Paper presented at IEEE ETFA 2017 conference in Limassol, Cyprus, 14 sep, 2017

Concept for AutomationML-based interoperability between multiple independent engineering tools without semantic harmonization

Experiences with AutomationML

Dr. Prerna Bihani
System Projects
ABB HVDC
Ludvika, Sweden
prerna.bihani@se.abb.com

Dr.-Ing. Rainer Drath
Automation and Grid Technologies
ABB Corporate Research Center Ladenburg
Ladenburg, Germany
rainer.drath@de.abb.com

Abstract **This paper describes a concept and its industrial pilot of a methodology for how to automate the data exchange between multiple engineering tools by means of AutomationML without a predefined semantic harmonization or a central database. Key challenges in the methodology and its industrial application are interoperability between 20 participating independent engineering tool platforms, identification of inconsistencies and guiding the project manager to areas of inconsistency. The key focus of this paper is not a commercial product; instead, the authors describe the methodological challenges, derive related requirements and describe the concepts to fulfill the collected requirements.**

Keywords—AutomationML, data exchange, CAEX, heterogeneous tool landscape, interoperability

I. The challenge of data exchange in Engineering

The term “Engineering” in this paper means all activities required to design, test and commission complex and unique process plants as well as discrete manufacturing lines [1]. It entails collaboration among different persons from various technical disciplines [2] with diverse educational backgrounds, performing different tasks such as process engineering, control engineering, cabinet planning, drive applications, safety applications, electrical engineering, PLC programming, robot programming, simulation engineering, IT setup, communication engineering etc. Each of them uses individual engineering software tools, usually from different vendors. This means engineering is teamwork. Engineers can perfectly exchange their ideas verbally, because it is a human ability to communicate. But when it is required to exchange data between their engineering tools the situation becomes tricky. The ability of engineering tools to collaborate with each other across their tool borders, geographical locations or workflow phases, called “interoperability”, is usually weak. Although considered an important indicator of engineering efficiency, it is rarely supported by today’s industrial software [3].

Instead, each engineering tool is locally optimized for its individual purpose but is not designed to interact with another. The handover of engineering data from tool to tool is a tedious, time-consuming and error-prone task, requiring mostly manual labor. Usual means of data exchange are printed diagrams, pdf reports or Excel files at best. In every industrial data exchange scenario, the project progress stalls during data exchange, and in many cases, already engineered data are particularly lost and need to be re-engineered.

In [3], the authors summarized the state of the art regarding engineering tool collaboration. In short, there are mainly two different approaches to achieve interoperability between engineering tools:

- The first approach is the use of a common database. All participating tools are integrated into a common system platform. This leads to a so called “tool suite” with subtools that are well-connected in the database ecosystem. Traceability is reached by linking of data and tools, so that data modifications on any work item are flagged and responsible persons are automatically informed. Tool suites with a common database sound promising, but they have significant drawbacks: a) they bind customers to one vendor, and b) the participating tools cannot innovate independently, meaning all innovations require significant amounts of agreements and harmonization across all connected subtools. Over time, the innovation speed of tool suites is by far slower compared to best-of-class engineering tools which innovate independently.
- The second approach is the use of file-based interfaces. Engineering tools provide predefined export/import interfaces, e.g. plugins based on XML or Excel. File-based data exchange creates a “copy” of the concerned data upon export from the source tool and another “copy” of the same data upon importing into the target tool. The result is consistency across the source and target tools at *one* particular time. The key benefit of file-based data exchange is the loose coupling of engineering tools which allows them to innovate independently and gain efficiency improvements within the tools. A key challenge however is the need for common semantics: importers need to understand the semantics of imported data, hence the exporters

are expected to provide data in a well-defined format including well-defined semantics. This is a crucial point: a standard for the semantics of engineering data has yet to be achieved. Standards only become mature with feedback from their practical application, but practical use is absent since engineering tool providers wait for the final standard: this is called the *standardization deadlock* according to [4]. But even assuming this would be solved: all occurrences of the same data would have their own lifecycles and may be changed in their source and target tools independently, whereby data inconsistency will continuously occur and traceability is missing. A fundamental property and drawback of this approach is that there is no "data ownership". With data sets changing independently, determining which one takes precedence requires constant human intervention. This manual work is error-prone and tedious. Repeated data exchange on file base requires careful change management. If a file-based data exchange is established in an environment with more than two engineering tools, the organizational overhead grows significantly. A network of file-based data exchanges in a project is practically a complex challenge, and the concept is unsuitable for complex data exchange scenarios among multiple engineering tools.

In sum, both approaches are practically unfit for industrial application [5]. A systematic and guided data exchange including difference handling and consistency checks is not available, resulting in expensive, self-manufactured, constantly re-invented, time-consuming, and error-prone data exchange solutions.

The described difficulties motivated ABB Corporate Research to invent a methodology to overcome these issues. This paper describes the methodology and an industrial pilot at HVDC in Sweden which has adopted this technology and introduced it into their workflows promising significant cost savings and quality improvements.

II. THE IDEA OF A FILE-BASED INTERMEDIATE SOFTWARE

The basic principles pursued in this contribution is based on [3], but it simplifies that approach significantly. The present chapter summarizes key innovations of the original methodology.

