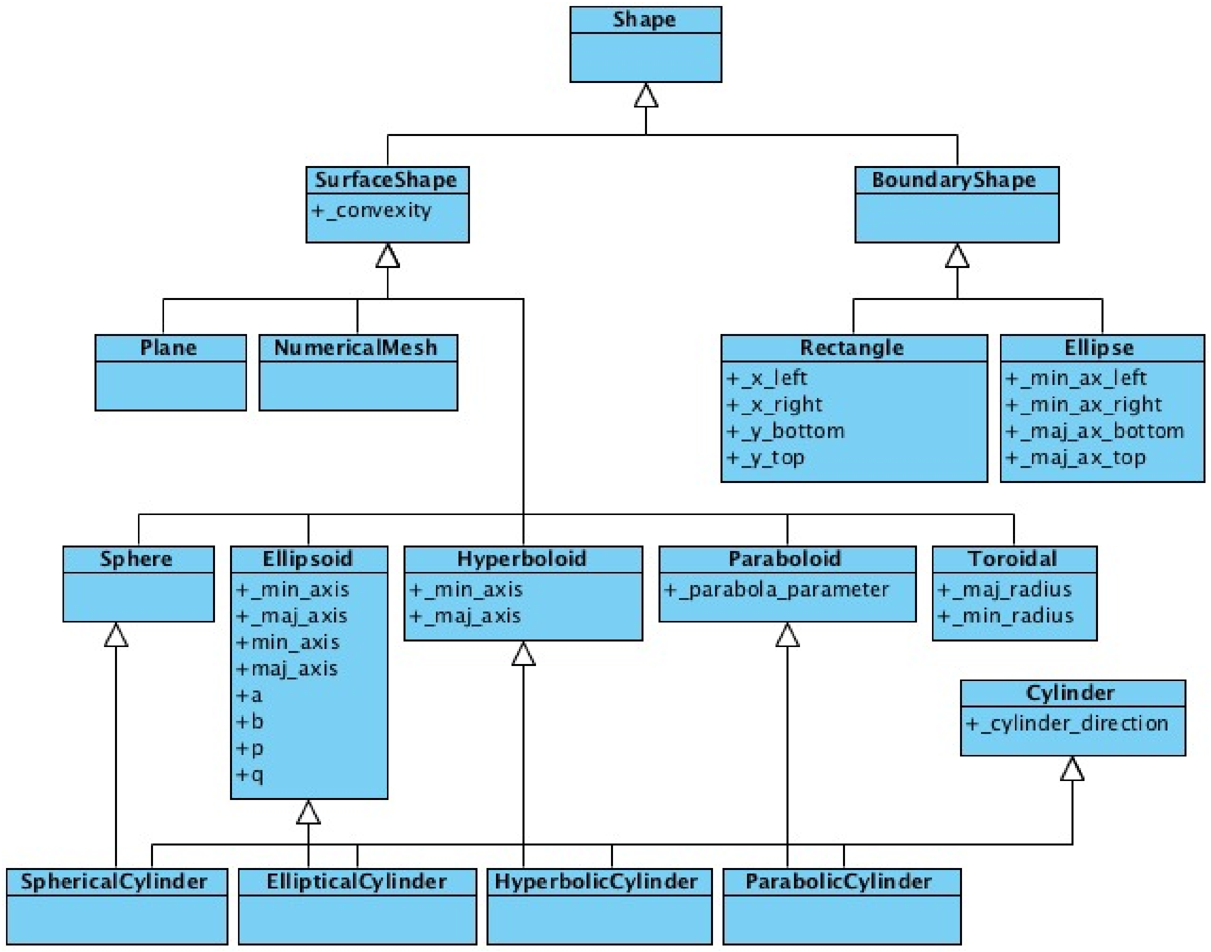


Figure 4. Class Diagram of the Shape entity in the SYNED library.

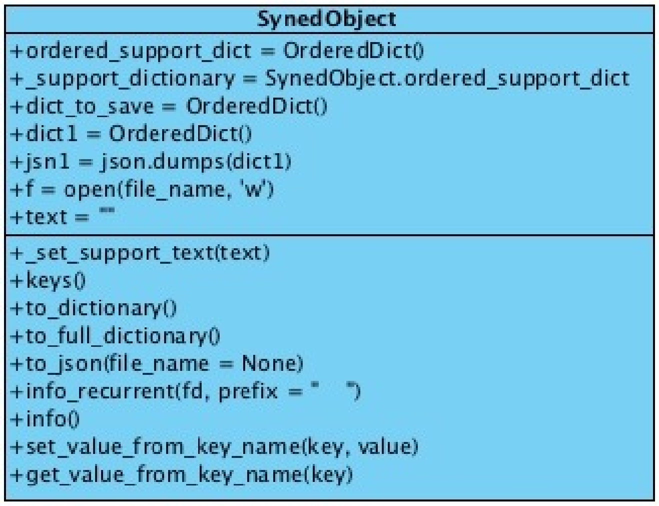
As an example, Figure 5 shows the code to create a cylindrical mirror with circular profile, together with its position on the beamline.

```
mirror = Mirror(name="mirror1",
                boundary_shape=Rectangle(x_left=-0.1,
                                         x_right=0.1,
                                         y_bottom=-0.6,
                                         y_top=0.6),
                surface_shape=SphericalCylinder(radius=2500.0,
                                                convexity=Convexity.UPWARD,
                                                cylinder_direction=Direction.TANGENTIAL),
                coating="Pt",
                coating_thickness=1e-4)

mirror_coordinates = ElementCoordinates(p=2.5,
                                        q=10.0,
                                        angle_radial=89.828,
                                        angle_azimuthal=180.0)

beamline.append_beamline_element(beamline_element=BeamlineElement(optical_element=mirror,
                                                                  coordinates=mirror_coordinates))
```

Figure 5. Example code of the SYNED library.

2.1 The JSON integration as exchange format

As usually happens in a object-oriented framework, all the main entities in SYNED derive their nature from a common abstract object (e.g. QObject in Qt) called SynedObject (see Figure 6). This kernel entity works not only as a main common data type, but contains the features able to import/export information to/from SYNED object and text files by using the JSON format.



