IADIS International Journal on Computer Science and Information Systems
Vol. 10, No. 1, pp. 47-63
ISSN: 1646-3692

INTELLIGENT AND SELF-ADAPTING INTEGRATION BETWEEN MACHINES AND INFORMATION SYSTEMS

Heiko Kern, Fred Stefan. *University of Leipzig, Business Information Systems, Leipzig, Germany.*

Vladimir Dimitrieski. *University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia.*

ABSTRACT

Increasing automation in the manufacturing industry requires a comprehensive integration of machines and business information systems. Driven by the Internet of Things or the high-tech strategy Industry 4.0, an efficient integration plays in this domain an increasing role. Despite powerful technologies, the integration is a challenging and labor-intensive task. To walk with the development, machines and information systems need flexible and powerful integration mechanisms with self-configuring and self-adapting features. The ideal conception would be a plug & produce mechanism, which follows the USB plug & play principle. In this paper, we address this problem and present a novel approach for a structured, automated and reusable integration of information systems and machines. The approach is realized as a framework which allows the development of transformations between different data schemas. Accordingly, the framework is positioned between machine and application layer. The framework consists of a declarative mapping language with a graphical notation and an intelligent solution for connecting different systems. In this contribution, we give an overview of the framework components and demonstrate the approach in a practical use case.

KEYWORDS

Integration, Mapping, Transformation, Automation

1. INTRODUCTION

Increasing automation in the manufacturing industry requires a comprehensive integration of machines and business application systems. According to emerged visions of Internet of Things (Mukhopadhyay 2014) or Industry 4.0 (Dujin et al. 2014), continuous integration is an essential requirement for the implementation of common business processes. The realization of this guiding principle requires a horizontal and vertical integrated information flow throughout the entire automation pyramid. Machines on the lowest level have to be vendor-independent, flexible and efficiently integrable with application systems from the IT-level and new cloud services.

With increasing automation and coupling degree the factors adaptability, quality and efficiency of the machine integration play a central role. Currently, the exchange of data within the automation pyramid does not meet future requirements in terms of flexibility and adaptability. As shown schematically in Figure 1, there is a horizontal gap between the machines on factory level and the overlying applications and services on enterprise level. Additionally, there exists a vertical gap between machines from different manufacturers, customers and domains.

Manufacturers of application systems are facing with the challenge to integrate their products into the existing machine landscapes of their customers. Often, the machine and equipment landscape is heterogeneous and characterized by many different interfaces. Despite a variety of standardized industry protocols or exchange standards, machine interfaces are often adapted for a certain domain, manufacturer, or machine. Thus, integration between machines and overlying application systems causes manual adaptation effort which is complex, time-consuming, and expensive. Moreover, quality and transparency of the integration solution are hindered.

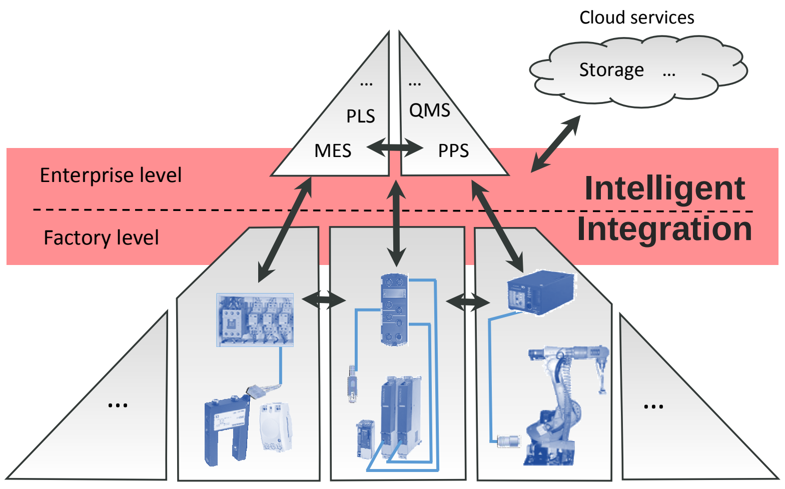


Figure 1. Horizontal and vertical lack of integration inside the automation pyramide

In this paper, we address the described integration problem and present a novel approach for a structured, automated, and reusable integration of information systems and machines. The central idea of our approach is a machine and information system independent coupling component, which allows a systematic reuse of integration knowledge from previous integration projects. The reuse or adaptation of existing integration knowledge to new projects is to be provided by a framework in an automated and transparent way. This simplifies the integration of new machines and improves the response time of production process changes.

The paper is structured as follows. In Section 2, we identify specific circumstances and challenges in the field of machine integration. In Section 3, we present our solution in detail. Afterwards, we present the implementation of the framework in Section 4. After that, we illustrate a use case in Section 5 and evaluate our approach in Section 6. In Section 7, we discuss related work and conclude this paper in Section 8 with a summary and suggestions for future work.

2. INTEGRATION DOMAIN

Generally, integration in the area of software and system development can be defined as the process of linking separate computing systems into a whole so that these elements can work together effectively (Linthicum 1999). System integration is a manifold discipline with a lot of different aspects. In this paper, we focus on the integration between machines and information systems. Furthermore, we are interested in the integration on data or function level (Ruh et al. 2001), more precisely, we investigate the data exchange between different data structures provided by systems.

A further integration aspect concerning our approach is the unification mechanism to overcome heterogeneity between systems. We can distinguish between the following two mechanisms: standardization and transformation. Standardization can be defined as a development process of a standard which avoids heterogeneity a priori by defining a common structure. For the integration between machines and application systems there are a variety of standards which overcome the technical heterogeneity (e.g. OPC, SECS/GEM, WSDL, ODBC, Ethernet, and Fieldbus) and standards for semantic/functional heterogeneity (e.g. MAP/MMS and B2MML) (SISCO 1995 and Scholten 2007). However, in practice, such standards are frequently adapted to a specific domain, manufacturer or machine. Despite these standards, many machines and systems offer proprietary formats or adapted standards. Thus, a mapping or transformation approach is necessary to overcome the heterogeneity between different structures. The aforementioned unification mechanisms are not mutually exclusive. A proprietary structure can be mapped to a standard by using a transformation. Our approach focuses on the semantic/functional heterogeneity and uses the transformation approach as a unification mechanism.