The main idea to overcome problems inherent in the commonly used approaches is to introduce a software that acts as a middleware providing collaboration *functionality* that cannot be supplied by any data format. The key concepts are *data ownership, Collaboration Objects* and *data exchange feedback loops*.

1. **Data ownership:** Data exchange is strictly directed from a source to a target engineering tool by means of a rigorous *data ownership concept.* The concept distinguishes between the *owner* and the *receivers* of engineering data. The owner defines the shareworthy data and lends those data to the receivers. The granularity of data is on the object level, while an object's exact constitution is determined by people: it may be a parameter, a signal, a physical device, etc. Receivers can borrow the owner's data but are not allowed to modify them. The approach implicitly avoids data conflicts, data branching and synchronization issues. Double ownership of data, though technically permitted, is conceptually disallowed: in rare case of a potential conflict, project management would need to intervene to ensure single ownership.
2. **Collaboration Objects:** Only the necessary and shareworthy engineering data of *data owners* will be packaged and transferred to their *receivers***.** Data owners are responsible for individually defining which data objects should be transferred to their receivers. The file exchange is implemented as file transfer and may occur via shared folders, emails, USB data storages or a cloud. A *shared folder* serves as the common location of data exchange and is defined in advance by a project manager. The *receivers* have knowledge about this shared folder – they can identify the files and import included data into their respective engineering tools, however the data still belongs to the *data owner*. In addition, a "link to the original data" is added for each data object. The source data are *not* transformed into a neutral or standardized semantics: instead, they are only *syntactically* neutralized. The file format is AutomationML which allows storage of source tool specific semantics in a neutral syntax. The mapping and translation into target tool specific semantics is solved in the data model.
3. **Data Exchange Feedback Loop:** In contrast to a normal file exchange, feedback loops are introduced. Both the *data owner* and the *receiver* benefit from utilizing the "link to the original data object" in order to verify consistency of the exchanged data automatically. Additionally, the receiver automatically implements feedback information about which data are imported and actually "used" in the target engineering tool. This is the basis for difference calculation and versioning.

In extension to [3], the concept of semantic harmonization, named *Collaboration Objects*, has been significantly simplified. While Collaboration Objects according to [3] require semantic standardization, the present approach does not. No common data models with standardized semantics are needed anymore. This solves the standardization deadlock described in [4] and allows immediate application without waiting for any standardization. It is the first step of the maturity level concept of semantic standardization defined in [4]. The standardization of data models can be executed gradually and independently from the time pressure of an industrial project.

Thus, the present approach results in functionality which a pure data format cannot provide: systematic and guided data exchange, change calculation, history tracking, versioning, consistency calculation, consistency visualization and a project manager cockpit.

A prerequisite for automatic data exchange with change management is to have stable identifiers for objects (see Openness criteria [7]), which are mandatory for the AutomationML format. The concept is explicitly designed for iterative data exchange between an arbitrary number of independent engineering tool pairs.

Figure 1 shows the key concept: Emily exports data following the source tool semantics, determines shareworthy data for the receivers, and submits relevant data to Lisa. The middleware performs the change management and inconsistency visualization.

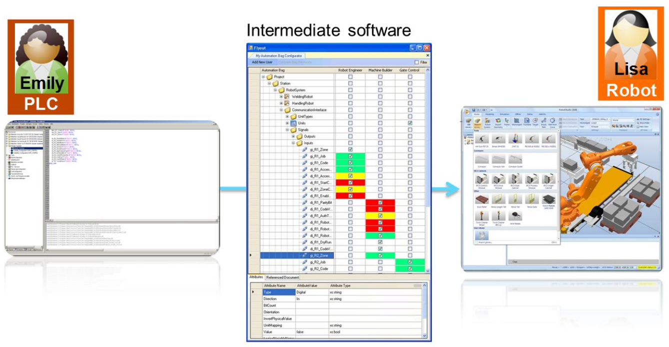


Figure 1: Software prototype illustrating the base concept [3]

Figure 2 illustrates a typical workflow: Emily is the owner of PLC engineering data and sends a subset of data to Lisa, a robot engineer, who acts as a data receiver. The middleware knows which data has been consumed by Lisa, and it can check and visualize at any time whether both data sets are in sync or not. The power of the intermediate software comes into place in the second iteration: Emily performs changes and sends a new version of the data to Lisa. The middleware is able to calculate the differences and visually present consistency status via color codes to both Emily and Lisa.

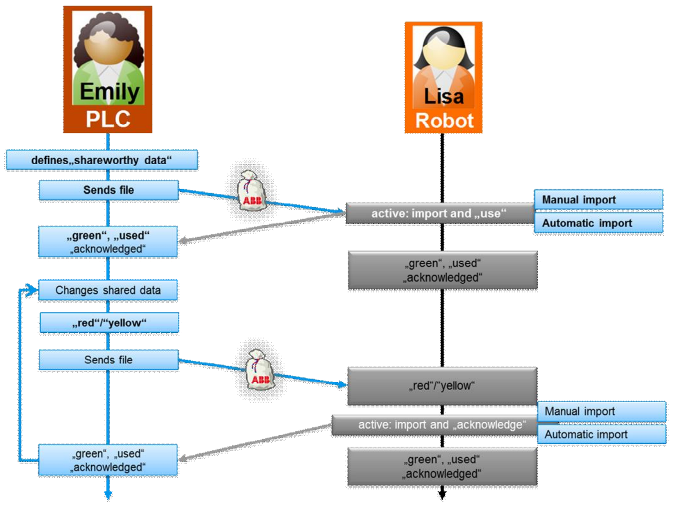

Emily
PLC
Lisa
Robot
defines„shareworthy data“
Sends file
ABB
active: import and „use“
Manual import
Automatic import
„green“, „used“ „acknowledged“
„green“, „used“ „acknowledged“
Changes shared data
„red“/“yellow“
Sends file
ABB
„red“/“yellow“
Manual import
active: import and „acknowledge“
Automatic import
„green“, „used“ „acknowledged“
„green“, „used“ „acknowledged“