SynedObject

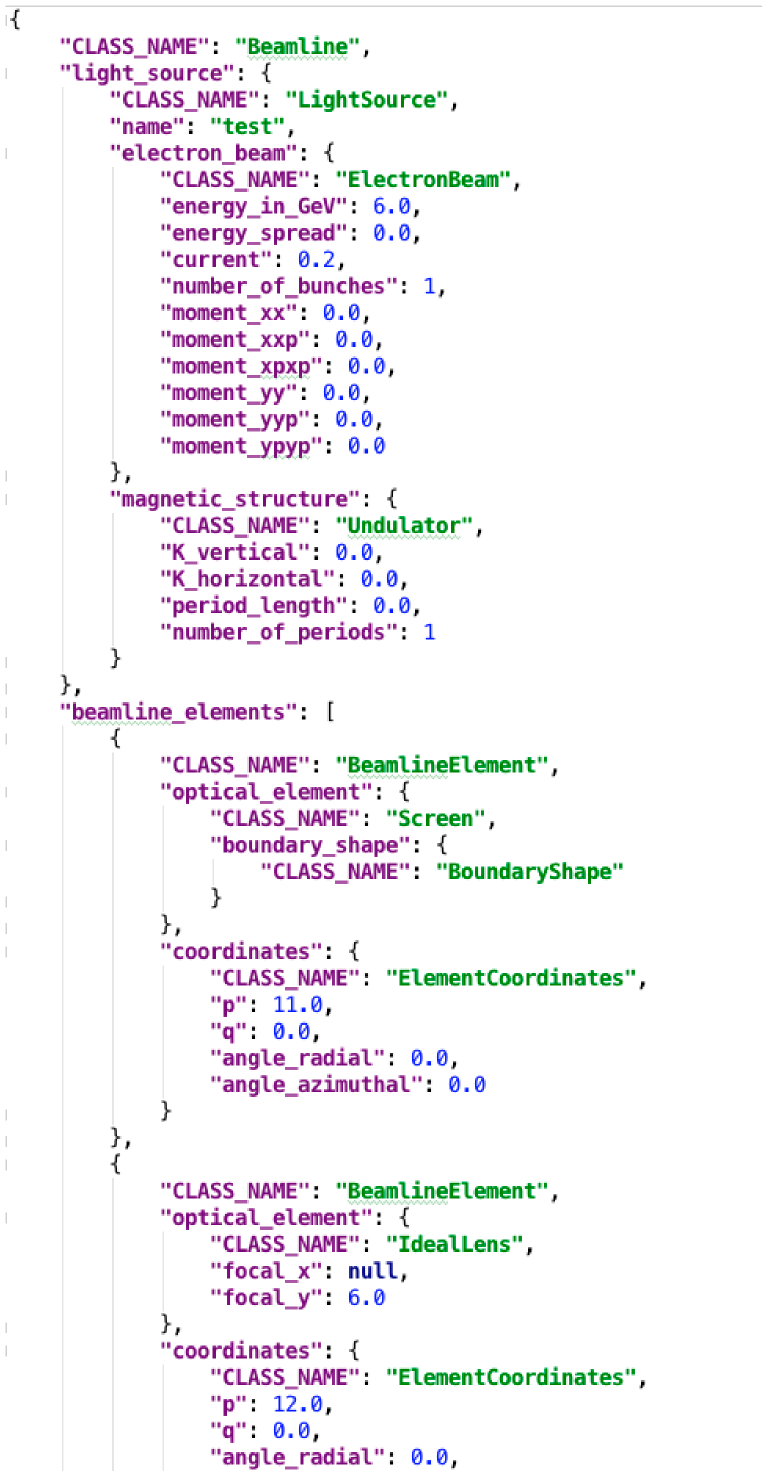
+ordered_support_dict = OrderedDict()
+_support_dictionary = SynedObject.ordered_support_dict
+dict_to_save = OrderedDict()
+dict1 = OrderedDict()
+jsn1 = json.dumps(dict1)
+f = open(file_name, 'w')
+text = ""

+_set_support_text(text)
+keys()
+to_dictionary()
+to_full_dictionary()
+to_json(file_name = None)
+info_recurrent(fd, prefix = "   ")
+info()
+set_value_from_key_name(key, value)
+get_value_from_key_name(key)

Figure 6. The "SynedObject" class, containing the import/export features through the JSON format.

JSON [16] has been chosen since it allows preserving the object-oriented nature of the described entities, by using a language-independent data format.

In Figure 7 and example of a file produced by calling the method "to_json" on a "Beamline" object, that recursively call the same method on its associate elements: the same file could be sent to a different SYNED user, to rebuild the same beamline. Moreover, JSON files with light sources or beamlines can be made available via internet, so the file name is replaced by the URL pointing to the remote file.



```json
{
    "CLASS_NAME": "Beamline",
    "light_source": {
        "CLASS_NAME": "LightSource",
        "name": "test",
        "electron_beam": {
            "CLASS_NAME": "ElectronBeam",
            "energy_in_GeV": 6.0,
            "energy_spread": 0.0,
            "current": 0.2,
            "number_of_bunches": 1,
            "moment_xx": 0.0,
            "moment_xxp": 0.0,
            "moment_xpxp": 0.0,
            "moment_yy": 0.0,
            "moment_yyp": 0.0,
            "moment_ypyp": 0.0
        },
        "magnetic_structure": {
            "CLASS_NAME": "Undulator",
            "K_vertical": 0.0,
            "K_horizontal": 0.0,
            "period_length": 0.0,
            "number_of_periods": 1
        }
    },
    "beamline_elements": [
        {
            "CLASS_NAME": "BeamlineElement",
            "optical_element": {
                "CLASS_NAME": "Screen",
                "boundary_shape": {
                    "CLASS_NAME": "BoundaryShape"
                }
            },
            "coordinates": {
                "CLASS_NAME": "ElementCoordinates",
                "p": 11.0,
                "q": 0.0,
                "angle_radial": 0.0,
                "angle_azimuthal": 0.0
            }
        },
        {
            "CLASS_NAME": "BeamlineElement",
            "optical_element": {
                "CLASS_NAME": "IdealLens",
                "focal_x": null,
                "focal_y": 6.0
            },
            "coordinates": {
                "CLASS_NAME": "ElementCoordinates",
                "p": 12.0,
                "q": 0.0,
                "angle_radial": 0.0,
```

Figure 7. JSON file produced by calling the method “to_json” on a “Beamline” object.