Application vendors are often focused on a certain industry domain and they must integrate their system with similar systems used by their customers in this domain. For example, a vendor of a Manufacturing Execution System (MES) or a Quality Management System (QMS) is specialized in quality management and must usually integrate data from measuring and testing machines. We assume that the semantic entities of these systems are similar and differ only in some issues, such as, naming, different attributes and relations, or different serialization formats. Therefore, the transformations between these entities in integration projects are similar and differ only in a defined variability. Despite powerful approaches, developers are often facing the challenge that they cannot apply their existing transformation knowledge to new integration projects.

Currently, in practice the integration is done as shown in Figure 2. In the most cases, the machines and application systems were integrated by individual, hard-coded machine connectors. Often, the interfaces comprising the transformation code are created manually. This means, changing the interface according to new needs (e.g. production change or machine updates), causes a lot of manual adaption effort. Additionally, the current approaches are characterized by an insufficient reuse. Oftentimes, existing transformations are simply copied and manually adapted to similar projects. This method is error-prone and contributes to a lack of transparency.

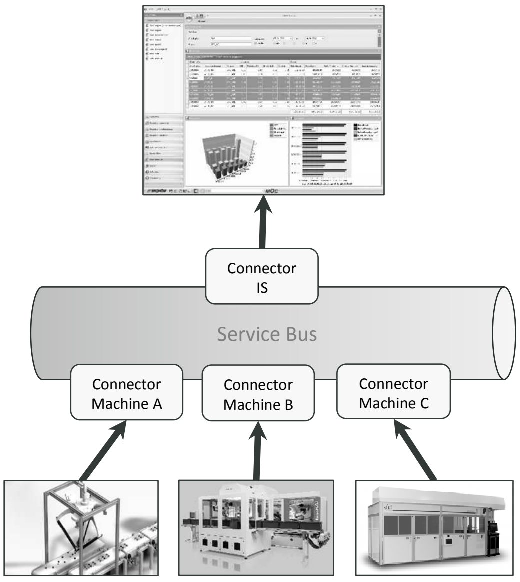


Figure 2. Current state of the art: manual developed machines connectors

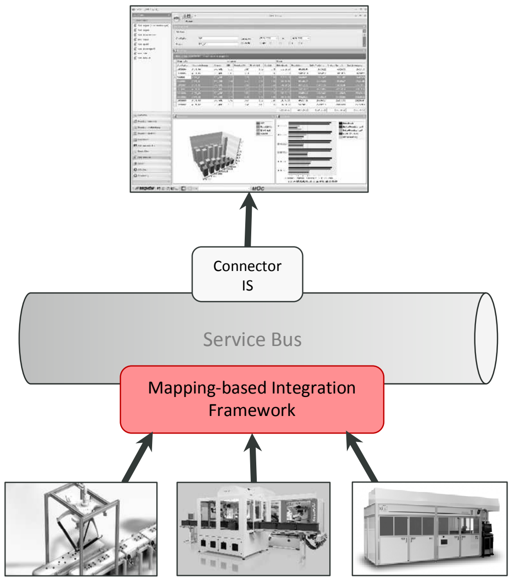


Figure 3. Planned objective: machine connection using the integration framework

To solve this integration challenge, our framework should allow the reuse of existing transformation knowledge in new and similar integration projects. As shown in Figure 3, our mapping-based integration framework should substitute the individual machine connectors. For this purpose, the framework should be non-invasively applicable as a standalone component on-the-top of existing machines.

Our solution should differentiate between the transformation logic itself and implementation of this transformation logic. This separation enables the portability of transformation knowledge to different scenarios and integration platforms. Additionally, the executable transformation code should be generated automatically. This automation minimizes the development effort and increases the quality of the solution. Beside the automatic creation of transformation code, the framework should suggest possible transformation logic in order to increase the automation of the entire development process. In the next section, we present our mapping-based integration framework.

3. MAPPING-BASED INTEGRATION FRAMEWORK

3.1 Framework Overview

The integration framework can be divided into several components. In Figure 4 we give an overview of the framework architecture. The first component is the importer of data schemas from the source and target systems. The second component is a binder which creates an abstract element tree to represent concepts (types, attributes and relations) of a schema. The element tree abstracts from concrete schema implementation details and allows the representation of schemas expressed in different technologies (e.g. XML schema, database schema, CSV, or JSON). This allows the access to the actual machine data on the source side and to the application systems on the target side. The data can be compliant to different protocols.

Based on the element trees, a mapping can be defined in order to overcome the heterogeneity of the source and target schemas. This functionality is provided by the mapper editor. A single mapping comprises a set of the mapping rules that are independent of a concrete schema technology because they are defined between element tree concepts.

Created mappings are stored in a repository. This repository can be regarded as a knowledge-base for mappings and allows their reuse in similar projects. The framework includes algorithms implementing comparison strategies to find repository mappings fitting to the objects of the element tree. Finding correct mappings can be seen as an intelligence feature that allows the automatic creation of new mappings.

Mappings and their rules are abstract correspondences between data schemas. To get an executable transformation, a generator iterates over the specified mapping rules and produces an executable data transformation. The generator is specific to a source and target schema technology and a selected environment for the transformation execution. The final step is the deployment and execution of the generated transformation that finally realizes the data exchange between source and target system.

