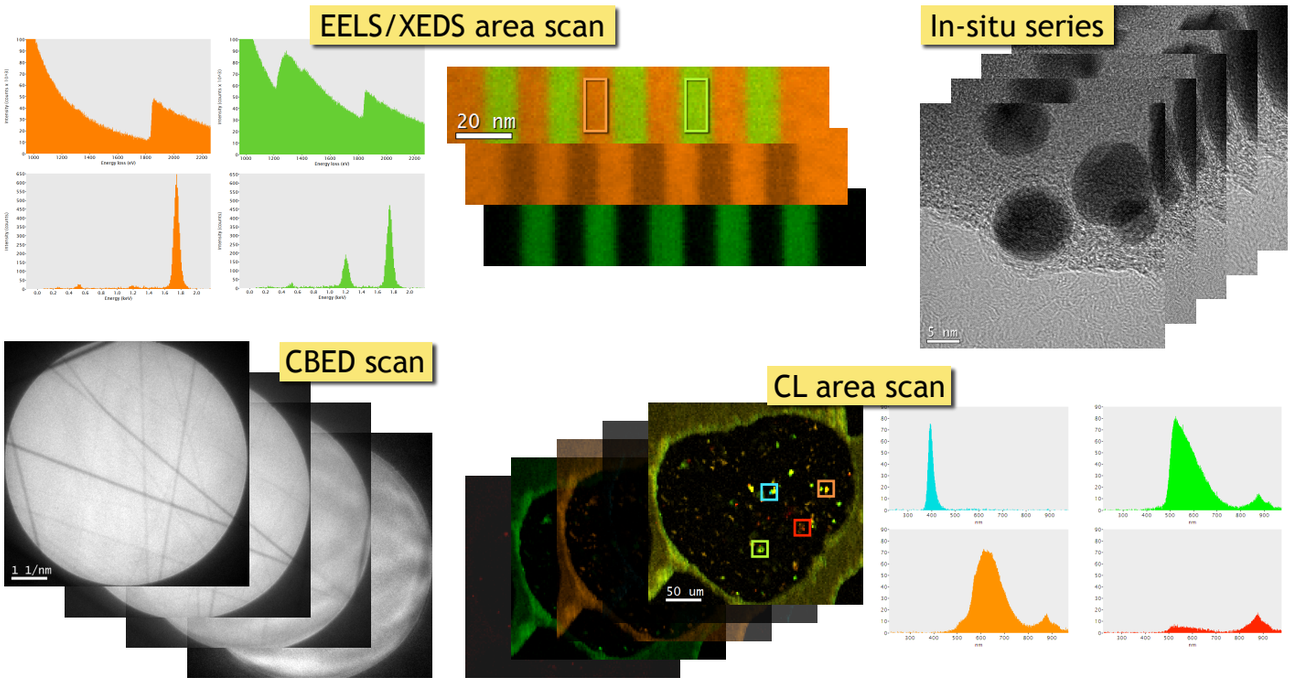
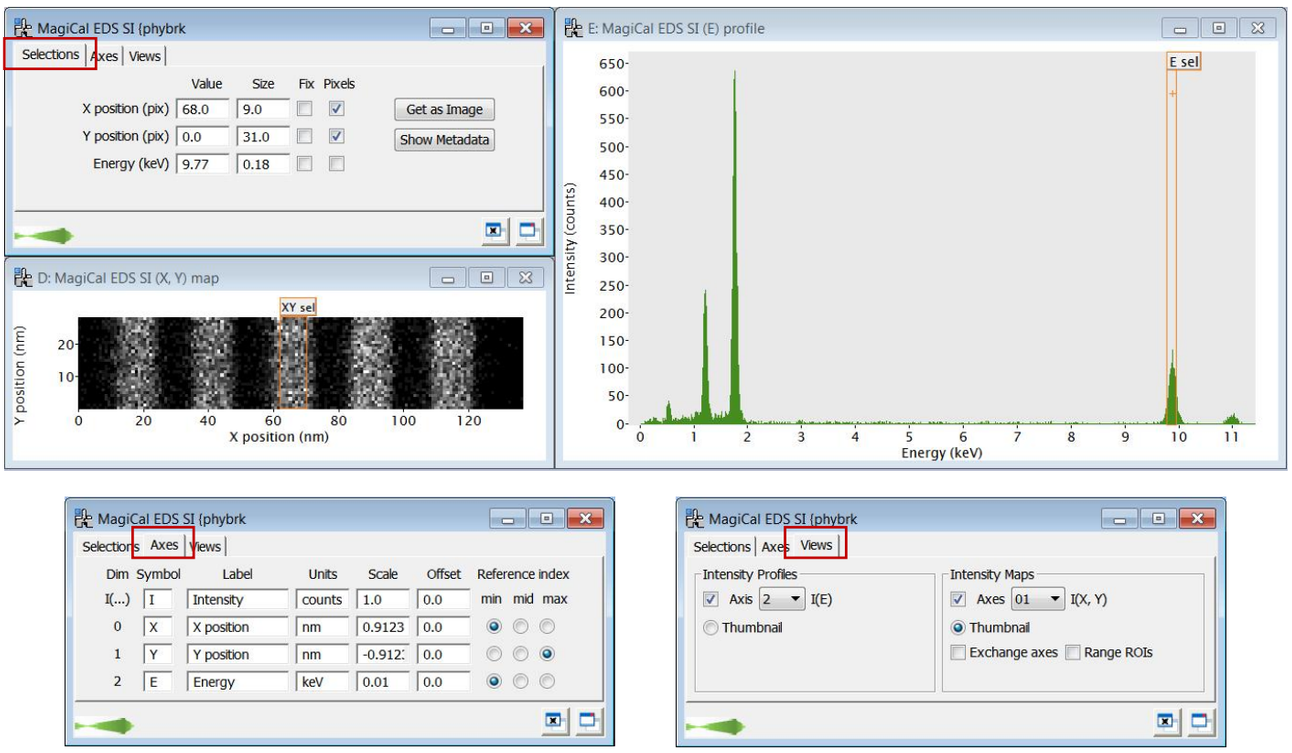
Workshop App: the Data Analysis Design Pattern of the Enabler Framework
Mike Kundmann