2.2 SYNED implementation into OASYS

SYNED library is fully integrated into OASYS, by providing a set of widgets able to create SYNED objects with its GUI. The basic idea is to be able to send the SYNED content to different applications, by wiring their widgets to SYNED widgets, with the final goal of inputting the common structural information once and keep all the wired application aligned. In Figure 8 a SYNED representation of a beamline with OASYS widgets is shown. An example of the input form of a Mirror is visible in Figure 9.

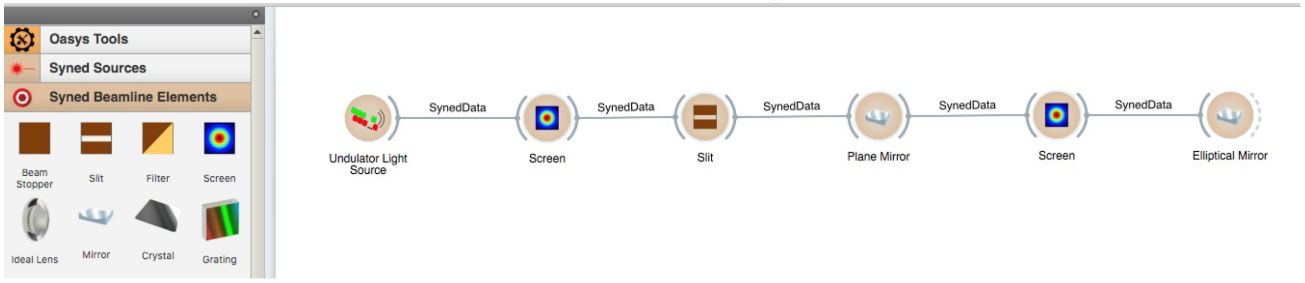
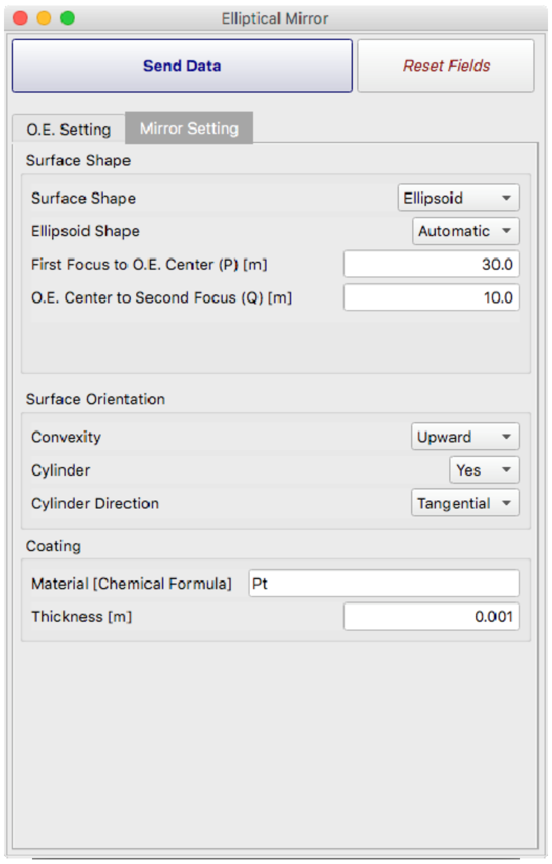


Figure 8. A complete beamline modeled with SYNED widgets in OASYS.



Surface Shape

Surface Shape	Ellipsoid
Ellipsoid Shape	Automatic
First Focus to O.E. Center (P) [m]	30.0
O.E. Center to Second Focus (Q) [m]	10.0

Surface Orientation

Convexity	Upward
Cylinder	Yes
Cylinder Direction	Tangential

Coating

Material [Chemical Formula]	Pt
Thickness [m]	0.001

Figure 9. Input form of a SYNED Mirror Widget.

In order to make the widgets of the other simulation tools compatible with SYNED data format, the library contains the interface "WidgetDecorator", containing methods to adapt the input setting of the widgets. It includes an abstract method "receive_syned_data", which must be implemented by the application receiver widgets (see Figure 10). The SRW, ShadowOui, WISE and XOPPY API in OASYS are compatible with SYNED. A complete example of the same simulation with different applications kept aligned by SYNED is shown in Figure 11.

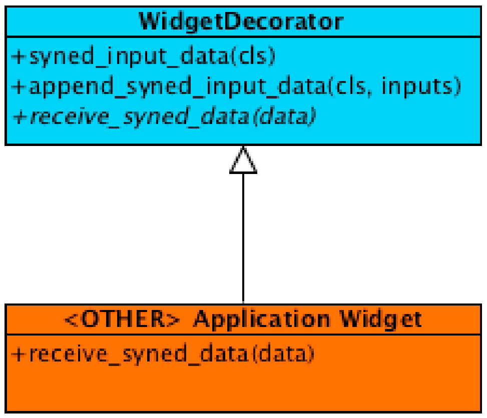

WidgetDecorator
+syned_input_data(cls)
+append_syned_input_data(cls, inputs)
+receive_syned_data(data)
<OTHER> Application Widget
+receive_syned_data(data)

Figure 10. The "WidgetDecorator" class, representing the abstract SYNED input receiver .