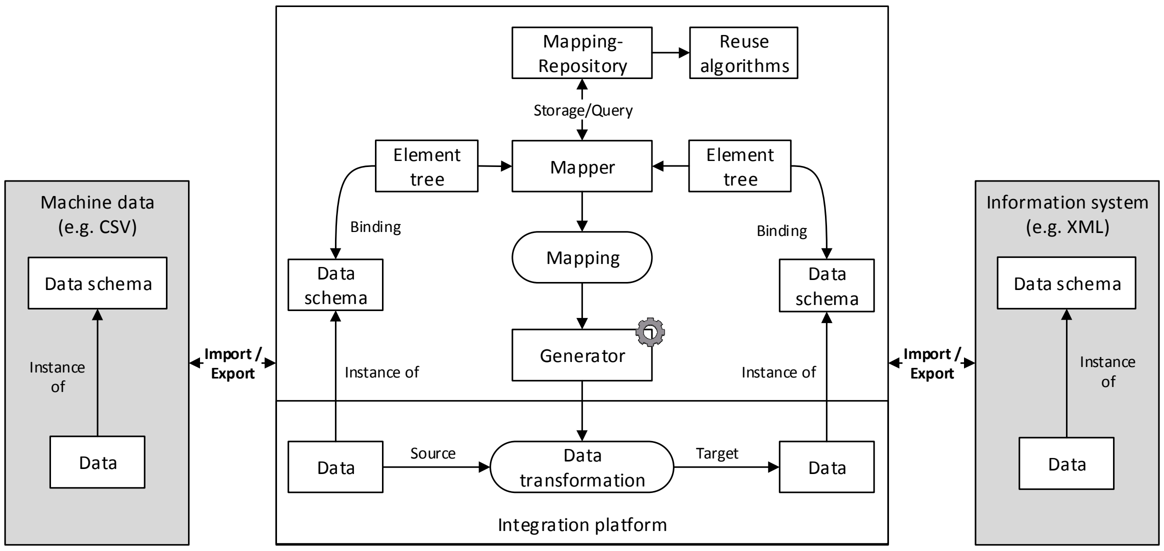


Figure 4. Overview of the mapping-based integration framework

Overall, the application of the integration framework consists of four main steps. These are presented in Figure 5. In the first step, the source and target structures are linked. This generates the abstract element trees, which forms the basis of the mapping. The second step is the creation of a mapping. We distinguish between a learning phase and application phase. During the learning phase, a user creates the mappings manually. Each mapping of the learning phase is stored in the repository. The more extensive the learning phase, the greater the knowledge of the repository. In the application phase, the acquired transformation knowledge can be automatically applied to the new source and target structures. Therefore, reuse algorithms compare elements from the current element trees with existing rules from the repository in order to find appropriate mapping candidates. Depending on the quality of the proposed mapping suggestions, they can be automatically adopted to the new structure. If the suggested solutions make no sense, mapping suggestions can also be modified by hand. The derived mapping can be also stored as transformation knowledge in the repository. After the mapping creation, a generator generates the corresponding transformation code for a runtime environment. In the last step, the generated code is automatically deployed and executed by an integration platform.

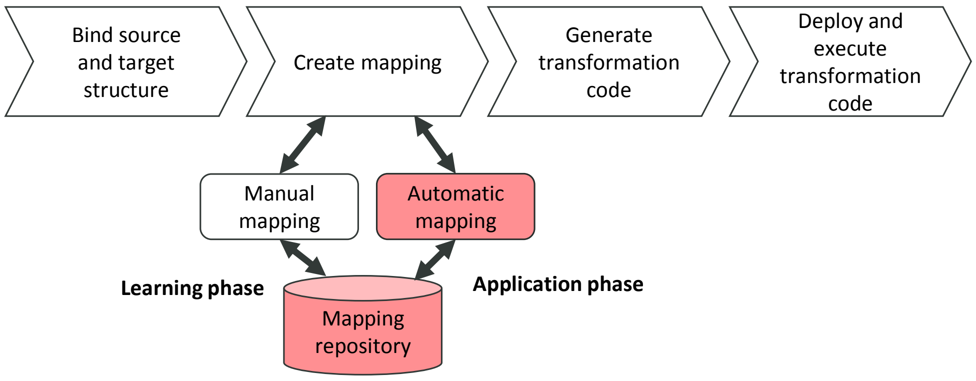

Bind source and target structure
Create mapping
Generate transformation code
Deploy and execute transformation code
Manual mapping
Automatic mapping
Learning phase
Application phase
Mapping repository

Figure 5. Necessary steps for the application of the integration framework

3.2 Representation of Data Schemas

The mapping approach aims to provide an abstract mechanism for specifying mappings regardless the underlying data schema technology. For this reason, we provide a generic tree representation of schemas. A binding component reads a data schema and creates an element tree structure of this schema. A user interface shows this element tree in a specific tree view. Based on this view, a mapping designer can specify a mapping. Each concept in an element tree holds a reference to the original concept of a data schema. This reference is important in order to get specific details of the data schema for the later generation of the transformation code.

An element tree consists of an element container and a set of elements. The element tree structure is part of our mapping language. An element container (*ElementContainer*) represents a data schema. Each element container has a name corresponding to the name of an imported data schema and a location of the schema. Furthermore, an element container has a binding type and a binding configuration. The binding type determines the responsible binding component, the interpretation of the binding configuration, and is important for the selection of the generator. Each element container comprises zero or more elements (*Element*). Each element has a binding string and a name. The name attribute corresponds to the name of an original element from an imported schema. The binding string stores the path to the native element in the original schema. The format of the binding string depends on the schema technology. The binding string or reference is necessary during the generation process of the executable transformation. Additionally, the mapper tool enables the presentation of different schemas in an element tree. Currently, we support the following binder components: CSV, XML and SECS/GEM. Figure 6 shows example of different schemas represented as element trees.

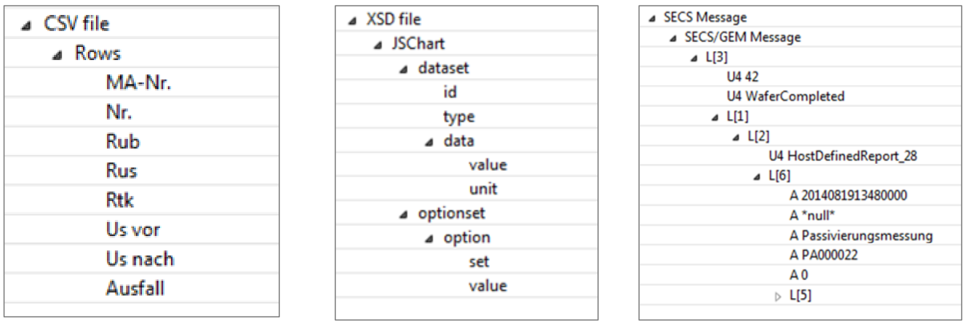


Figure 6. CSV, XSD and SECS binder

3.3 Mapping Description

Based on the created element trees, a mapping can be defined by using a mapping language. The abstract syntax of this language is presented in Figure 7. The root element of a mapping description is a mapping container (*MappingContainer*). A mapping container comprises zero or more links (*Link*) and nodes (*Node*). Each mapping container references at least one source and one target element container (*ElementContainer*). A mapping rule is represented by an operator (*Operator*) linked to source and target elements. In dependency of the source and target links in a mapping rule, we classify operators as one-to-one (*OneToOne*), one-to-many (*OneToMany*), many-to-one (*ManyToOne*), many-to-many (*ManyToMany*), and zero-to-any (*ZeroToAny*). The zero-to-any operator requires only a target element. This operator provides the creation of any number of target elements. Operators have a name for their unique identification in a mapping container, and an operator expression for specifying a filter on source elements. Each operator is connected to source and target elements using links (*Link*).