A system for viewing and analyzing multi-dimensional, multi-component microscopy data



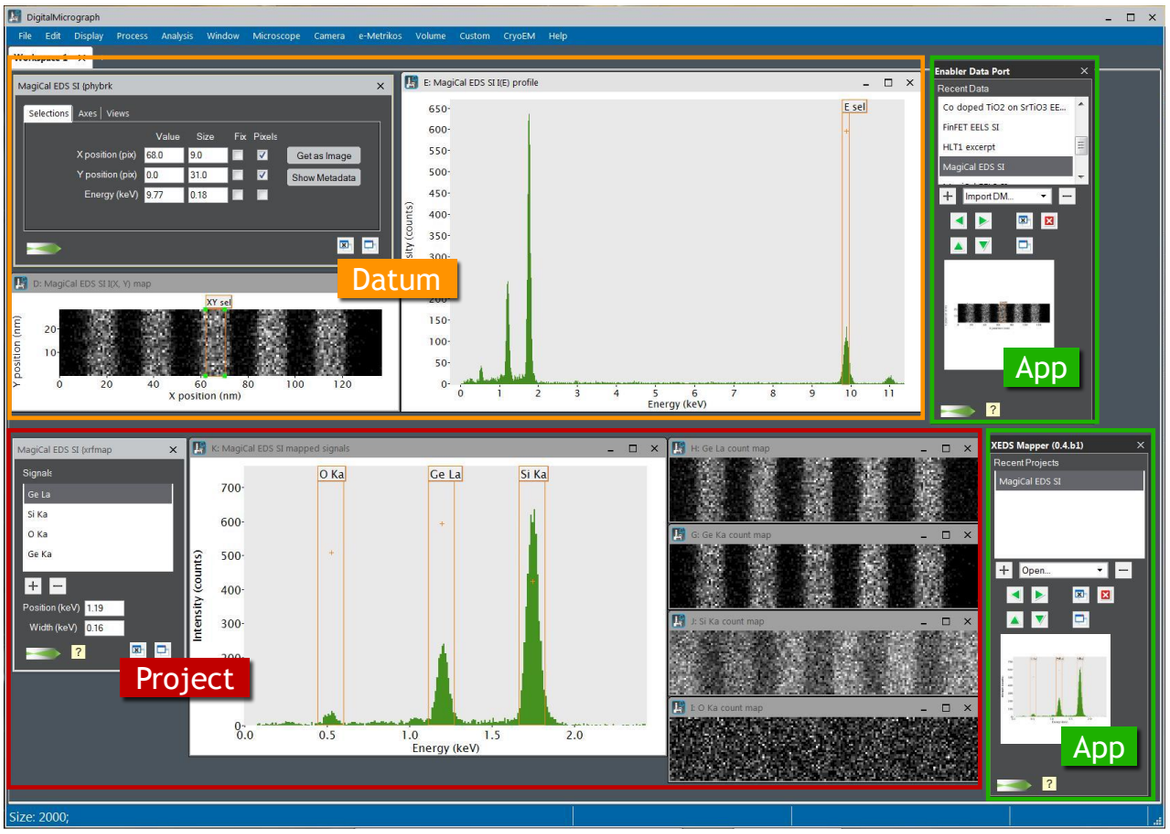
Project motivation and goal

The Enabler framework project seeks to establish a unified context for exploring, integrating, and analyzing microscopy data sets captured with a broad range of techniques and instrumentation [1, 2]. Its goal is a software system in which hyper-dimensional, multi-component microscopy data sets of various types can be viewed and analyzed in a common environment to extract maximum sample information.



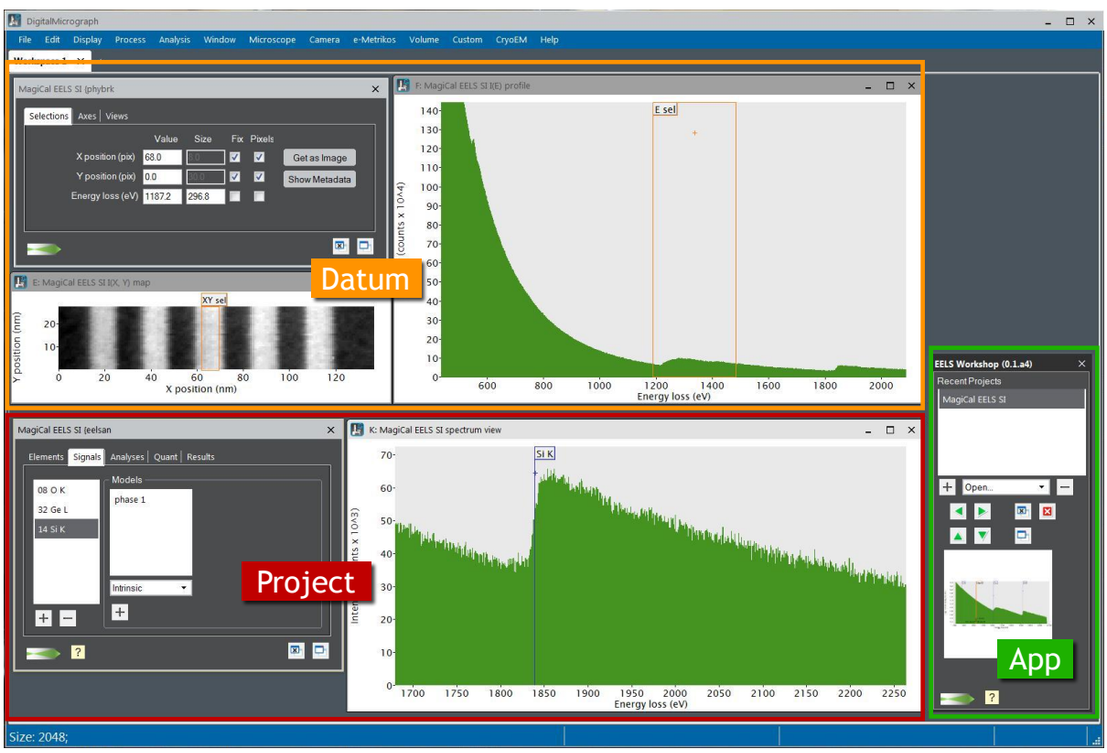
Above is an EDS spectrum image viewed as a typical Enabler data object (type PhysicalBrickDatum). Bottom frames show info accessible via Axes and Views tabs.