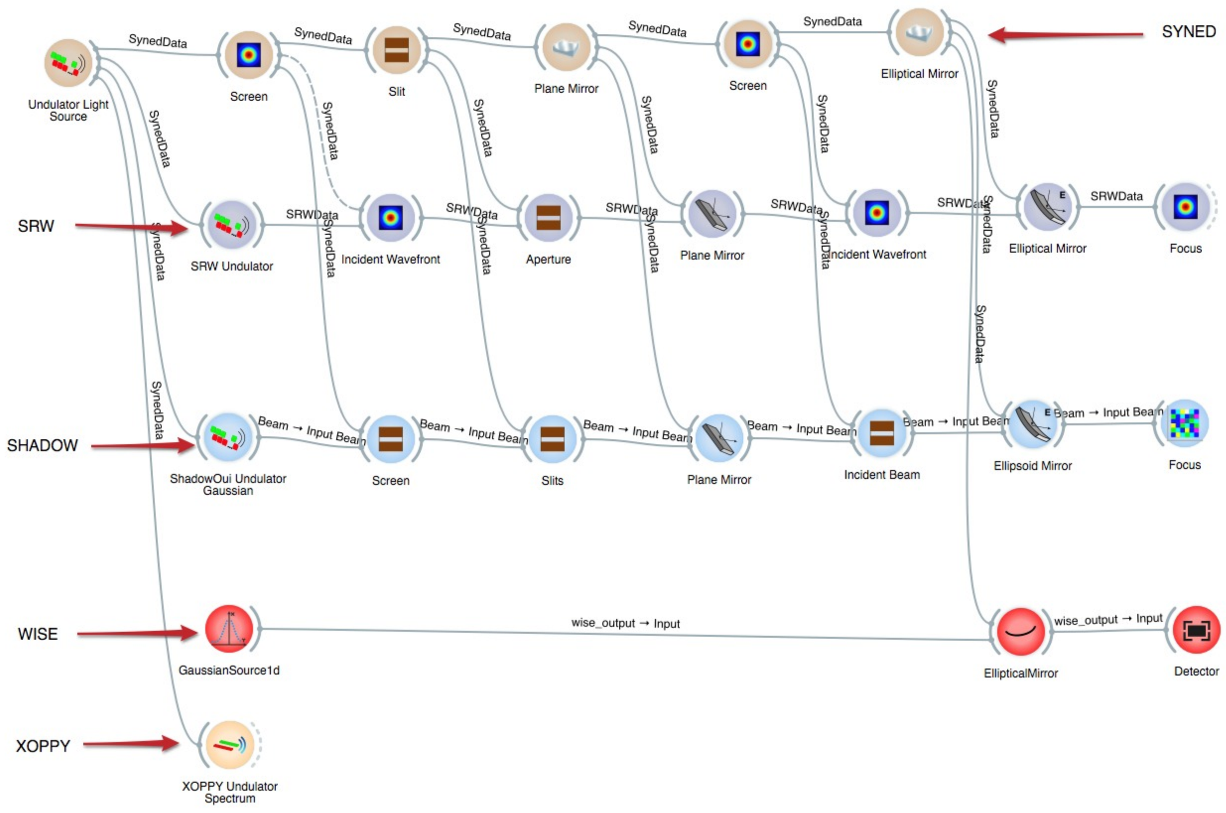


Figure 11. OASYS workspace of the same simulation with different applications kept aligned by SYNED.

3. THE WOFRY PACKAGE

WOFRY framework library is centered on the “Propagation Manager” and “Propagator” objects, representing the interface to the wavefront propagation algorithms, i.e. the representation of the wavefront propagation mechanism at the highest abstraction level (see Figure 12). In order to be as flexible as possible, the architecture of the classes has been developed according to “Template” and “Chain Of Responsibility” design patterns [17]:

- “Propagation Manager” is the manager of the chain of the wavefront propagation executors, specialized entities (once per wavefront propagation algorithm) of the abstract object “Abstract Propagator”, that contains the method “is_handler”, to be recognized as a handler of the propagation and the abstract method “do_propagation”, called by the “Propagation Manager” (see Figure 13). In order to ensure consistency to the mechanism, “Propagation Manager” has been implemented according to the “Singleton” design pattern, ensuring that just a single instance will be present in the application. As visible from the code, the actual chain of propagators has been indexed via a hashing mechanism according to the dimension of the wavefront (1D or 2D), avoiding the search of the propagation handler within unsuitable elements.
- The “Propagator” class, inherits from “Abstract Propagator” and represents the template of any specialized actual propagator, by giving the skeleton of the mechanism of wavefront propagation and deferring via the abstract methods the steps belonging to specialized real implementations (see Figure 14).