Operators can depend on other operators. The dependency of operators results from the parent-child relation of elements from the source or target structure. If all elements that are being mapped have no parent elements in the operator tree, the mapping is considered to be a root operator. If one of the participating elements has a parent element in the tree, the operator is considered to be a child operator.

The source and target links of an operator reference only a certain set of data which is defined by a type element in a schema. The mapping logic between these links is descripted as assignment statements inside an operator. Each source and target link of an operator has an identifier or variable name. This identifier can be used in an assignment statement to define the concrete mapping between input and output of an operator. Additionally to the assignment statements, the expression of select statements is possible. This is necessary to select a subset of data from a schema element. Additional to this assignment and select statement, an operator allows the definition of an execution constraint. This constraint is a condition which controls the execution of an operator.

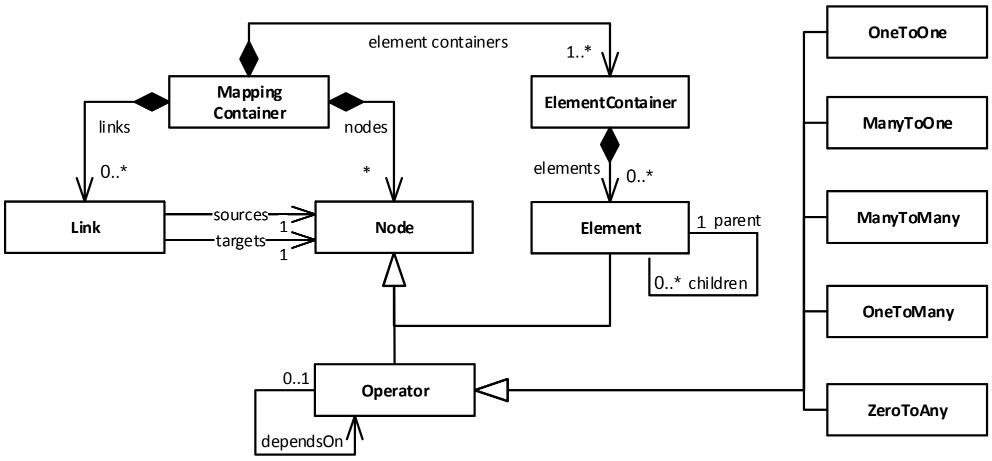


Figure 7. Structure of mappings and element trees

3.4 Transformation Execution

A mapping describes only the relationship between elements of data schemas involved in the process. This description must be executed in order to transform the data from the source into the target system. We use a generator approach for the creation of executable transformations. The generator iterates over the mapping rules and creates transformation code. The generator can query specific information of the source and target schema which is referenced via the elements in the element tree. There is a generator for each combination of (i) execution environment, (ii) source and (iii) target schema technology. The transformation can be executed by an existing environment (e.g. transformation systems, programming languages, or integration platforms) and can be realized as a module or plugin for an integration platform (MuleESB or OpenESB) or an independent program executable in a general programming language (e.g. Java or C#). In summary, the generator approach is platform-independent and enables the portability to other transformation environments.

3.5 Reuse of Mappings

A key concept of this framework is the reuse of mappings and the automatic derivation of new mappings based on existing mappings. Each mapping is stored in a common repository. The repository represents a knowledge-base and is the basis for the creation and adaptation of new mappings. In the reuse process, reuse algorithms compare elements from the current mapping with existing repository rules in order to find rule candidates for reusing. The output of the algorithm is a set of rules with probabilities which indicate the appropriateness of these rules for the currently observed mapping.

In Figure 8 we present the process for calculating rules for a new source and target structure. The whole process consists of the following steps.

- Step 1: In the first step, a framework user may select at least one source and target element which should be mapped to each other.

- Step 2: In the second step, the similarity calculation is executed. The input of this calculation is the element selection from the first step. This step is descripted in more detail in the next subsection.
- Step 3: After the execution of the similarity calculation, a list of possible rules are identified as possible candidates for reuse. Each candidate has a probability which describes the matching factor between the rules in the repository and the source and target element from the current element tree. The similarity calculation is described in more detail in the next section.
- Step 4: In the fourth and final step, the calculated mappings are adapted and applied to the current source and target structure. The application depends on a definable threshold value, which can be set automatically or manually. The chosen candidate rules are then applied to the current mapping and represented in the mapping editor.

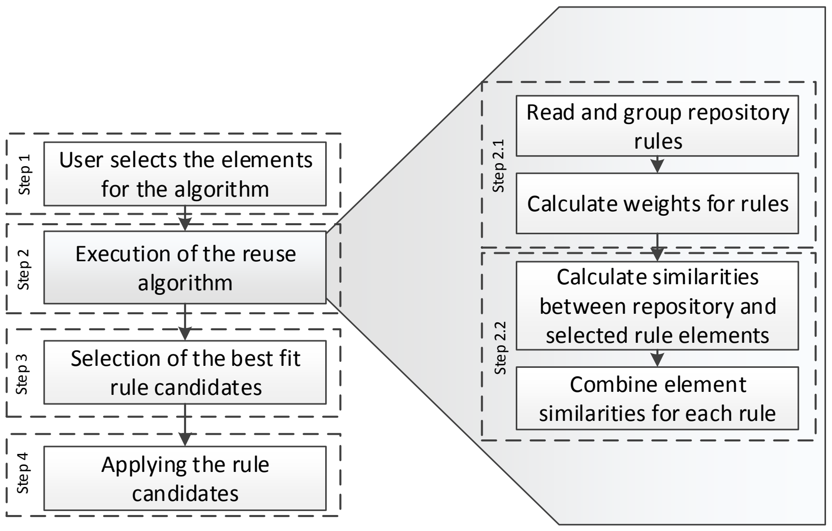

Step 1
User selects the elements for the algorithm
Step 2
Execution of the reuse algorithm
Step 3
Selection of the best fit rule candidates
Step 4
Applying the rule candidates
Step 2.1
Read and group repository rules
Calculate weights for rules
Step 2.2
Calculate similarities between repository and selected rule elements
Combine element similarities for each rule