Figure 2: Data exchange workflow with 2 participants [3]

Technically, the middleware is characterized by a simple software architecture without databases, client-servers or SOA. It is a simple file-based approach only requiring access to a common file server, SharePoint or other cloud-based storage system. As file format, the middleware uses the AutomationML standard according to IEC62714 [8]. Sending AutomationML files is done by a simple file transfer into a known folder. Even offline data exchange is supported via email or per USB-stick. The software automatically archives all transferred files and provides comparison functionality to observe and visualize all changes over time. It supports data exchange between independent tools that are not designed to interact with each other. Moreover, data ownership conflicts are systematically avoided to achieve data consistency across engineering tools. The middleware provides benefits for different groups:

- For the engineer, it delivers a transparent way to exchange data with other engineers thus offering continuous information about the state of inconsistency between his engineering tool and his receivers' engineering tools. The data exchange is initiated by the engineers themselves – this emphasizes the responsibility of the engineers, allows spontaneous data transfers between arbitrary pairs of engineering tools and consequently utilizes the self-organizing capabilities of an engineering team.
- For the project manager, it conceptually provides all information about the current "state of inconsistency" across the overall project, highlighting focal points for attention.
- For the software developer and hosting organization, it delivers means to minimize their effort in exporter/importer development.

III. Pilot Application at HVDC Sweden

A. Background

ABB HVDC Sweden discovered this research concept while looking for a solution for their data exchange challenges. The HVDC technology was pioneered in 1954 by ABB for reliable and efficient electrical power transmission over long distances with minimum losses. ABB HVDC has since been awarded over 110 projects for a total capacity of more than 120,000 megawatts and accounting for around half the global installed base. The engineering of these HVDC systems requires around 40 engineering tools on 20 engineering tool platforms. Approximately 30 parallel workflows are in operation on an annual basis each with around 400 data exchange interaction points between different engineering studies at the HVDC System Design department alone. The vast majority of these data transfers is today performed via paper format, which is not only error-prone but also requires several man-years of effort each year. Combined with multiple iterations, lack of change management, lead-time delays, and essentially duplicate scenarios across other engineering departments, the total cost for HVDC is much higher.

Several improvement efforts were initiated including a quality improvement project on dataflows to understand the complexity of data transfers during engineering studies. The need for automatic data exchange with change management became increasingly apparent, and a number of desired functionalities were identified for a data exchange software. Partial attempts were made with an internal Excel-based data exchange tool. However, due to that tool's basis in the Excel platform, proper change management functionalities were not implementable and data exchange was restricted to occurring between Excel workbooks. A simultaneous initiative to learn about the software tools of ABB Corporate Research Center Germany (DECRC) put the HVDC group in touch with DECRC scientists and marked the beginning of HVDC's interest in the research prototype.

With the research prototype and its basis in AutomationML, a neutral XML-based data format designed specifically for exchanging engineering data, simple and effective automatic data transfer between heterogeneous engineering tools combined with change management is made possible.

B. *Methodological challenges and industrial requirements*

The requirements identified for a suitable automatic data exchange tool with change management system can be categorized into typical industrial requirements for maximum efficiency, optimal quality, and low maintenance.

<u>Maximum efficiency requirements:</u> Given the heterogeneous tool landscape with multiple users, maximizing both development and operational efficiencies are essential. For fast implementation, a solution allowing efficient development for individual plugins is considered ideal. In particular, it is important to avoid introducing semantic standardization for handling diverse data models and instead allow common semantic models to gradually emerge from increased usage. Furthermore, a technology allowing extensible tool classes is necessary for transferring additional data. For achieving operational efficiency, the entire data exchange process ranging from project setup to data transfer to data report generation and subsequent iterations should be automated for fewest mouse-clicks via an intuitive graphical user interface. This would contribute to shorter lead times throughout the entire engineering workflow. Handling capacity for large amounts of data is needed while enabling transfer and tracking of only the relevant parameters. Additionally, email notifications for any changes in data versions combined with difference calculation functionality are required for effective change management.

<u>Optimal quality requirements:</u> To ensure quality of the data transfers, quality-affirming features are identified at different levels. Firstly, it is considered mandatory to have clear ownership for all transferred data to prevent distribution of conflicting data versions. At the individual data item level, a change tracking mechanism is necessary for communicating consistency status of particular data items to both sender and receivers. An approval option is required at the data version level for confirming quality of the transferred data and digitalizing the data approval process. Finally, at the highest level, an overview feature with current usage and inconsistency status for project-wide data transfers is needed to assist project managers in measuring progress and identifying problem areas.