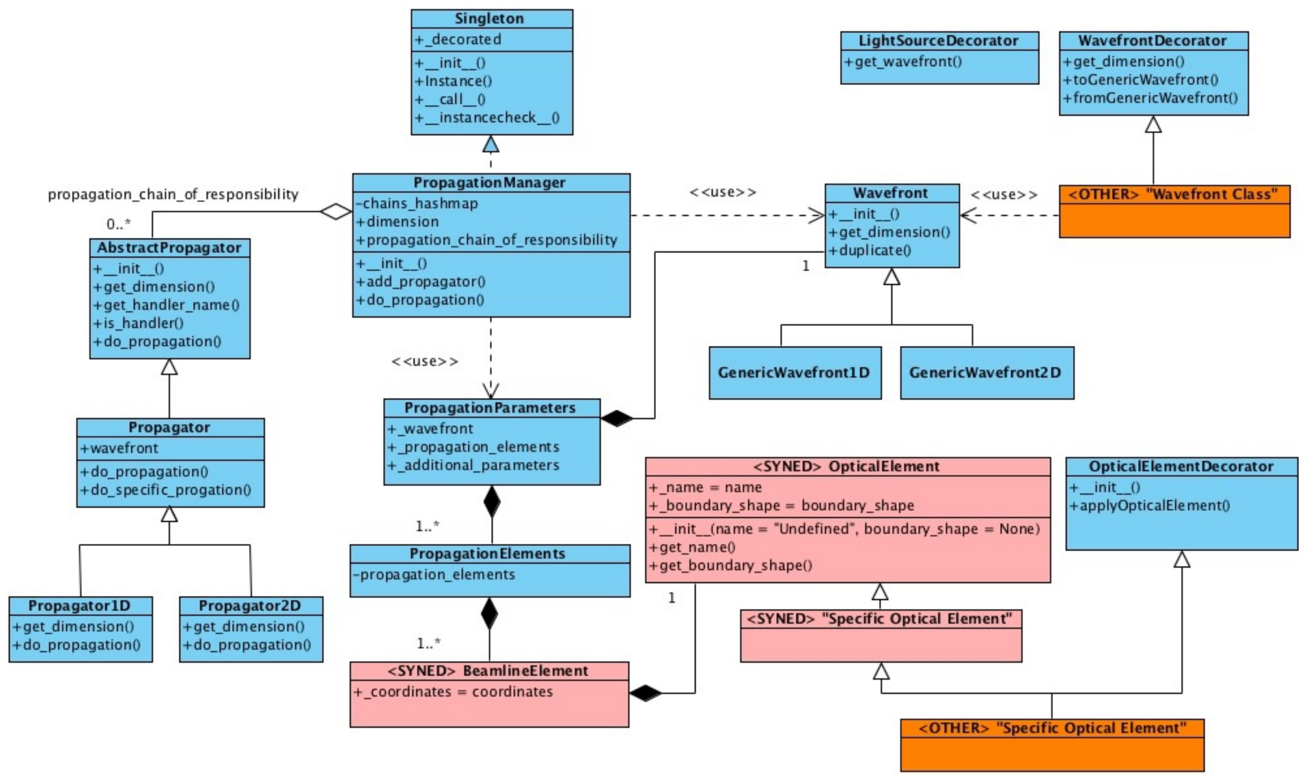
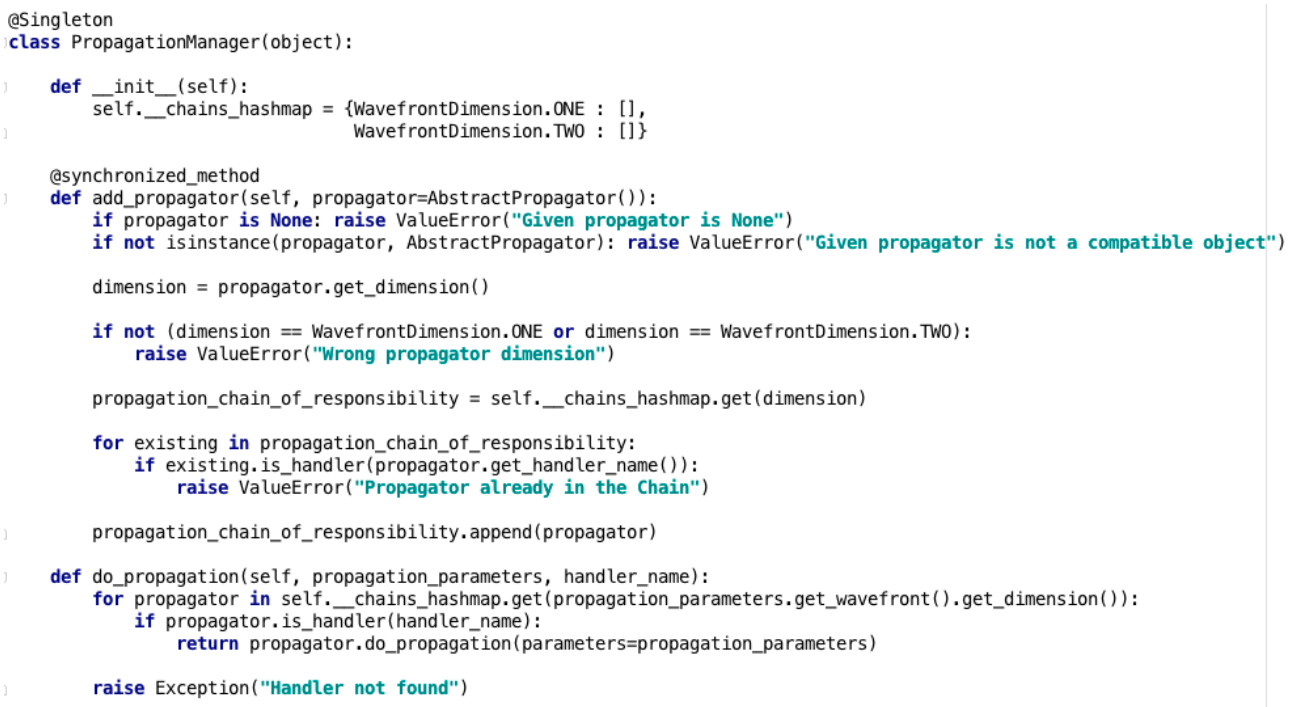
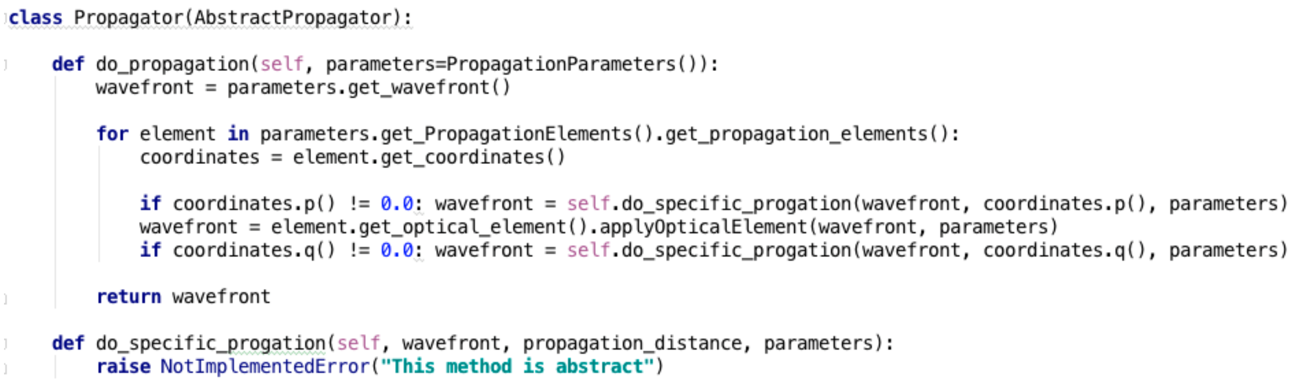


Figure 12. Class diagram of WOFRY framework.



```
@Singleton
class PropagationManager(object):

    def __init__(self):
        self.__chains_hashmap = {WavefrontDimension.ONE : [],
                                 WavefrontDimension.TWO : []}

    @synchronized_method
    def add_propagator(self, propagator=AbstractPropagator()):
        if propagator is None: raise ValueError("Given propagator is None")
        if not isinstance(propagator, AbstractPropagator): raise ValueError("Given propagator is not a compatible object")

        dimension = propagator.get_dimension()

        if not (dimension == WavefrontDimension.ONE or dimension == WavefrontDimension.TWO):
            raise ValueError("Wrong propagator dimension")

        propagation_chain_of_responsibility = self.__chains_hashmap.get(dimension)

        for existing in propagation_chain_of_responsibility:
            if existing.is_handler(propagator.get_handler_name()):
                raise ValueError("Propagator already in the Chain")

        propagation_chain_of_responsibility.append(propagator)

    def do_propagation(self, propagation_parameters, handler_name):
        for propagator in self.__chains_hashmap.get(propagation_parameters.get_wavefront().get_dimension()):
            if propagator.is_handler(handler_name):
                return propagator.do_propagation(parameters=propagation_parameters)

        raise Exception("Handler not found")
```