Figure 6. Process of the reuse algorithm

Based on (Manakanatas and Plexousakis 2006), we may classify reuse algorithms as: (i) reuse algorithms based on isolated element information, (ii) reuse algorithms based on element structure, and (iii) reuse algorithms based on element semantic. Our goal is to create a generic algorithm that may be used in the creation of mappings between any source and target structure. In such a case, we can only rely on isolated mapping information as we do not know in advance which source and target structure are being mapped. Therefore, the used reuse algorithm belongs to the first category. In addition to considering isolated element information, the presented algorithm also considers past executions and previous user choices in order to improve the accuracy of the results.

The individual steps of the algorithm are presented in the right part of Figure 6. The first step (step 2.1 in Figure 8) of the algorithm is a preprocessing for all repository rules. During this step, a number of occurrences of each repository rule is calculated. Based on the number of occurrences, the probability of a repository rule is calculated as:

$$W_{S_r \to T_r} = \frac{N_{S_r \to T_r}}{N_{S_r \to \forall}}$$

$W_{S_r \to T_r}$ is the probability of the rule $S_r \to T_r$ being the appropriate repository rule for the reuse algorithm. $S_r \to T_r$ describes a rule which maps a source element onto a target element. With $N_{S_r \to T_r}$, we denote the number of occurrences of the $S_r \to T_r$ rule in the repository. S_r and T_r are sets of source and target elements of a repository rule, respectively. With $S_r \to \forall$ we present all repository rules that have S_r as the set of source elements. For example, let us consider a repository containing two instances of the rule: $A \to B$ and one instance of the rule $A \to C, D$. In total, there are 3 rules with the A set of elements as source. Therefore, the initial probability that the element set A should be mapped onto B is $W_{A \to B} = \frac{2}{3} \approx 0.67$ and that A should be mapped onto C, D is $W_{A \to C,D} = \frac{1}{3} \approx 0.33$.

In the second step of the algorithm (step 2.2 in Figure 8), user selected elements are matched against the elements from the repository rules. The element comparison is based on element names and done by combining different comparators. A comparator takes two element names and produces a single number representing a similarity between these elements. Currently, we have implemented several string comparison algorithms, such as, Levenshtein (Levenshtein 1966) and Jaro-Winkler (Winkler 1990) algorithms. Each pair of elements can be compared with an arbitrary number of comparators. Similarities calculated by different comparators can be combined into a single value by weighted multiplication of produced values. The weights are chosen globally by a user, in the tool settings, and assigned to all comparators. Therefore, the element similarity is calculated as:

$$S_{E,E_r} = \frac{\sum_{i=1}^{n} (S_{E,E_r,C_i} \cdot W_{C_i})}{n}$$

S_{E,E_r} represents the similarity of the selected element E and a repository rule element E_r. With S_{E,E_r} we denote the similarity of elements E and E_r calculated by the comparator C_i. Comparators produce a normalized similarity that fits the $[0,1]$ interval. Additionally, W_{C_i} is the weight assigned to each comparator by a user and it has a value in the same interval. The sum of all calculated similarities is divided by the number of comparators n for the final similarity to be also normalized and to fit the same interval.

In order to calculate a probability of a repository rule being an appropriate candidate for reuse, similarities between all repository rule elements and user selected elements must be calculated and combined into a single rule-specific value. This is calculated as follows:

$$P_{S_r \to T_r} = \left(\frac{\sum_{i=1}^{n} S_{E_{s_i}, E_{rs_i}} + \sum_{i=1}^{k} S_{E_{t_i}, E_{rt_i}}}{n + k} \right) \cdot W_{S_r \to T_r}$$

$P_{S_r \to T_r}$ represents the probability of a rule $S_r \to T_r$ being a candidate for reuse. With E_s we represent a selected source element, while with E_{rs} we denote a source element of a repository rule. $S_{E_{s_i}, E_{rs_i}}$ represents a similarity between aforementioned source elements. Similarly, $S_{E_{t_i}, E_{rt_i}}$ represents the similarity between a selected target element E_t and a target element of a repository rule E_{rt}. Both user-selected element collections and repository rule element collections are ordered in the same way and comprise the same number of source elements n and target elements k. $W_{S_r \to T_r}$ is a weight factor calculated in the first step of the algorithm.

We should note here that the collection of user-selected elements may contain zero or more source/target elements. If the user initiated the algorithm without selecting any elements, the algorithm will search for the rule candidates containing any elements from a source or target generic element tree. If a user selects one or more source elements, the algorithm considers only these elements instead of all generic tree elements. In the case when, for example, all selected source elements correspond only to a subset of a repository rule source elements, other rule source elements must be also considered. They are compared to the rest of the unselected generic source tree elements to find a match. Only when a match is found for all of these other rule elements, it can be considered as a candidate. This is due to the fact that we consider a rule to be an atomic semantic unit that is either considered for reuse with all of its elements, or completely ignored. We do not consider rules with just a subset of its elements. The algorithm works in a similar way when the user selected elements comprise zero or more target elements.