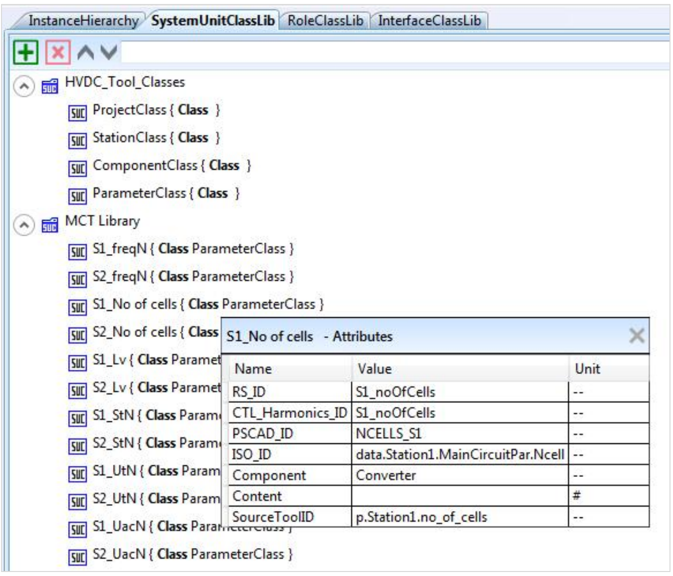
<u>Low maintenance requirements:</u> For smooth and effective administration during project execution, any necessary updates in users and parameters should require minimum effort.

C. *Concepts to solve the collected requirements*

The middleware prototype with its basis in AutomationML already satisfied the essential requirements of compatibility with different engineering tool platforms, support for large data transfers, an intuitive graphical user interface, feedback loops, change tracking, versioning, difference calculations, sender as data owner, and role-based usage. It also supports external document attachments and messaging for group communication. For developers, it enables quick plugin connection for open engineering tools due to minimum effort required for AutomationML programming [6]. As part of its customization for HVDC, the following innovations were made to fulfill the remaining identified requirements (outlined in Figure 5).

1) *Mapping technology: dealing with different semantics*

A key challenge has been to overcome the need for semantic standardization. For this, the data to be exchanged have been analyzed and modelled as SystemUnitClasses as shown in Figure 3.



HVDC_Tool_Classes
- ProjectClass { **Class** }
- StationClass { **Class** }
- ComponentClass { **Class** }
- ParameterClass { **Class** }

MCT Library
- S1_freqN { **Class** ParameterClass }
- S2_freqN { **Class** ParameterClass }
- S1_No of cells { **Class** ParameterClass }
- S2_No of cells { **Class**
- S1_Lv { **Class** Paramet
- S2_Lv { **Class** Paramet
- S1_StN { **Class** Param
- S2_StN { **Class** Param
- S1_UtN { **Class** Param
- S2_UtN { **Class** Param
- S1_UacN { **Class** Para
- S2_UacN { **Class** ParameterClass }

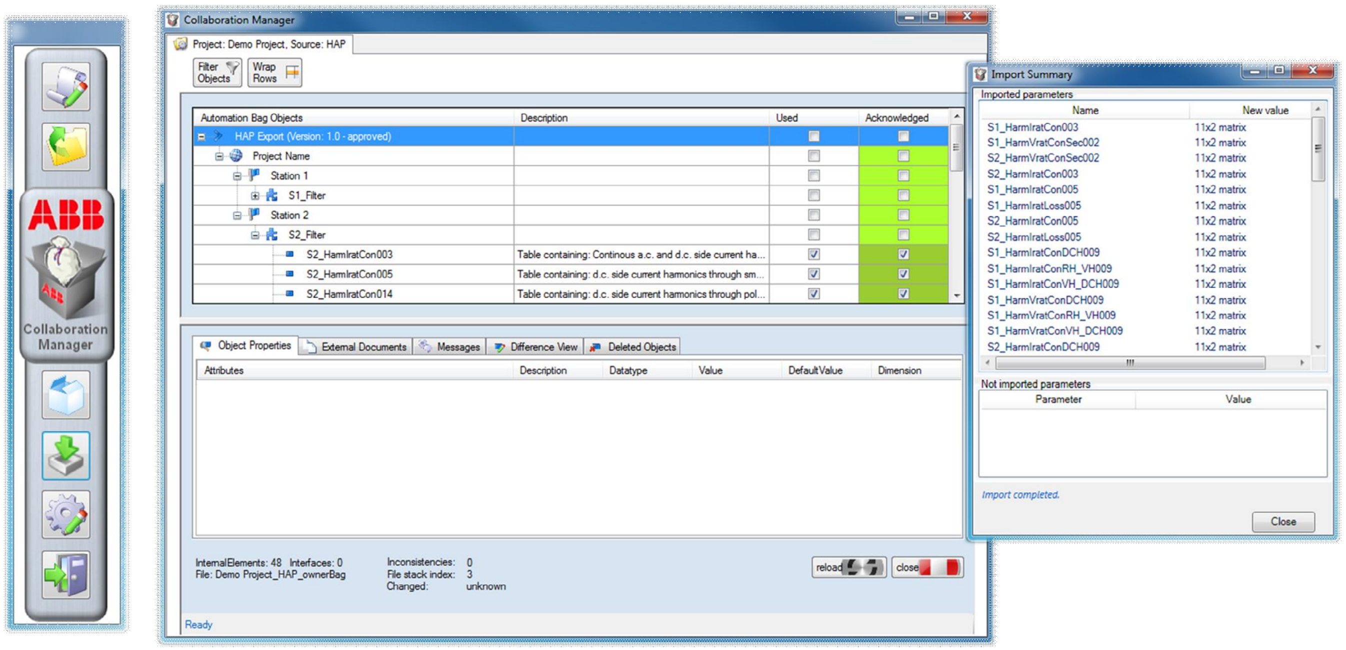
S1_No of cells - Attributes

Name	Value	Unit
RS_ID	S1_noOfCells	--
CTL_Harmonics_ID	S1_noOfCells	--
PSCAD_ID	NCELLS_S1	--
ISO_ID	data.Station1.MainCircuitPar.Ncell	--
Component	Converter	--
Content		#
SourceToolID	p.Station1.no_of_cells	--

