Automated Data Mapping Specification via Schema Heuristics and User Interaction

Sebastian Bossung[1], Hermann Stoeckle[2], John Grundy[2, 3], Robert Amor[2] and John Hosking[2]

[1]*Software Systems Group, Technical University of Hamburg, Harburger Schloßstr. 20, D-21071 Hamburg, Germany*
bossung@gmx.de

[2]*Department of Computer Science and* [3]*Department of Electrical and Computer Engineering, University of Auckland, Private Bag 92019, Auckland, New Zealand*
{herm, john-g, trebor, john}@cs.auckland.ac.nz

Abstract

Data transformation problems are very common and are challenging to implement for large and complex datasets. We describe a new approach for specifying data mapping transformations between XML schemas using a combination of automated schema analysis agents and selective user interaction. A graphical tool visualises parts of the two schemas to be mapped and a variety of agents analyse all or parts of the schema, voting on the likelihood of matching subsets. The user can confirm or reject suggestions, or even allow schema matches to be automatically determined, incrementally building up to a fully-mapped schema. An implementation of the mapping specification can then be generated.

1. Introduction

Data transformation is one of the most common problems facing systems integrators as source data is often in an inconsistent format or structure for systems wanting to use that data. This requires integrators to implement code for the mapping operations required to convert the data from one form to another e.g. from one XML document format to another. The code to do this is often tedious to write, consisting typically of pages of C++, Java, or XSLT code, and, as a result, tends to be error prone.

In earlier work we have developed a range of domain specific tools to assist in this task, with the intention of reducing the amount of coding required, and, by choosing appropriate metaphors for expressing mappings, to make mapping specification more accessible to a wider group of developers. Domains we have developed such tools for include B2B systems for business data exchange [7] [12], health systems for patient data exchange [10], building and construction for design tool integration [3], and software development environments for software model data and view exchange [9] [17]. While the tools we have developed have generally proved to be very useful, all of them require element-by-element specification of correspondences between one or more elements in a source schema and one or more in a target schema. For large problems this becomes extremely tedious and the tools struggle to scale when visualising and managing the data mapping process. One observation resulting from our work across these domains is that many elements of a mapping specification for a particular schema pair are "obvious" in the sense that a perusal of the schemas along with example data quickly suggests many obvious correspondences. These may be due to elements having the same names, same types, their example data values being the same, or complex type structures may be semantically the same even though element names differ. These heuristics guide us as developers when developing mapping implementations.

Our motivation in this work was to make use of such properties in our data mapping specification and code generation tools. This paper presents a new data mapping specification tool, VisAXSM (Visual Automatic XML Schema Mapper), to assist in automatically determining correspondences between source and target XML schema elements. This tool is the visual front-end for AXSM, which provides an extensible set of schema analysis agents that suggest inter-element mappings using several heuristics. These suggestions are pruned, by user interaction and/or a multiple agent voting strategy, to identify the desired inter-schema mapping specification. The resulting XML-based mapping specifications can be used to generate XSLT, Java or other data mapper implementation code. These generated data mappers take an XML data file in the source schema format and produce a new XML data file in the target schema format. While developed as a standalone proof of concept system here, a combination of this tool with other mapping tools is an obvious extension of this work.

We motivate our research and describe related work, then outline our approach to automated mapping determination and illustrate the use of our prototype tool with an example. The architecture of AXSM/VisAXSM is described and an evaluation of its utility presented. We conclude with a discussion of the implications of our work.