In the case where a collection of selected source elements has fewer elements than n or a collection of selected target elements has fewer elements than k, then the following formula may be used to calculate the rule probability for reuse:

$$P_{S_r \to T_r} = \left(\frac{\sum_{i=1}^{n} S_{E_{s_i}, E_{rs_i}} + \sum_{i=1}^{m} S_{E_{gst_i}, E_{rs_i}} + \sum_{i=1}^{k} S_{E_{t_i}, E_{rt_i}} + \sum_{i=1}^{l} S_{E_{gtt_i}, E_{rt_i}}}{n + m + k + l} \right) \cdot W_{S_r \to T_r}$$

Two new segments are added to this formula. The $\sum_{i=1}^{m} S_{E_{gst_i}, E_{rs_i}}$ segment represents the calculation of similarities between the repository rule elements that are not paired with any of user selected elements E_{rs_i} and one of the elements from the generic element source tree E_{gst_i}. The number of repository rule elements which are not paired with the selected source elements is denoted with m. The element from the generic source tree is chosen to have the maximum similarity with the element E_{rs_i}. This maximum similarity must be larger than a user defined threshold. Analogously, $\sum_{i=1}^{l} S_{E_{gtt_i}, E_{rt_i}}$ segment represents a calculation of similarities of the unmatched target elements of the repository rule. The number of unpaired repository rule target elements is denoted with l.

4. IMPLEMENTATION

The framework comprises the following components: binders, a mapping editor, a repository, generators and comparison algorithms. The framework is implemented in Java as an Eclipse-based application. Each component is implemented as a plug-in.

For each data schema technology, there is a binder plug-in. The binder plug-in implements the creation of the element tree and the binding of the elements to the concepts in the data schema. If there is no explicit data schema, the binder is responsible for the analysis of instance data and the inferring of a corresponding data schema. Beside the creation of the element tree, the binder offers a specific tree view for the mapping editor.

The mapping editor is the central component which connects different binders, the repository, comparison algorithms and generators. The mapping editor is implemented with the Standard Widget Toolkit (SWT). On the left-hand and right-hand side of the user interface a view for the source and target schemas is implemented. In the middle part of the editor, there is a canvas which allows the drawing of mappings between elements. A property view at the bottom allows the editing of properties, such as names, assignment statements, expression of select statements, or rule conditions.

The data structure of the mapping and element container is implemented with the Eclipse Modeling Framework (EMF). This framework offers code generation of data models and a serialization and deserialization of data in order to store and load mapping models as files.

We use Xtend for the implementation of generators. Each generator is realized as a plug-in and depends on a source and target schema technology, as well as, the transformation execution environment. Xtend allows access on EMF data and offers navigation or analysis of the EMF mapping models.

The repository is currently implemented as a file system directory. All mappings are stored in a defined directory. The repository component reads the mappings from the repository and can iterate over all mappings. The comparison algorithms are implemented in Java.

5. USE CASE

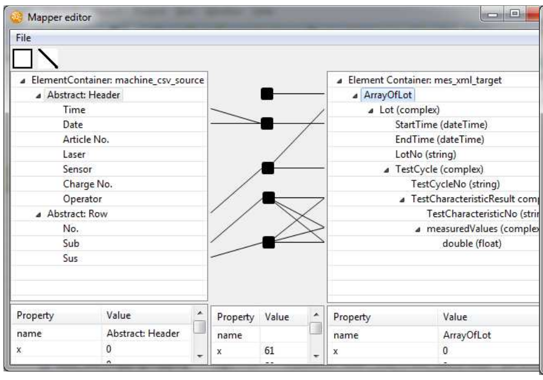
In this section, we present a case study to demonstrate our integration framework. The selected use case concerns measuring thickness of wafers during their production. This measurement is important to ensure the quality throughout the production process. For this purpose, the measurement machines offer different methods, such as, grid, profile or spot measurements. Depending on the selected method, the machine produces different output data. In this case, each machine produces one CSV file per operation containing measured values. For data processing and analysis, CSV data must be imported into a manufacturing execution system. The manufacturing execution system offers data interfaces which allow the import of XML documents conforming to a defined schema. Beside the technical heterogeneity between the CSV format of the source system and XML format of the target system, the import mechanism must overcome the functional heterogeneity between source and target systems. The existence of different measurement methods lead to a variability in CSV files. That is, the MES vendor needs a set of different adapters for the integration of the measuring machines. The manual implementation may be insufficient, time-consuming, costly, and error-prone. Hence, we use our integration framework in order to develop an integration solution which concerns the variability in the machine data.

The integration solution must transform the CSV files into XML files which can be imported into the MES. The source CSV files are structured into a header and a payload section. The header contains metadata which characterize the measurement process (e.g. time, laser, charge/batch number, or operator). The header is followed by the payload section which contains the measured results of the wafer thickness. The payload conforms to the commonly used CSV specification and is structured in a tab delimited table. Depending on the measurement method, the number of measurement layers can differ. This affects the structure of the CSV table. The target XML schema for the representation of the same data comprises *Lot*, *TestCycle*, *TeastCharacteristicResult*, and *measuredValues* elements.

First, the integration framework creates a transformation for a single-layer point measurement. Algorithms automatically recognize the structure of the CSV file. A binding component creates an element tree of the analyzed CSV file structure. Afterwards, the XML schema of the MES is imported into the mapping editor. A binder creates an element tree for the XML schema. Based on the element trees, the mappings between the source and target elements can be described graphically by using the mapping language. The mapping and the created element trees are represented in Figure 9. The mapping consists of the following operators (denoted as O).

- O_1 creates the root XML element *ArrayOfLot*
- O_2 maps the metadata values *Time* and *Date* into the XML element *StartTime*
- O_3 creates for each row in the CSV payload a *Lot* and *TestCyle* element in the XML file.
- O_4 maps *Sub* values (from Sensor 1) into *TestCharacterisiticResult*, *measuredValues*, and *double* XML elements.
- O_5 is similar to O_4 but this operator maps *Sus* values (from Sensor 2) to the corresponding XML elements.

The created mapping is stored in the repository. Subsequently, the acquired mapping-knowledge should be applied for the creation of a new mapping. The measurement machine now processes a double-layer point measurement. The CSV file includes four sensor values per test cycle instead of two values (two values for both layers A and B). The mapping can be derived in a fully automatic manner because the differences between the two CSV files are marginal. The header sections are identical. Hence the operator O_1-O_3 can be applied without changes. Due to the name similarity between the elements (*Sub* -> {*Sub_A*, *Sub_B*} and *Sus* -> {*Sus_A*, *Sus_B*}) the operators O_4 and O_5 can be adapted to these new CSV elements (see Figure 10). The user can execute the reuse algorithm which offers a set of calculated mapping rules. These mapping rules have the highest similarity value and are adapted to the new source and target structure. The user can apply the mapping rules by the selection of rules in a dialog.