Examples of Workshop App in practice: XEDS Mapper and EELS Workshop

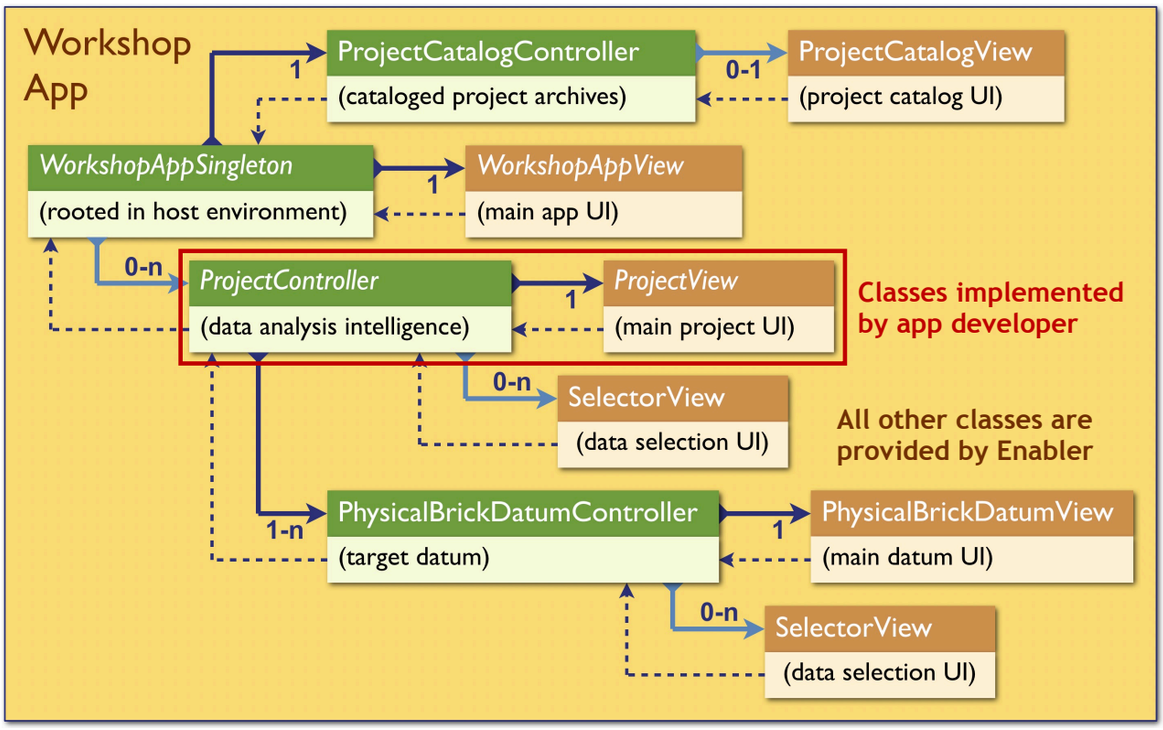


The Enabler workshop concept

Just as a physical workshop brings together materials, tools, and work products, Enabler provides a software context for data, apps, and analysis projects. Each object in the Enabler workshop is a model-view controller (MVC) [3] with its own UI, persistent archive, and references (links) to other objects in the workshop.



Creating a Workshop App data analysis module within the Enabler framework



App architecture for analysis projects

The Workshop App design pattern [3] is depicted above. The app object, at top left, is chiefly a tool for managing project instances of a specific type. The essential core of any workshop app is its ProjectController class. It encapsulates the know-how for extracting key information from the particular type of data set the app is designed to analyze. Each project controller is at the root of a tree or network of further objects linked in a task-specific structure or pattern. Through these linkages, UI or processing events within the project are passed to appropriate target objects, triggering updates of results. Collapsed outline views of the project controller and view classes of the XEDS Mapper app are given below.

```
class e_Metrikos__XEDSMapper_ProjectController : e_Metrikos__Enabler_ProjectControllerBase
{
    String selectedSignalName;
    Object signalNameList;

    //************************************************************************************
    // Required method overrides to manage controller user interface and archiving
    //************************************************************************************
    String GetArchiveObjectTypeExtension(Object self)...
    void SpecifyRequiredAxes(Object self)...
    void ValidateRequiredAxes(Object self)...
    void SetAndArchiveDefaults(Object self)...
    void LoadMembersFromArchive(Object self)...
    Object GetSelectorViewDatum(Object self, String selectorViewID)...
    Object GetSelectorInfoDictionaryForView(Object self, String selectorViewID)...

    //************************************************************************************
    // Public controller interface
    //************************************************************************************
    Number GetSignalCount(Object self)...
    Number GetSignalPosition(Object self, String signalName)...
    Number GetSignalWidth(Object self, String signalName)...
    Number GetSelectedSignalPosition(Object self)...
    Number GetSelectedSignalWidth(Object self)...
    String GetSpectralUnits(Object self)...
    String GetSelectedSignalName(Object self)...
    String GetSignalName(Object self, Number signalIndex)...