Figure 13. The WOFRY “PropagationManager” class code.



```
class Propagator(AbstractPropagator):

    def do_propagation(self, parameters=PropagationParameters()):
        wavefront = parameters.get_wavefront()

        for element in parameters.get_PropagationElements().get_propagation_elements():
            coordinates = element.get_coordinates()

            if coordinates.p() != 0.0: wavefront = self.do_specific_progation(wavefront, coordinates.p(), parameters)
            wavefront = element.get_optical_element().applyOpticalElement(wavefront, parameters)
            if coordinates.q() != 0.0: wavefront = self.do_specific_progation(wavefront, coordinates.q(), parameters)

        return wavefront

    def do_specific_progation(self, wavefront, propagation_distance, parameters):
        raise NotImplementedError("This method is abstract")
```

Figure 14. The WOFRY “Propagator” class code.

As shown in Figure 14, the template algorithm receives as input an object of type “Propagation Parameters”, an aggregate entity composed by a “Wavefront” and a collection of SYNED “Beamline Elements”. The optical element contained in the “Beamline Element” is an object that inherits its nature from a SYNED “Optical Element” (like “Mirror”, “Slit”, etc.), but must provide an additional WOFRY method containing the calculation of its effects on the wavefront. This mechanism has been designed as a “Decorator” design pattern, by providing the interface “OpticalElementDecorator”, that must be implemented by the specialized real optical elements objects (i.e. belonging to the specific API), as visible in Figure 12. A similar mechanism is designed for the sources of wavefronts, by providing the interface “LightSourceDecorator”.

The input/output entity of the propagation mechanism is the abstract object “Wavefront”: WOFRY library contains two implementations of 1D and 2D “Generic Wavefronts”, data structures able to store and manipulate the wavefront information (spatial range of definition, electric field and phase). Specialized propagators can directly use them, but they actually represent the generic definition of a wavefront, to which and from which every WOFRY specialized implementation can pass/derive its wavefront data. This "translation" mechanism can be realized by implementing a specialized "Wavefront" object, "decorating" it by inheriting the interface "WavefrontDecorator", containing the abstract method responsible of the translation from/to a "GenericWavefront".

This "Decorator" pattern and the generic data structures are the main actors in the integration mechanism between different propagators, belonging to different APIs., since the "Propagation Manager" can be invocated on a "Wavefront" coming from a first API, then the result could be converted in Generic and passed to a second API, again via the "Propagation Manager", where its specific propagator can initialize its specific wavefront by translating the generic one. This mechanism ensures a complete separation between different APIs that need to communicate, since the do not need to know each other details (and avoiding software maintenance nightmares)

The WOFRY code, completed with a basic implementation of itself by "decorating" SYNED optical elements and providing a basic set of propagators, and including a set of examples can be found online in Ref. [18].

3.1 SRW OASYS API implementation

The first OASYS API prototype fully implementing and built using both SYNED and WOFRY libraries drives SRW. This first version of the API contains Light Sources as Undulator and Bending Magnet and several optical elements: Aperture, Obstacle, Plane Mirror, Elliptical Mirror, Plane Grating, Screen and Ideal Lens. The code repository is available online in Ref. [19].

The API performs only Single Electron (SE) wavefront propagations. The SRW Multi Electron (ME) mechanism to deal with partial coherence is extremely powerful but it is as well time consuming (not available as interactive tool) and uses high amount of computing resources. For this reason ME calculation doesn't fit a single-user, desktop application like OASYS and, as a matter of facts, it won't be implemented into it. In any case, the SE simulation that can be easily designed by using the OASYS widgets is actually the template for the ME simulation and a widget able to automatically create the python script with the ME instructions will be provided in future. This script could be run in a proper environment, like a cloud computing facility or a computing cluster.

Figure 15 shows a complete simulation representing the Example 12 of the SRW distribution [20]. Figure 16 shows an example of an element input form, in this case the variable line spacing plane grating monochromator.

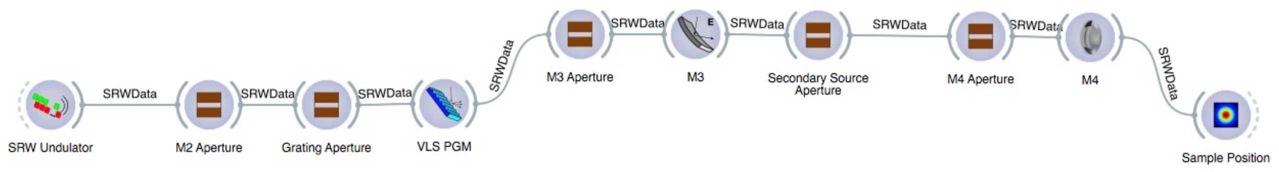
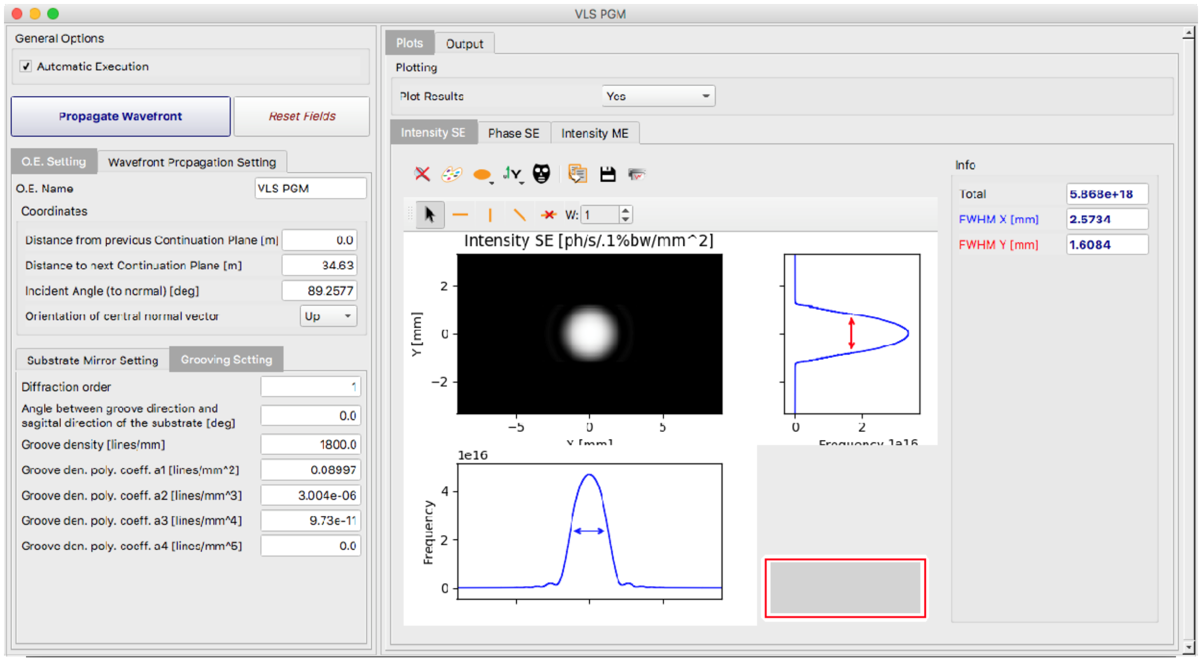