2. Background and related work

Figure 1 shows parts of two XML schemas representing information about lists of people, illustrating the basic

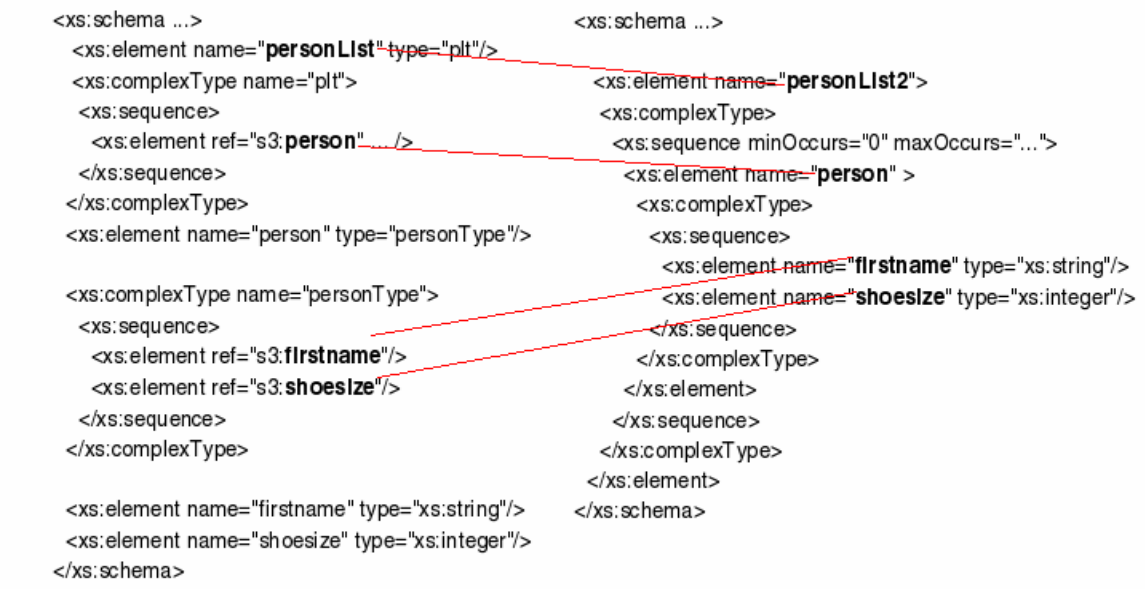

Figure 1: Example schema mapping problem and some correspondences between source and target elements.

issues of the schema mapping problem. Superimposed are some mappings between the two schemas which are "obvious" to a human reader. We should emphasise that these are very small schema fragments, and the difficulty in developing a mapping is typically due to the sizes of the schema involved. In some of the domains we've worked in, these can run to several hundred elements or more. Nevertheless, even in this simple an example, considerable difficulties are evident, including:

- Complex types can be named and declared globally (as in schema 1) or can be declared locally and anonymously inside the declaration of the element that is of the type. The same applies to elements: they can be declared globally and referenced (not used in the example here) or locally inside a complex type.
- There can be multiple elements of the same name in different locations. Schema 1 has two elements named "firstname" and in this case it is quite obvious which of them maps to the "firstname" element in schema 2. However, the relationship is not always this obvious.
- Some non-obvious mappings become evident when example XML data is available e.g. a source "ID" element and target "UniqueValue" for a person always holding the same value in example data files.
- Types may need conversion e.g. "shoesize" may actually be represented as different values and require formulaic conversion. Similarly, names, addresses, descriptions and so on may need reformatting.
- Some elements have no correspondence in the other schema e.g. when the source to target translation is "lossy" or the target format does not have corresponding data in the source.

Programming such mappings by hand is an arduous task. Even with tool support, specifying mappings between large schemas can be extremely time consuming due to the size of the schemas and the number of element mappings involved. Tools supporting this process require facilities for elision, zooming, etc to manage this complexity.