Figure 3: Mapping of identifiers as part of the data model

A custom SystemUnitClassLib named “HVDC Tool Classes” has been introduced containing the classes named “Project Class”, “Station Class”, and “ComponentClass” for storing the upper hierarchical levels; and “ParameterClass” for modeling the individual parameters (data objects) transferred between the tools. In particular, the “ParameterClass” is used for identifying each source tool parameter’s identifiers in the target tools which may differ from its identifier in the source tool. With mapping between different identifiers built directly into the data model, data can be exchanged without semantic standardization.

- In step 1, AutomationML SystemUnitClasses have been defined for all types of data, e.g. ParameterClass.
- In step 2, a SystemUnitClassLib is generated for each source tool containing source tool parameters of ParameterClass type, e.g. “MCT Library” with MCT parameter “S1_No of cells”, named according to source tool, of ParameterClass.
- In step 3, each source tool parameter is augmented with multiple parameter identifier attributes for its identifier values in the participating tools, e.g. MCT parameter “S1_No of cells” having source tool identifier “p.Station1.no_of_cells” assigned as “SourceToolID” attribute value is also assigned receiver tool identifier attribute “PSCAD_ID” with value of “NCELLS_S1”, the parameter’s identifier in the PSCAD receiver tool (see Figure 3). This models two types of information in the same place:



Automation Bag Objects	Description	Used	Acknowledged
HAP Export (Version: 1.0 - approved)		☐	☐
Project Name		☐	☐
Station 1		☐	☐
S1_Filter		☐	☐
Station 2		☐	☐
S2_Filter		☐	☐
S2_HarmlratCon003	Table containing: Continous a.c. and d.c. side current ha...	☑	☑
S2_HarmlratCon005	Table containing: d.c. side current harmonics through sm...	☑	☑
S2_HarmlratCon014	Table containing: d.c. side current harmonics through pol...	☑	☑

Attributes	Description	Datatype	Value	DefaultValue	Dimension

InternalElements: 48 Interfaces: 0
File: Demo Project_HAP_ownerBag

Inconsistencies: 0
File stack index: 3
Changed: unknown

Imported parameters

Name	New value
S1_HarmlratCon003	11x2 matrix
S1_HarmVratConSec002	11x2 matrix
S2_HarmVratConSec002	11x2 matrix
S2_HarmlratCon003	11x2 matrix
S1_HarmlratCon005	11x2 matrix
S1_HarmlratLoss005	11x2 matrix
S2_HarmlratCon005	11x2 matrix
S2_HarmlratLoss005	11x2 matrix
S1_HarmlratConDCH009	11x2 matrix
S1_HarmlratConRH_VH009	11x2 matrix
S1_HarmlratConVH_DCH009	11x2 matrix
S1_HarmVratConDCH009	11x2 matrix
S1_HarmVratConRH_VH009	11x2 matrix
S1_HarmVratConVH_DCH009	11x2 matrix
S2_HarmlratConDCH009	11x2 matrix

Not imported parameters

Parameter	Value

Import completed.

Figure 4: Receiver's view in the middleware after 1-click data import into multiple files

a parameter's target tools and its identifier within each target tool. Since objects are expected to have stable IDs, the mapping only needs to be performed once per class and can be reused for multiple projects. Additionally, new parameters can always be added during project execution to the parameter library file. The process is automated via Excel, meaning that the administrator only needs to update the relevant Excel table cells then execute an automatic generation script to create the project-independent parameter library file. The common library file is then immediately readable by all instances of the middleware software for all projects.

This technique has a variety of benefits:

- Exporters only have to instantiate the predefined tool classes and fill in the parameter values. The effort of programming exporters is significantly reduced this way. They don't need any awareness about the target tools; instead, they just need to export the data in source tool specific semantics, since the mapping is hidden in the AutomationML class model.
- Importers read the AutomationML file, navigate through the parameter instances and investigate the sub attributes. If a known identifier is found, the parameter value is imported into the right position of the target tool; otherwise, the parameter is ignored.
- The exporter and importer plugins can be programmed generically. In case of workflow changes due to additional engineering tools, changed identifiers, and new parameters, the AutomationML data model can be automatically generated via a simple Excel file update by an administrator. Since the exporter and importer plugins remain unchanged, no maintenance effort is required.

The mapping technology via AutomationML allows to read and interpret the data during an import on the fly. As a result, plugin development can be started without waiting for semantic standards. The operational benefits are twofold: 1) the relevant data for each receiver is automatically selected for transfer at sender's end; 2) receivers can import data into multiple files with a single-click and without needing to manually perform the mapping. This works without having a common semantic standard, a key property of AutomationML. Instead, common semantic models can naturally emerge with extended use, avoiding the initial standardization deadlock problem (see [4]).