Figure 15. OASYS SRW API: Workspace containing the Example 12 of the SRW distribution.


VLS PGM

General Options

☑ Automatic Execution

Propagate Wavefront | Reset Fields

O.E. Setting | Wavefront Propagation Setting

O.E. Name	VLS PGM

Coordinates

Distance from previous Continuation Plane [m]	0.0
Distance to next Continuation Plane [m]	34.63
Incident Angle (to normal) [deg]	89.2577
Orientation of central normal vector	Up

Substrate Mirror Setting | Grooving Setting

Diffraction order	1
Angle between groove direction and sagittal direction of the substrate [deg]	0.0
Groove density [lines/mm]	1800.0
Groove den. poly. coeff. a1 [lines/mm^2]	0.08997
Groove den. poly. coeff. a2 [lines/mm^3]	3.004e-06
Groove den. poly. coeff. a3 [lines/mm^4]	9.73e-11
Groove den. poly. coeff. a4 [lines/mm^5]	0.0

Plots | Output

Plotting

Plot Results	Yes

Intensity SE | Phase SE | Intensity ME

Info

Total	5.868e+18
FWHM X [mm]	2.5734
FWHM Y [mm]	1.6084

Figure 16. OASYS SRW API: input form of a Plane Grating widget, showing with the plots of the output wavefront.

4. CURRENT AND FUTURE DEVELOPMENT

The mechanism of wavefront data exchange between different wavefront propagation API is under development. As already discussed, WOFRY "Generic Wavefront" acts as a common language and allows the proper decoupling between different APIs. Special widgets moving wavefront information from and to the common layer are provided, in order to let the user easily exchange data between different APIs. In Figure 17 an example of data exchange between SRW and WISE is shown.

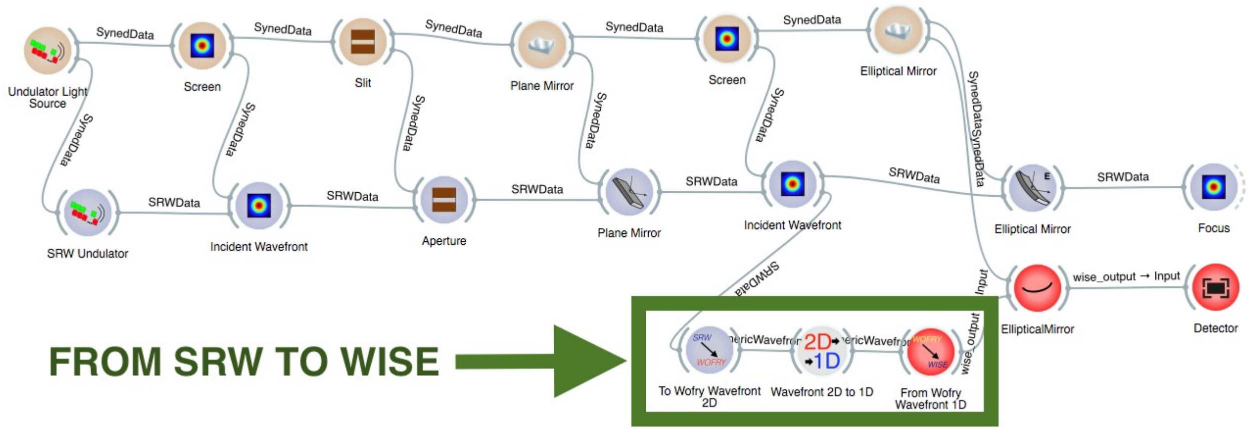


Figure 17. OASYS WOFRY API: example of data exchange from SRW to WISE.

These converters are presently under development and still under discussion. A possible mechanism to exchange and combine information between ray-tracing and wavefront propagation is being developed. The WOFRY integration mechanism is in place and it is necessary to find and implement physically consistent ways to create a wavefront from a beam of rays, and viceversa. It is worth to mention that the full conversion is not always possible: in fact, a ray-tracing beam is in general representing a beam of incoherent photons and a wavefront is, by definition, a single fully coherent wave. In some ideal cases or under some assumptions this conversion is possible and can be used to study particular effects in a beamline.

A possible scenario is visible in Figure 18, where the analysis of the effects of the height error profile is performed with three different APIs: ShadowOui, WISE and SRW. Such a transformation must carried out very carefully, since the wavefront is by definition totally coherent (and referred to a single electron, in the case of SRW), while the ray-tracing beam is kind of incoherent sum of several wavefronts, according to the electron beam spatial and angular characteristics.