As mapping data between different representations is a common task, much work has been done on the subject, differing mainly in the targeted user base (ranging from expert-programmers to complete non-programmers) and the degree of automation desired. Most EDI and many XML-based messaging technologies have function libraries that programmers use to encode and decode messages [13] [20]. Programmers thus implement message mappings manually using these function libraries, which is time consuming, error-prone and difficult to maintain [10]. Some message mapping systems have been developed [1], but these typically use a low-level representation of mappings incapable of handling complex transformations. Message-Oriented Middleware systems, such as MQ Integrator™ [11], provide message integration tools. These have limited abstract message translation facilities, thus requiring low-level programming. XML-based message encoding and message translators include XSLT, Seeburger's data format and business logic converter [16], eBizExchange [14] and Mapforce [2]. Based on XSLT, these systems lack expressive power and modularity (especially for complex hierarchical mappings) and tools only partially support visual mapping and XSLT script generation. Some Enterprise Application Integration products, such as Vitria BusinessWare™, [19] BizTalk™ [6] and the Universal Translation Suite [5] support message translation for database, message and XML-encoded data. However, these solutions are limited to simple record structures and are

difficult to use. In our own work, we have experimented with several visual approaches to mapping specification, using a variety of visual metaphors. These include the View Mapping Language, which uses a UML like icon and connector approach, the Rimu Visual Mapper, which uses drag and drop links between hierarchical tree structures, and the Form Based Mapper, which uses drag and drop links between business forms [8].

Rahm and Bernstein [15] overview a variety of approaches to schema mapping, and, in particular, algorithms for generating automatic mappings. They introduce notions of composite and hybrid mapper architectures, which we have adopted in VisAXSM, together with the use of both schema level and instance level mapping approaches. Su et al's Xtra system [18] attempts to automatically determine mappings between two DTDs. This is similar to our work, but basing the mapping on DTDs rather than XML Schema, limits significantly the amount of information available for matching. Mapforce, discussed earlier, also includes facilities for automatic discovery of matches, but this is very limited, requiring exact name matching and for elements to be direct sub-elements of known matched elements. It also has significant limitation in handling types associated with the matches.

Examining the deficiencies in this prior work suggested the following requirements for our prototype tool:

- The tool should automatically traverse the two schemas to be matched and suggest correspondences;
- A user interface to the tool must allow the user to focus on parts of the schema mapping at hand and be used to constrain the automatic traversal and suggestions;
- Users should be able to accept or reject suggested correspondences and have the tool provide an updated list of suggestions, providing an interactive environment in which the overall solution space of suggestions is pruned into a usable mapping. Users may even accept suggestions automatically if the probability of correctness is above some user-defined threshold;
- Ideally the tool framework should be flexible enough to incorporate an extensible set of matching algorithms using a wide variety of different heuristics, to be incorporated as "plug ins";
- The ability to generate mapping implementations e.g. in XSLT or Java from a refined mapping specification

3. Our Approach

In our new approach to supporting complex schema data mapping determination and data mapper code generation, source and target XML Schema data files are repeatedly analysed by a set of "analysis agents", each of which applies different heuristics to elements in the schema, to determine if one or more element in each schema are likely to correspond. Data elements "correspond" when, if translating data represented by the source schema to the format described by the target, the source element(s) can be converted into the target elements by either direct copy or a function over their value(s). The analysis agents can be targeted to only analyse small subsets of the two schemas to manage complexity. The architecture permits agents to be added or removed in a "plug and play" fashion. As it is impossible to fully automate a mapping correspondence determination process [15], users interactively accept, reject or defer suggested correspondences. This re-focuses the agents on different parts of the schemas where correspondences are not yet determined. Eventually all elements will have a correspondence, or the user will have specified that none exists, and a data mapper can be generated from the correspondences. The data mapper will take XML data files in the source schema format and produce XML data files in the target schema format.