Figure 4 illustrates the middleware after data exchange with a target tool. The main form provides a tree view of the imported data with the right column presenting color-coded consistency status (with green color indicating used values) for individual data exchanged between the source tool and the target tool. The form on the right summarizes the data import success.

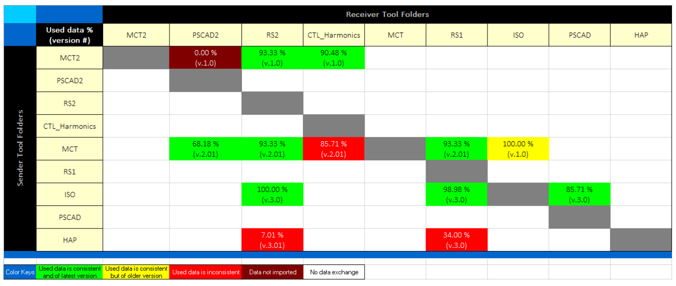
2) Digital Approval: managing quality at version level

In many industrial cases, data sets need to be approved by an expert before they are accepted for import. In this pilot, a digital data approval has been introduced for confirming data version quality by an "Approver" role, a special receiver role introduced with approve/reject functionality to allow digitalization of the data approval process.
Multiple approvers can be assigned to a particular data version, which is subsequently marked "approved" if confirmed by all approvers and is otherwise labelled "rejected". Furthermore, versioning has been linked to approval status for easy identification of data quality.

3) *Project Cockpit: managing quality at project level*

So far, we have considered the state of inconsistency on individual data object level visible to data owners and their receivers in the middleware. One of the key challenges in data exchange across multiple engineering tools lies in obtaining a project-level overview of the inconsistency status across data versions in participating engineering tools for guiding the project manager to problematic areas. A project management cockpit has been invented to satisfy this requirement.

The Project Cockpit is designed to present version history and current status snapshot for data transfers across an entire project – a matrix showing color-coded status for latest data transfers between all sender and receiver tools in the project (see illustration in Figure 6). The idea is to present the state of consistency and usage for every data exchange point in the sender-receiver matrix, where each individual cell signals a receiver's usage and synchronization status for the corres-ponding sender's data. White fields indicate no data exchange. Green fields denote full consistency with respect to latest data version exchange for the tool pair. Red fields report presence of inconsistencies between a tool pair, while yellow fields indicate consistency with respect to an older data version. Gray fields lie on the diagonal and are therefore null. The percentages in



		Receiver Tool Folders								
	Used data % (version #)	MCT2	PSCAD2	RS2	CTL_Harmonics	MCT	RS1	ISO	PSCAD	HAP
Sender Tool Folders	MCT2	▨	0.00 % (v.1.0)	🟩 93.33 % (v.1.0)	🟩 90.48 % (v.1.0)					
	PSCAD2		▨							
	RS2			▨						
	CTL_Harmonics				▨					
	MCT		🟩 68.18 % (v.2.01)	🟩 93.33 % (v.2.01)	🟥 85.71 % (v.2.01)	▨	🟩 93.33 % (v.2.01)	🟨 100.00 % (v.1.0)		
	RS1						▨			
	ISO			🟩 100.00 % (v.3.0)			🟩 98.98 % (v.3.0)	▨	🟩 85.71 % (v.3.0)	
	PSCAD								▨	
	HAP			🟥 7.01 % (v.3.01)			🟥 34.00 % (v.3.0)			▨

Color Keys	🟩 Used data is consistent and of latest version	🟨 Used data is consistent but of older version	🟥 Used data is inconsistent	Data not imported	No data exchange

Figure 6: The Project Cockpit, Example of status overview for 9 Project Engineers with individual tool folders, 6 tools

each field indicate usage with respect to shared data amount.

Since each user is assigned a unique tool folder, and all tool folders appear in the project cockpit, the consistency matrix implicitly shows related users, helping the project manager to get in contact with them.

The matrix is automatically generated from a collection of AutomationML statistics classes designed to store and update status information during user operations. A statistics class is a predefined AML SystemUnitClass designed to gather a data owner's version information during user operations in a given project. It is generated by data exporters and updated by data