Retrieving amplitude and phase of the wavefront from a ray-tracing beam, and sampling rays with spatial and angular distributions deduced from wavefront are, in principle, mathematically possible. It is however very important to analyze and understand under which condition these conversions have a physical meaning. For example, when one can guarantee that the photon beam has high degree of spatial coherence the conversion is supposed to work.

Finally, a future development will be a framework library conceptually identical to WOFRY for ray-tracing APIs, allowing again easy and fast integration between different APIs, exchanging data and results through a common layer.

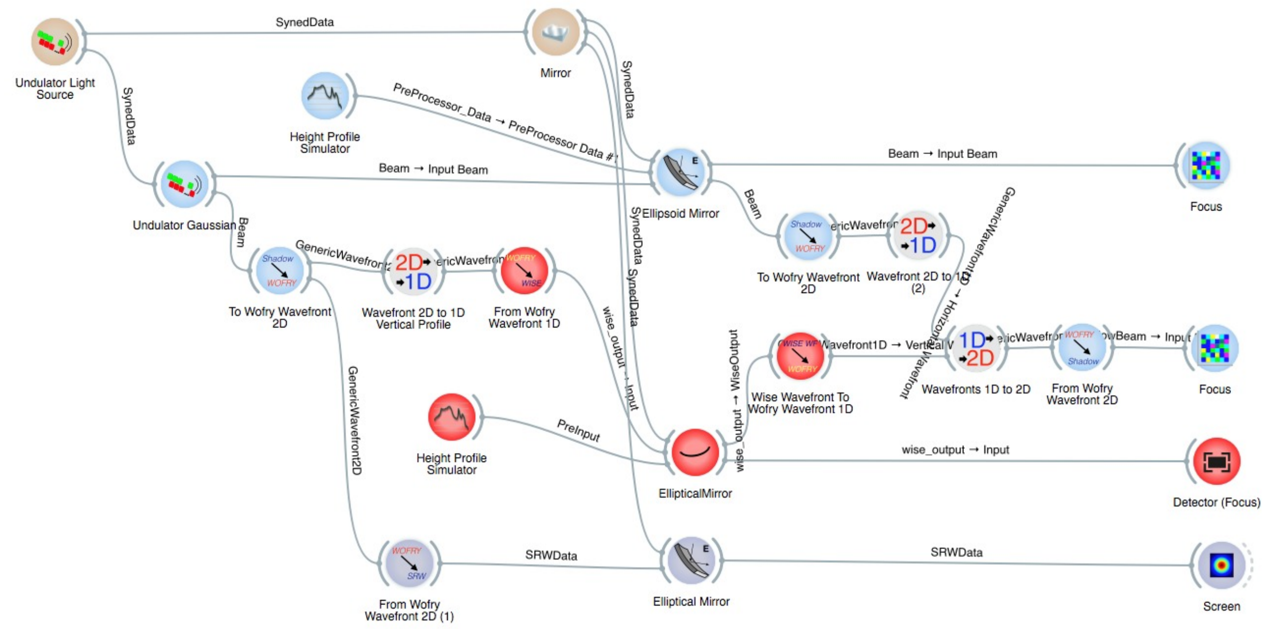


Figure 18. Example of data exchange between ray-tracing and wavefront propagation APIs via the WOFRY framework.

REFERENCES

[1] Eriksson, M., van der Veen, J. F. and Quitmann C., “Diffraction-limited storage rings - a window to the science of tomorrow,” J. Synchrotron Rad. 21, 837-842 (2014).

[2] Sanchez del Rio, M., Canestrari, N., Jiang, F. and Cerrina, F., "SHADOW3: a new version of the synchrotron X-ray optics modelling package," Journal of Synchrotron Radiation, 18, 708-716 (2011).

[3] Schäfers, F., [Modern Developments in X-Ray and Neutron Optics, Springer Series in Modern Optical Sciences], Springer-Verlag, Berlin Heidelberg, 137, 9 (2008).

[4] Klementiev, L. K., "A powerful scriptable ray tracing package xrt," Proc. SPIE 9209, 920909 (2014).

[5] Bergback Knudsen, E., Prodi, A., Baltser, J., Thomsen, M., Kjaer Willendrup, P., Sanchez del Rio, M., Ferrero, C., Farhi, E., Haldrup, K., Vickery, A., Feidenhans'l, R., Mortensen, K., Meedom Nielsen, M., Friis Poulsen, H.,

Schmidt, S. and Lefmann, K., "McXtrace: a Monte Carlo software package for simulating X-ray optics, beamlines and experiments," Journal of Applied Crystallography 46, 679-696 (2013).

[6] Chubar, O., Elleaume, P. "Accurate And Efficient Computation Of Synchrotron Radiation In The Near Field Region," Proceedings of the EPAC98 Conference, 22-26 June 1998, 1177-1179 (1998)

[7] Bahrdt, J., Flechsig, U., Grizolli, W. and Siewert F., "Propagation of coherent light pulses with PHASE," Proc. SPIE 9209, 920908 (2014).

[8] Sanchez del Rio, M., Rebuffi, L., Demšar, J., Canestrari, N. and Chubar, O., "A proposal for an open source graphical environment for simulating x-ray optics," Proc. SPIE 9209, 92090X (2014).

[9] www.elettra.eu/oasys.html

[10] Demšar, J., Curk, T., and Erjavec, A. "Orange: Data Mining Toolbox in Python," Journal of Machine Learning Research 14, 2349–2353 (2013).

[11] Rebuffi, L., Sánchez Del Río, M., "ShadowOui: a new visual environment for X-ray Optics and synchrotron beamlines simulations," J Synchrotron Radiat. 23(6), 1357-1367 (2016).

[12] Raimondi, L. and Spiga, D., "Mirrors for X-ray telescopes: Fresnel diffraction-based computation of point spread functions from metrology," Astron. Astrophys. 573, 1–19 (2015).

[13] Sanchez del Rio, M. and Dejus, R. J., "Status of XOP: v2.4: recent developments of the x-ray optics software toolkit," Proc. SPIE 8141, 814115 (2011).

[14] http://www.uml.org

[15] https://github.com/lucarebuffi/syned

[16] http://www.json.org

[17] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., [Design patterns: elements of reusable object-oriented software], Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (1995)

[18] https://github.com/lucarebuffi/wofry

[19] https://github.com/lucarebuffi/wofrysrw

[20] https://github.com/ochubar/SRW