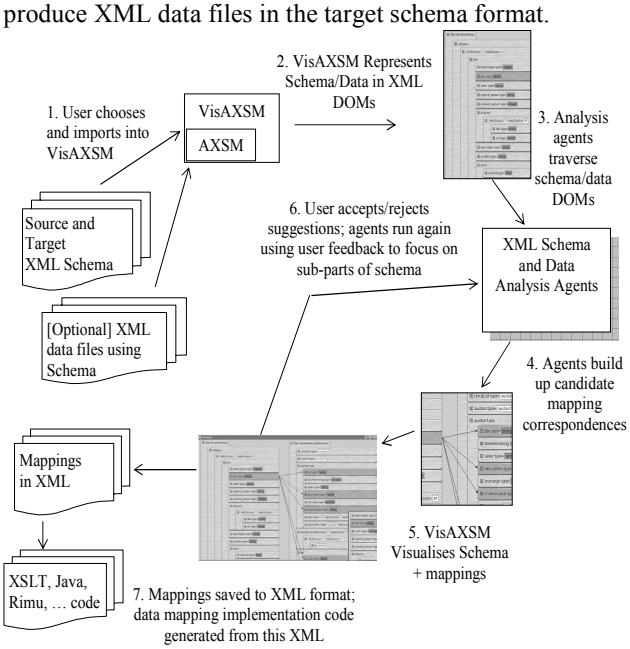

Figure 2: TheVisAXSM mapping process.

The way our VisAXSM automated data mapping tool is used is illustrated in Figure 2. The user first selects a source and target XML Schema (1). We could also use DTDs or other specifications of data formats e.g. RDBMS schema, but XML Schema definitions provide a good range of information on the structure of their XML data files. The user may also optionally specify one or more example data files that are based on the definition in each source and target schemas. VisAXSM parses the schemas and data files and loads them into an extended form of XML Domain Object Models (DOMs) where they can be traversed by the analysis agents (2). The analysis agents examine the schemas using the root nodes as their initial context and generate suggestions of candidate mappings (3). These candidates are also represented using an XML DOM-based structure in the tool (4). The VisAXSM user interface displays the schemas and mappings, using the current

context to elide (often large) parts of the schemas not currently of interest (5). Users indicate mappings they accept, reject or haven't decided on yet, and may refocus the agents on different parts of the schemas manually. This causes the re-execution of the agents (6) and subsequent update of the schema mapping correspondences. This process (3-6) continues until the user is satisfied and saves the mappings to an XML file (7). This file can be reloaded to continue the mapping refinement process or used as input to code generators which produce data mapper implementations. These generated data mappers take XML data files in the source schema format and generate XML data files in the target schema format.

4. Mapping Agent Heuristics

The core of VisAXSM is a set of mapping agents that traverse the source and target schemas and determine possible element correspondences. Because of the complexity of the data mapping problem, these agents can very seldom fully automatically determine correct mappings. Similarly, because of the size of some schemas to be mapped, the heuristics used by agents to determine possible mappings need to be restricted to a (often very small) subset of the overall schema structures.

We have identified a wide range of heuristics that can be applied to XML Schemas or example XML data files based on those schemas, to identify likely element matches. Our approach incorporates these heuristics into "agents", each of which in our VisAXSM tool applies a single heuristic to its input and suggests possible element mappings with differing levels of weighting i.e. probability of correctness.

VisAXSM combines the suggested mappings from all available agents when comparing two schemas portions, giving each distinct mapping a "ranking weight". Combination of individual weights is done using a voting and ballot system with each agent suggesting a weighted vote for candidate matches, with different agents providing different ranges of weights depending on the likely quality of the agent heuristic. All weights for suggested corresponding elements are ranked and the highly likely schema element mappings are highlighted and displayed more prominently. The user can request that rankings above a high threshold be automatically accepted by the tool without showing the user. Similarly, low ranking mappings e.g. suggested by a single agent which uses a heuristic of low quality, can be automatically rejected and not shown.

Some of our VixAXSM mapping agents are listed below with a brief description of their input, heuristic technique, i.e. things they look for in schemas or data structures, and "quality" of resultant mapping correspondence suggestions.

Exact Name Matcher. This agent compares element names in one schema to those in another, suggesting mappings when two have the same tag name. This works well when tag names are the same and unique across each document e.g. PrimaryPatientID in both schemas. It produces many false matches when the same tag name occurs many times e.g. DateValue, although if focused on a small subset of each schema again can work reasonably well.