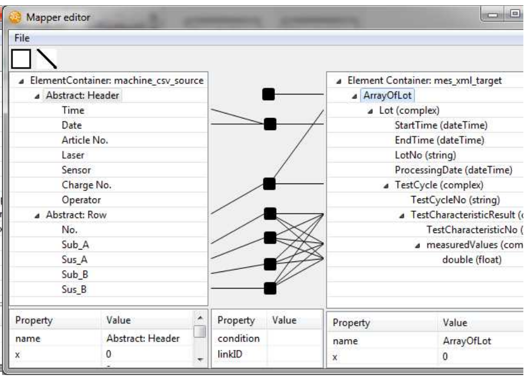


Property	Value
name	Abstract: Header
x	0

Property	Value
name	
x	61

Property	Value
name	ArrayOfLot
x	0

Figure 9. Mapping of single-layer measurement



Property	Value
name	Abstract: Header
x	0

Property	Value
condition	
linkID	

Property	Value
name	ArrayOfLot
x	0

Figure 10. Mapping of double-layer measurement

After the selection of the suggested mapping rules, a developer can check and correct the mappings in the mapping editor. The implemented generator creates a data transformation based on the mapping specification. In this use case, Java code is generated that can be executed in the integration platform. Within the scope of this use case, the integration platform

MuleESB (MuleSoft Inc. 2015) was used. The integration platform reads the CSV files, executes the generated transformation and writes the created XML files into the MES.

6. EVALUATION

We evaluated the integration framework on the basis of this use case and similar applications in the area of machine integration. The mapping editor works well with different data structures. Organizing data schemas as element trees is suitable for the representation of many different data schema technologies. The mapping language provides developers with enough concepts to express all needed mappings and to generate an executable transformation between the participating data schemas.

The graphical notation of the mapping is intuitive and enables a good overview of the mappings. Nevertheless, we find that the graphical representation of complex mappings could be confusing due to the amount of lines. Hence, we consider for the future work a table-based view in mappings. Our mapping tool also improves the usability of the exchange process. Instead of programming a transformation, the tool generates the executable transformation. Generally, the mapping tool supports exchange between various data sources. The representation of data structures as element trees allows the import of schemas from various machines and information systems.

The automatic derivation of mapping rules based on existing mapping rules containing in the repository works well in our use case. The finding of mappings in the repository depends on the quality of the mapping database and the comparison algorithms. The development of sufficient algorithms and the optimal configuration is a work in progress. We assume that automatic mapping is possible but the developer must have the possibility to adapt the suggested mappings in the editor.

Furthermore, the approach improves the efficiency and quality of transformation development. By separating of mapping logic and the transformation execution aspect, it is possible to change the mapping logic without writing transformation code. Furthermore there are different levels of error handling: during the development time of the mappings and the code generation of the data transformation.

7. RELATED WORK

Adapter boxes or protocol converters, for instance, Anybus-adapters by HMS Industrial Networks (HMS 2014), are the easiest way to integrate machines and application systems and to overcome technical and functional heterogeneity. These converters often focus on the translation of technical machine protocols. Additional to this, some converter boxes allow overcoming the functional or semantic heterogeneity. But the transformations are hard-coded into these devices. Our approach allows for dynamic creation of mappings to overcome functional heterogeneity.

In the simplest case, the associated transformations can be implemented by a general programming language, specialized mapping languages (e.g. Altova MapForce) (Altova Map Force 2014) or transformation languages (e.g. XSLT) (Kay 2007). These technologies offer limited concepts for the reuse of transformation knowledge and the automatic derivation of new transformations.

In many cases, integration platforms and middleware solutions, such as, IBM WebSphere, Microsoft BizTalk Server, MuleESB or OpenESB, are used to exchange data between application systems. These platforms usually offer a wide range of different standard adapters, which can be used for the machine integration. These adapters enable the technical integration to systems. The transformation between participating adapters to concrete data structures is still necessary. Our tool supports this task and can be seen as an additional add-on to such an integration platform.

The development concept of the mapping-based framework follows the Model-Driven Development (MDD) paradigm (Stahl and Völter 2006). We use different concepts such as model transformations or model comparison algorithms. Several mapping approaches and environments are proposed in literature. (Wimmer 2008) proposes an approach to model-based tool integration in the context of the ModelCVS project. However, this approach heavily depends on the EMF technical space and it is not easily applicable to other technical spaces. (Bézivin et al. 2005) present the ATLAS Model Management Architecture (AMMA). It provides an extendable core language for specifying platform independent transformations. Authors of the paper argue that for a tool integration process, a specific language should be derived from the core language in order to cover the specific need of that process. However, we feel that this could be burdensome for the users of such a tool as for each integration scenario they need to create new concepts. Our goal is to provide a single and powerful mapping language that can be used regardless of the tools being integrated. Several other mapping approaches can be found in literature, such as, Clio (Miller et al. 2001), Rondo (Melnik et al. 2003), RDFT (Omelayenko 2002), and a UML-based approach (Hausmann and Kent 2003). All of these approaches focus on the integration of certain technical spaces or languages, such as, XML, Relational Databases, or UML. However, none of these approaches allows a single language for the integration of arbitrary technical spaces.

The reuse of transformations is a well-known problem. As it is presented in (Kusel et al. 2013), currently there is a strong focus on the reuse in the implementation phase. However, the reuse across all development phases is not yet accomplished. Our focus is on the design phase. In addition to reuse in transformation languages, our approach is influenced by the ontology alignment approaches. In papers (Euzenat and Valtchev 2004, Gross et al. 2013, Jung 2010, Kappel et al. 2006) multiple ontology alignment scenarios and approaches are proposed. Based on the observations from these papers, we were able to fine tune our algorithm and also see its drawbacks that should be improved in the future.

8. CONCLUSION

In this paper we presented a mapping-based integration framework which enables data exchange between different data structures. The integration framework focuses on the connection of machine data and information systems. The approach consists of different components. The first component reads different data schemas and represents a schema as an element tree. Another component is the declarative mapping language with a graphical notation to specify mappings between source and target schemas. Based on these mappings, generators can create an executable data transformation. A special feature of this approach is the reuse of mappings and the derivation of new mappings from existing mappings stored in a repository. With the help of this knowledge and derivation functionality, the specification of a new mapping can be automated. A special charm of the solution is that in addition to new machines also successively existing machines can be connected minimally invasive through the integration framework. Thus, they get a posteriori industry 4.0 ready and will get able to meet the more and more dynamic market demands more effectively.