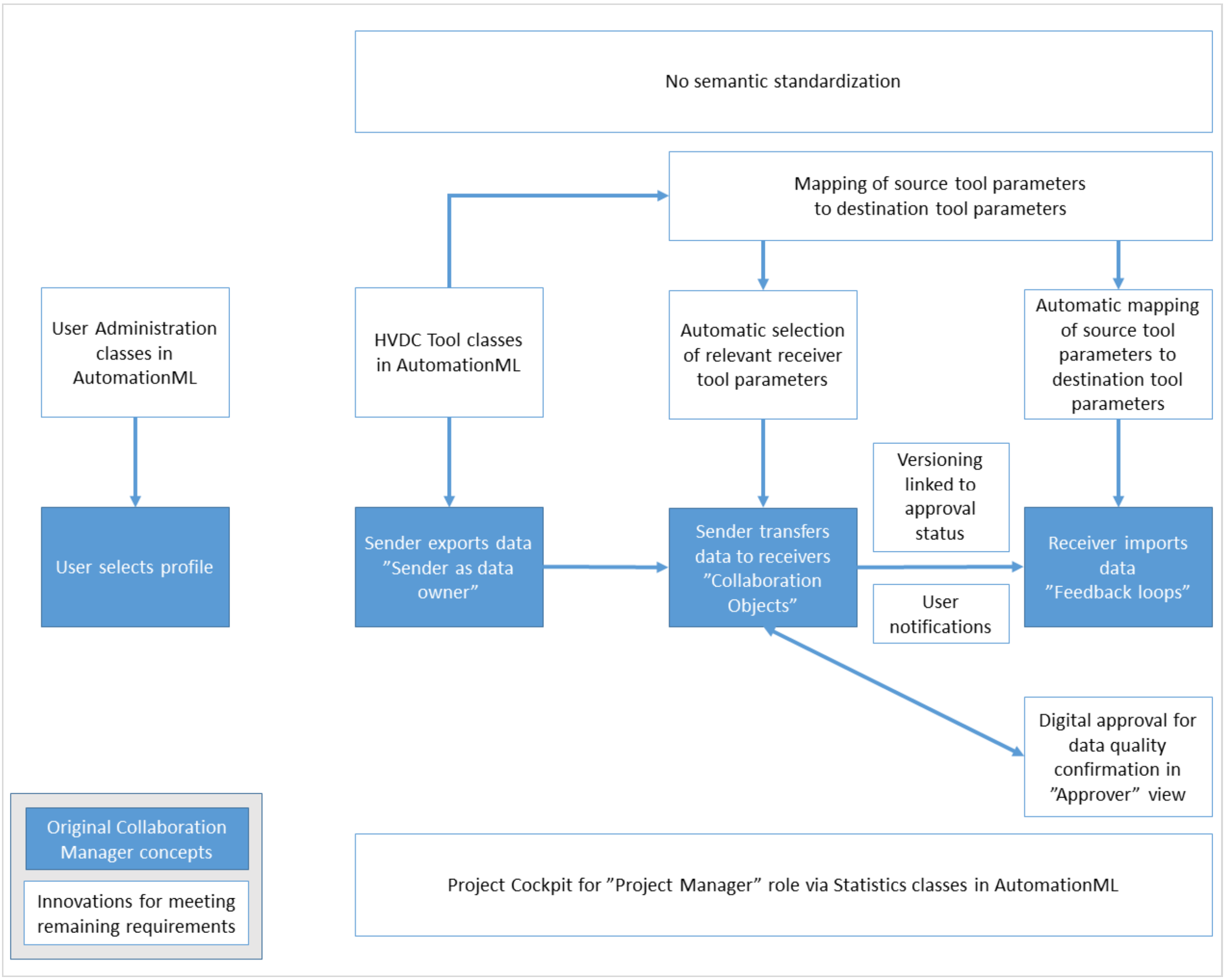

No semantic standardization
Mapping of source tool parameters to destination tool parameters
User Administration classes in AutomationML
HVDC Tool classes in AutomationML
Automatic selection of relevant receiver tool parameters
Automatic mapping of source tool parameters to destination tool parameters
Versioning linked to approval status
User selects profile
Sender exports data "Sender as data owner"
Sender transfers data to receivers "Collaboration Objects"
Receiver imports data "Feedback loops"
User notifications
Digital approval for data quality confirmation in "Approver" view
Original Collaboration Manager concepts
Innovations for meeting remaining requirements
Project Cockpit for "Project Manager" role via Statistics classes in AutomationML

Figure 5: Overview of original middleware concepts and innovations for meeting remaining requirements

importers. The gathered history for all data owners across the entire project is then used for project information and consistency matrix in the project cockpit.

Current statistics information is relayed to the Project Cockpit with a single click by Project Managers who can now view status updates and easily identify potential problem areas for resolution. Data owners can also access the cockpit for an overview of their data versions and consistency status for their respective receivers.

The statistics classes could be further used to measure significant trends over time while the project cockpit could be extended to provide a current status overview across all projects for higher management view.

4) *User Administration via AutomationML*

The overall setup of engineering tools, user roles and users is managed via an AutomationML file. This has been achieved by modelling multiple role classes in RoleClassLib for the project engineer role, the project manager role and the approver role; and system unit classes for users, projects, and tools in SystemUnitClassLib. The classes are instantiated for each user together with their projects, roles, and tools information in an InstanceHierarchy. Roles are assigned by the administrator based on project requirements in the AML administration file, which is read by the middleware for user profiles. Access to available profiles and data versions is handled by the middleware based on native operating system security and designated roles. The resulting AutomationML file provides complete setup for all users of the middleware.

In order to avoid unauthorized modifications of this administration model, the AutomationML file has been stored in a folder that is readable by all middleware installations but modifiable only by the administrator. This means that all security is solved by means of the operating system.

All middleware installations read the administration file and provide features according to the role of the user. Due to the admin file, no user can select a wrong role or project.

The key benefit of modelling the administration file via AutomationML is that user classes defined in AutomationML allow for user profiles to be set up for project execution in a matter of a few minutes, using just the AutomationML Editor or any XML editor. No additional tools are required. As an extension of this approach, the generation of these classes has been automated, thereby requiring the administrator to only update a simple Excel file.

5) *User Notifications*

Whenever a user submits new data to a receiver or approver, the middleware notifies the relevant persons within the middleware. In addition to these internal notifications, automatic email notifications are generated in Outlook for avoiding unnecessary lag. Data owners are also notified about approval decisions by email and within the middleware.