Partial Name Matcher. This looks for a substring that matches in each name, e.g. PatientName to PrimaryPatientName. Often element tag names for corresponding elements are similar in two documents but not always the same. This agent can use upper/lower case delineation to recognise similarly named items, but if it looks for too small a sized substring many false matches occur e.g. DoctorName and PatientName match on "Name" but are highly unlikely to correspond. The likelihood of correspondence is thus less for this agent, but again focused comparison can reduce false matches.

Levenshtein Name Matcher. This computes a function that works out the "Levenshtein distance" between two names, which is the number of edit operations needed to convert one name into another: the smaller the distance, the closer the match [15]. Again, focusing the agent on subschema produces a better likelihood of matches.

Element Type Matcher. This compares the data type name of elements e.g. PatientID:Integer and UniqueIdentifier: Integer, or PatientRecord:TPatient and ThePatient:TPatient. Like the Partial Name Matcher, it must be focused on a small subset in each schema to avoid large numbers of false positive matches. This matcher ignores the name of elements but if results are combined with those of the Partial Name Matcher better suggestions can result.

Record Type Matcher. This compares record types (sets of elements) rather than leaf element types (single types). For example, Patient:TPatient and ThePatient:PatientRecord may correspond if the complex (multi-valued record types) TPatient and PatiendRecord are the same or can be converted. The agent compares the sub-types of the record type to determine if a match is likely. Because records can contain a large number of elements, some of them also other record types, this matching agent produces lower quality suggestions the more complex the record type.

Synonym Matcher. This can be applied to element tag names or element type names. The Synonym Matcher compares names, or parts of names, to see if they are synonyms of each other e.g. DOB and DateOfBirth are likely to correspond in some way. Similarly, Address and StreetName correspond but the latter target element is part of the source element data, needing a formula to parse and extract the street from the address value in the final mapper.

Domain-specific Matchers are similar to the *Synonym Matcher* but each uses a set of specific domain knowledge e.g. accounting, finance, motor trade, health etc to identify names or types with similar meaning. For example, identifying that TreatmentProvider and Hospital are likely to be the same. Their accuracy can be high depending on the commonality of the corresponding names in the domain.

Exact Data Value Matcher. This looks at XML data records rather than the schema and identifies a correspondence between a single source and target element if their values are the same. This can be generalised to applying simple formulae to the source or target e.g. applying different number or date formatting functions to find a match. Like all data matchers, this must be constrained heavily as XML data files can have hundreds or thousands of records using even very restricted schema. Simple number values can throw false positives but this agent is usually very accurate.

Partial Data Value Matcher. This looks at XML data values from one or multiple elements and computes a likelihood match, similar to the Name Closeness Matcher for element and type names. It must be heavily constrained to a very small subset of source and target elements and the example XML data used must also have a very small number of records to apply to, otherwise it quickly becomes computationally infeasible to use.

When a mapping suggestion from agents is identified by the user as "correct", the matched elements may require data conversion in the generated data mapper. Some agents associate a conversion function suggestion with their mapping suggestions. The user can also specify a conversion function name with the source schema elements as its arguments. This formula is used by the data mapper code generator to implement the required type conversion.

5. An Example

In this section we illustrate how VisAXSM is used on an example data mapping problem. Two fairly simple XML schemas are used but they illustrate many of the complexities that occur when trying to map data from one format to another. Figure 3 shows two different notations for information about auctions. We use these to show VisAXSM specifying a mapping between the schemas [4].

Firstly a user selects the source and target XML schema to map. These are then parsed and a visual representation displayed. This representation is simple and easily understood even by non-technical professionals. Currently it uses a tree-based representation for XML schemas but could be adapted to use other visualization techniques (e.g. form-based). Each notation element of the XML schema is presented as an element in the source or target schema tree as appropriate. Each also has a pop up menu facility (a), providing user access to all mapping and display manipulation actions and information for the element.