    Object GetSignalNameList(Object self)...
    void SetSelectedSignalPosition(Object self, Number newSignalPosition)...
    void SetSelectedSignalWidth(Object self, Number newSignalWidth)...
    void SetSelectedSignalName(Object self, String newSelectedSignalName)...
    void ChangeProjectThumbnail(Object self)...
    void AddSignalToList(Object self)...
    void RemoveSelectedSignalFromList(Object self, Number confirmRemoval)...

    //************************************************************************************
    // Event handlers - override for selector responses and custom behavior
    //************************************************************************************
    void DoRangeParameterChangeEvent(Object self, String parameterName)...
    void ReactToServerFrameParameterChange(Object self, Object serverController, String parameterName)...
    void ReactToDatumIntensityAxisUpdate(Object self)...
    void ReactToDatumIntensityLabelUpdate(Object self)...
    void ReactToDatumDimensionAxisUpdate(Object self, Number dimensionIndex)...
    void ReactToDatumDimensionLabelUpdate(Object self, Number dimensionIndex)...
    void EndSelectorMouseDownTracking(Object self, String trackedSelectorName)...
}
```

```
class e_Metrikos__XEDSMapper_ProjectView : e_Metrikos__Enabler_ControllerRootViewBase
{
    //************************************************************************************
    // Private UI layout creation
    //************************************************************************************
    void PrepareSignalList(Object self, TagGroup signalListSpec)...
    TagGroup CreateSignalInfoSectionLayout(Object self)...
    void AppendContentDescription(Object self, TagGroup contentItemsDescription)...

    //************************************************************************************
    // Private UI event handlers
    //************************************************************************************
    void DoSignalListChangeEvent(Object self, TagGroup listSpec)...
    void DoSignalListActionEvent(Object self)...
    void DoAddSignalButtonEvent(Object self)...
    void DoRemoveSelectedSignalButtonEvent(Object self)...
    void DoSignalPositionChangeEvent(Object self, TagGroup fieldItem)...
    void DoSignalWidthChangeEvent(Object self, TagGroup fieldItem)...

    //************************************************************************************
    // Interface used by parent controller to update UI
    //************************************************************************************
    void UIUpdateSelectedSignalInfo(Object self)...
    void UIUpdateSignalListSelection(Object self)...
    void UIUpdateSignalList(Object self)...
}
```