D. *Pilot description*

The plugin architecture of the middleware software has hitherto been used to develop plugins for six core HVDC engineering tools across multiple platforms: Main Circuit Toolbox, Harmonic Analysis Program, insulation coordination program ISO Light, harmonic voltage sources calculation tool CTL Harmonics, transient studies simulation software PSCAD, and Requirement Specifications in Excel. These six tools exchange thousands of parameter values and comprise a major portion of data transfers at HVDC's System Design department. Data can now be exchanged automatically between senders and receivers with a few simple mouse-clicks and without waiting for paper reports to be issued. Further extension to other departments of HVDC is planned.

IV. SUMMARY

Using AutomationML as a neutral file format for the exchange of engineering information between two engineering tools is a known an accepted method. But using AutomationML for synchronization of engineering data between 20 tool platforms for multiple users and projects in parallel, with no semantic standard under industrial requirements for cost efficiency and time pressure is new. The application answers challenges that naturally arise in a setup having more than just two tools: e.g. multiple semantic models across all participating engineering tool platforms, multiple users and user roles, role authorization, complex change tracking, multiple paths across the workflow, adaptations of the workflow, quality approval, and the complexity of inconsistency and its visualization.

The presented middleware concept consequently applies the digitalization of workflows, providing seamless electronic data exchange between independent engineering tools based on AutomationML, change tracking, data responsibility, a new digital data approval process, versioning of data sets and messaging. It combines the benefits of using best-of-class tools from different vendors with data consistency of a tool suite.

Traceability is implemented between data owners and their immediate receivers. Further linking of dependencies downstream would require engineering tools to provide dependency information among input and output parameters (based on internal algorithms) to the middleware, which is technically feasible.

Scientifically of interest, this is the first industrial application of AutomationML in data exchange without semantic standards for the maturity-level concept described in [4]. The data exchange between all tools happens without any semantic standardization. Consequently, development of data exporters and importers does not require prior standardization across all participating tools nor do the plugins require knowledge from each other. Instead, data mappings are externalized from the exporter/importer software and built directly into the AutomationML information model. Hence, all data models across the tool platforms remain proprietary and no semantic standards must be developed. This eliminates the standardization deadlock and instead allows gradual standardization to occur over time [4].

Changes in the workflow and additional data items don't need any modifications of the importer or exporter plugins, only the AutomationML information model needs to be updated. This leads to reduced development and maintenance efforts as well as high flexibility in the workflow.

Furthermore, this contribution has for the first time widened the industrial application scope of AutomationML. On top of modelling and transferring *engineering* data, AutomationML has been successfully applied for modeling *statistics* data, *mapping* information and project *administration* data for user authorization. This demonstrates the applicability of AutomationML for all engineering workflow data.

A key innovation is the introduction of a project cockpit which synthesizes the statistics generated for all tool pairs into a single comprehensive overview about project-wide data transfers with a color-coded snapshot to guide the project manager to the centers of inconsistencies. The industrial application of the presented methods results in shorter lead times, earlier cross-checks, and data reports that are ready earlier for submission to customers. In numbers, automatic data exchange results in savings of roughly 95% of time currently spent in paper-based data exchange. Relevant adaptations of the dataflow processes leads to related improvements. Elimination of manual, tedious, paper-dependent workflows results in improved focus on engineering. The application promises significant cost savings and quality improvements for HVDC workflows.

Beneficial in this first industrial application of an AutomationML-based data exchange at HVDC Sweden are the relatively low complexity of the data model and the software support from the AutomationML e.V. with its free AutomationML Engine [9].

The software has entered its first industrial pilot application in 2017.

References

[1] Fay A., Efficient engineering of complex automation systems, In: Schnieder (editor): Will traffic automatically be safer?, Braunschweig, 2009, pp. 43-60 (in German language).

[2] Biffl S., Sunindyo W.D., Moser T.: Bridging Semantic Gaps between Stakeholders in the Production Automation Domain with Ontology Areas. In: Proc. “21st Int. Conf. on SW Engineering and Knowledge Engineering (SEKE 2009)”, Boston, USA, 2009, pp. 233-239.

[3] Drath R., Barth M..: Concept for interoperability between independent engineering tools of heterogeneous disciplines. In: proceedings of the ETFA 2011, Toulouse, 2011.2012

[4] Drath R., Barth M.: Concept for managing multiple semantics with AutomationML - maturity level concept of semantic standardization. In: Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012), IEEE (2012), Krakow, 2012.

[5] Tauchnitz, T.: Schnittstellen für das integrierte Engineering. atp – Automatisierungstechnische Praxis 56 (2014) H. 1-2, S. 30-34.

[6] Drath R.: Let’s talk AutomationML - What is the effort of AutomationML programming? In Proceedings of the “IEEE Conference on Emerging Technologies and Factory Automation (ETFA), September 2012.

[7] Barth M., Drath R., Fay A., Zimmer F., Eckert K.: Evaluation of the openness of automation tools for interoperability in engineering tool chains. In Proceedings of the “IEEE Conference on Emerging Technologies and Factory Automation (ETFA), September 2012.

[8] IEC 62714: Engineering data exchange format for use in industrial automation systems engineering (AutomationML). September 2012.

[9] www.automationml.org