To distinguish between external (b) and internal (c) data types VisAXSM uses different colours, in this representation external types are highlighted in yellow and internal types in white. Because every schema element in VisAXSM has its own tree node renderer, it is straightforward to develop different kinds of visual appearances for elements. For example (d) represents a reference element using a more graphical form instead of textual. As every element visualisation has its own menu, this allows navigation between different views by selecting hypertext links.

Our example in Figure 3 demonstrates an unfiltered view of both auction system schemas. However in general showing all information can quickly result in information overload. To prevent this, VisAXSM has several options controlled by context sensitive menus. Figure 4 illustrates several of these. Some are actions across the entire VisAXSM environment.

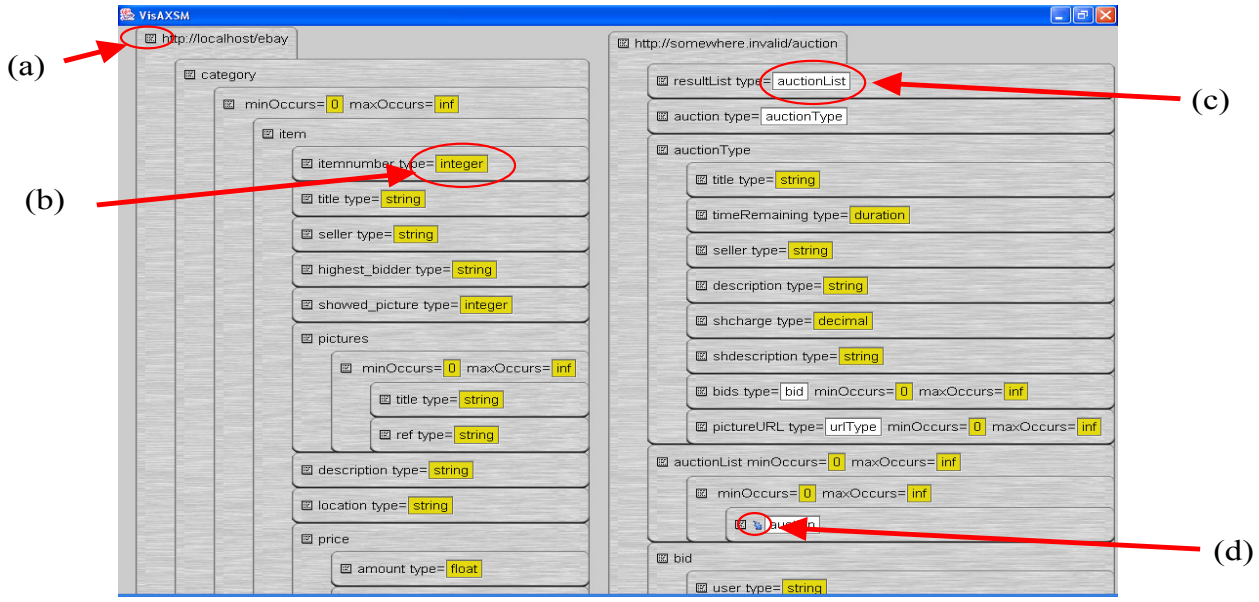

Figure 3: Two sample schemas for an auction system.

Those in (a) are actions for opening schemas, showing a high-level mapping overview, display preferences e.g. arrows to indicate schema element correspondence, and changing other AXSM options e.g. which mapping agents to enable/disable. Example operations at the schema level are shown in (b). The VisAXSM environment provides a view to show all elements of a schema, which can be useful if the developer is familiar with the schema or is looking for a specific element to manually map. Another view of the schema only shows elements for which AXSM mapping agents can provide mapping suggestions using the current mapping context. The opposite view showing which elements in the focus sub-schema AXSM so far have no suggestions available is also useful. Other views can show only elements which are resolved by the developer in this schema or only those not resolved.