XEDS Mapper app has less than 700 lines of project code

Flexibility and power of software designs based on object networks

Encapsulation of expertise in objects

A chief strength of object-oriented (OO) design techniques [4] is that they realize domain-specific concepts as functional software modules (classes). In this sense, they turn the abstractions and thought processes of domain experts into working mechanisms that can be instantiated and linked together in novel arrangements to solve new problems. Classes contain both member data and member functions, thus encapsulating both reference data and procedural knowledge in complete modules of expertise, intelligence, and know-how. Enabler harnesses this key aspect of OO design to embody the concepts used in thinking about, working with, and quantifying microscopy data in ready-to-use software classes that can be deployed in a variety of data analysis applications. A further benefit of this approach is that source code and object archive structures are closely linked, making Enabler highly self-documenting and thus more easily portable to other OO contexts. In effect, the documented structure and archives of an object-based design tend to give a software system greater transparency and traceability than is typically available for code using algorithmic function libraries.

Further Enabler classes and features

The Enabler framework has, so far, been implemented using the DM-scripting environment [5], but its concepts and archives can be deployed and accessed with other software platforms. It extends the basic function and class library of DM-scripting by adding the following classes as primitives of the framework: MessageTool, FileAccessTool, ArchiveLocator, XMLElement, List, Dictionary, DateTime, Vector, Range, RectangularFrame, NumericBrick, VectorBrick, and PhysicalQuantity. Further classes for generating spectral model fits and atomic data such as electron shell binding energies and cross-sections are currently being added to support spectral analysis apps like XEDS Mapper and EELS Workshop. Enabler's base classes for app development support both commercial (licensed, app-store distributed) and community-serving (free, developer-distributed) applications. The current release (version 3.5) is available at no cost to all interested parties (send email to enabler@e-metrikos.com). Detailed documentation about its public class libraries and app development tools will be posted on the e-Metrikos website [6].

References

[1] M. Kundmann, Microscopy and Microanalysis, 22 Suppl. 3, (2016) p. 290.
[2] M. Kundmann, 2018 M&M Meeting, late-breaking abstract and poster (un-published).
[3] E. Gamma et al., "Design Patterns", (Addison-Wesley, 1994) p. 4.
[4] G. Booch et al., "Object-Oriented Analysis and Design with Applications", 3rd ed. (Addison-Wesley, 2007) chs. 2 and 3.
[5] http://www.gatan.com/products/tem-analysis/gatan-microscopy-suite-software
[6] http://www.e-metrikos.com/enabler