In order to evaluate the approach, we presented an application scenario for our mapping tool. The case study concerns the exchange of data between machine data represented as CSV and an MES which allows the import of XML data. The presented framework is suitable for mapping definition and for finding new mappings.

One direction of future work is to improve the user interface of the mapping tool. In case of large data schemas, a mapping diagram could get overcrowded with links and operators. This could be improved by using a tabular view of mappings with less graphical lines between elements. Furthermore, we plan to implement comparison algorithms which take also the semantic of elements into consideration.

Due to the separation between runtime and configuration environment, it would also be conceivable, to develop beside additional transformation models, further value-added services. This could be realized both on premise and on demand as a cloud offer.

ACKNOWLEDGEMENT

Research presented in this paper was supported by the research project SAAMI "Intelligent Integration Framework for Connection of Machines and Information Systems" funded by the BMWi as well as the German Exchange Service and Ministry of Education, Science and Technological Development of the Republic of Serbia as part of the bilateral project "Discovering Effective Methods and Architectures for Integration of Modeling Spaces with Application in Various Problem Domains".

REFERENCES

Altova Map Force, 2014. MapForce – Graphical Data Mapping, Conversion, and Integration Tool. http://www.altova.com/mapforce.html

Bézivin, J., Jouault, F., Rosenthal, P. and Valduriez, P., 2005. Modeling in the Large and Modeling in the Small. *Model Driven Architecture - European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, Twente, The Netherlands, June 26-27, 2003 and Linköping, Sweden, June 10-11, 2004. Revised Selected Papers,* Vol. 3599, Springer, pp. 33–46.

Dujin, A., Geissler, C. and Horstkötter, D., 2014. Industry 4.0: The new industrial revolution – How Europe will succeed. Roland Berger Strategy Consultants, Munich, Germany.

Euzenat, J., Valtchev, P., 2004. Similarity-based ontology alignment in OWL-lite. *Proc. 16th European Conference on Artificial Intelligence (ECAI), Valencia (ES),* pp. 333–337.

Gross, A., Dos Reis, J.C., Hartung, M., Pruski, C., Rahm, E., 2013. Semi-automatic Adaptation of Mappings between Life Science Ontologies. *Data Integration in the Life Sciences,* Vol. 7970, Springer, pp. 90-104.

Hausmann, J.H. and Kent, S., 2003. Visualizing model mappings in UML. *Proceedings of the 2003 ACM symposium on Software visualization,* ACM, pp. 169–178.

HMS Industrial Networks, 2014. Anybus Product Index. http://www.anybus.com/products/prodindex.shtml

Jung, J.J., 2010: Reusing ontology mappings for query routing in semantic peer-to-peer environment. *Information Sciences*, Elsevier, Vol. 180, No. 17, pp. 3248–3257.

Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M., 2006: Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. *Model Driven Engineering Languages and Systems - 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006*, Vol. 4199, Springer, pp. 528–542.

Kay, M., 2007. XSL Transformations (XSLT) Version 2.0, W3C Recommendation 23 January 2007, W3C, 2007 http://www.w3.org/TR/xslt20/

Kusel, A., Schönböck, J., Wimmer, M., Kappel, G., Retschitzegger, W., Schwinger, W., 2013. Reuse in model-to-model transformation languages: are we there yet? *Software & Systems Modeling,* Vol. 14, No. 2, Springer, pp. 537-572.

Levenshtein, V. I., 1966. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady,* Vol. 10, No. 8, pp. 707–710.

Linthicum, D., 1999. *Enterprise Application Integration*. Addison-Wesley Professional, Toledo, OH, USA.

Manakanatas, D. and Plexousakis, D., 2006. A Tool for Semi-Automated Semantic Schema Mapping: Design and Implementation. *Proceedings of the CAISE*06 Workshop on Data Integration and the Semantic Web DisWeb '06*, Vol. 238.

Manouvrier, B., Menard, L., 2008. *Application Integration: EAI B2B BPM and SOA*. Wiley-ISTE, London, UK.

Melnik, S., Rahm, E. and Bernstein, P.A., 2001: Rondo: A programming platform for generic model management. *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ACM, pp. 193-204.

Miller, R.J., Hernández, M.A., Haas, L.M., Yan, L.-L., Ho, C.H., Fagin, R. And Popa, L., 2001. The Clio project: managing heterogeneity. *Newsletter ACM SIGMOD*, Vol. 30, No. 1, pp. 78-83.

Mukhopadhyay S, 2014. *Internet of Things: Challenges and Opportunities*. Vol. 9, Springer, Switzerland.

MuleSoft Inc., 2015. muleESB. http://www.mulesoft.org

Omelayenko, B., 2002. RDFT: A Mapping Meta-Ontology for Business Integration. *Proceedings of the Workshop on Knowledge Transformation for the Semantic for the Semantic Web at the 15th European Conference on Artificial Intelligence (KTSW-2002),* pp. 77-84.

Rautenstrauch, C., 1997. *Integration Engineering*. Addison-Wesley, Bonn, Germany.

Ruh, A., Maginnis, F., and Brown, W., 2001. *Enterprise Application Integration: A Wiley Tech Brief.* John Wiley & Sons, New York, NY, USA.

SISCO, 1995. Overview and Introduction to the Manufacturing Message Specification (MMS). Sterling Heights, MI, USA.

Scholten, B., 2007. The Road to Integration: A Guide to Applying the ISA-95 Standard in Manufacturing. ISA, Durham, NC, USA.

Stahl, M. and Völter, M., 2006. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, Chichester, West Sussex PO19 8SQ, England.

Wimmer, M., 2008. From Mining to Mapping and Roundtrip Transformations - A Approach to Model-based Tool Integration. PhD Thesis, Vienna University of Technology.

Winkler, W. E., 1990. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. *Proceedings of the Section on Survey Research Methods (American Statistical Association),* pp. 354-359.