Element-level pop-up menus display focused information to a developer. In example (c), the developer has selected a schema element. VisAXSM displays for each possible corresponding element detailed information about which matcher agent voted for this element and the likelihood of the correspondence. This match is voted very likely (vote 1.0) by the three matchers shown (Same name, Partial name and Levenshtein) giving a total vote of 3.0.

Elements can be hidden from the current view to provide better focus for the user. Elements can later be revisualised by the *redisplay all hidden elements* functionality at a schema level or as required as the user re-focuses on different schema elements after accepting or rejecting suggestions. In our experience the ability to selectively hide/show multiple elements and sub-elements is more helpful than in many other tree-based representations. It is common to still show collapsed place-holders in these approaches, but we found such approaches still disturb the user's view of relevant information. Hidden elements are not considered by AXSM matching agents when searching for mapping suggestions. This technique focuses the tool on displayed elements, producing "sub-schemas" for the agents to narrow their search on. As previously discussed, this can greatly improve the performance of many matching agents and prevent AXSM from giving suggestions for elements which are known by the developer not to be relevant. Typically previously mapped elements, whether displayed or not, are not given to matching agents for further suggestions (though this behaviour can be over-ridden by the user if desired). If a target element has more than one source elements the developer can indicate this to VisAXSM by enabling *multiple sources*. As long as this option is enabled, VisAXSM/AXSM will not remove this element correspondence from its internal search and will use its agents to find more correspondence candidates for this target element. The same functionality is available on the source element to indicate multiple target schema elements.

Figure 5 illustrates the process of defining element matches between two schemas with VisAXSM. First the developer has to select a source element. Then VisAXSM runs its mapping agents over the target schema elements in the current mapping context (the displayed elements) to produce a set of element mapping suggestions. VisAXSM highlights the correspondence candidates according to their weighting (by colour ranging from red for low weighting to green for high weighting). Additionally, the developer can switch on drawing of arrows to highlight possible correspondences, however this can be confusing if a large number of possible correspondences are detected. In the example shown, the developer has selected the source schema 'title' element and the matching agents have identified several possible target schema correspondences. In this example, possible mappings include the elements 'title', 'description', 'shdescription' under 'auctionType' record, and other items under other target schema elements.

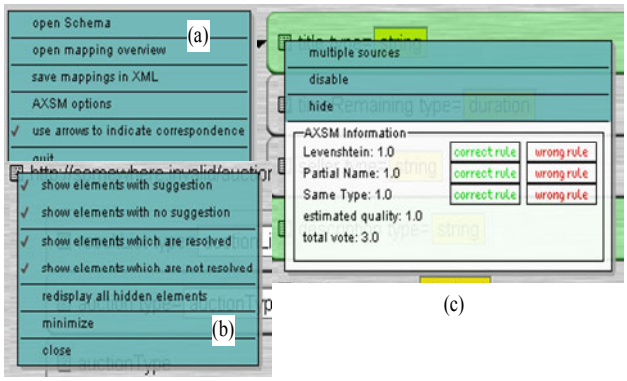

Figure 4: Context sensitive menus and weight information provided by VisAXSM.

The developer can now request information from one or more of the correspondence candidates by selecting their menu (a). VisAXSM displays available matcher agent information and possible actions. In this example the user has selected the target schema 'auctionType.title' element. AXSM reports that the matchers *Levenshtein*, *Partial Name*, *Same Type* have voted strongly for this element as a correspondence. The developer can indicate the correctness of this mapping or notify AXSM that the mapping is wrong. In the former case, the source and target elements are specified as "mapped", changing the next mapping context for the matching agents. If the user rejects the suggestion, AXSM records this information and uses it to refine its other suggestions. In our example the developer decides this is the correct mapping and uses the menu entry *correct rule* (b).

VisAXSM visualizes this unidirectional mapping by drawing both elements in the same colour and an arrow is drawn from source to target. It automatically removes the selected correspondence from its list of possible correspondences for other elements.

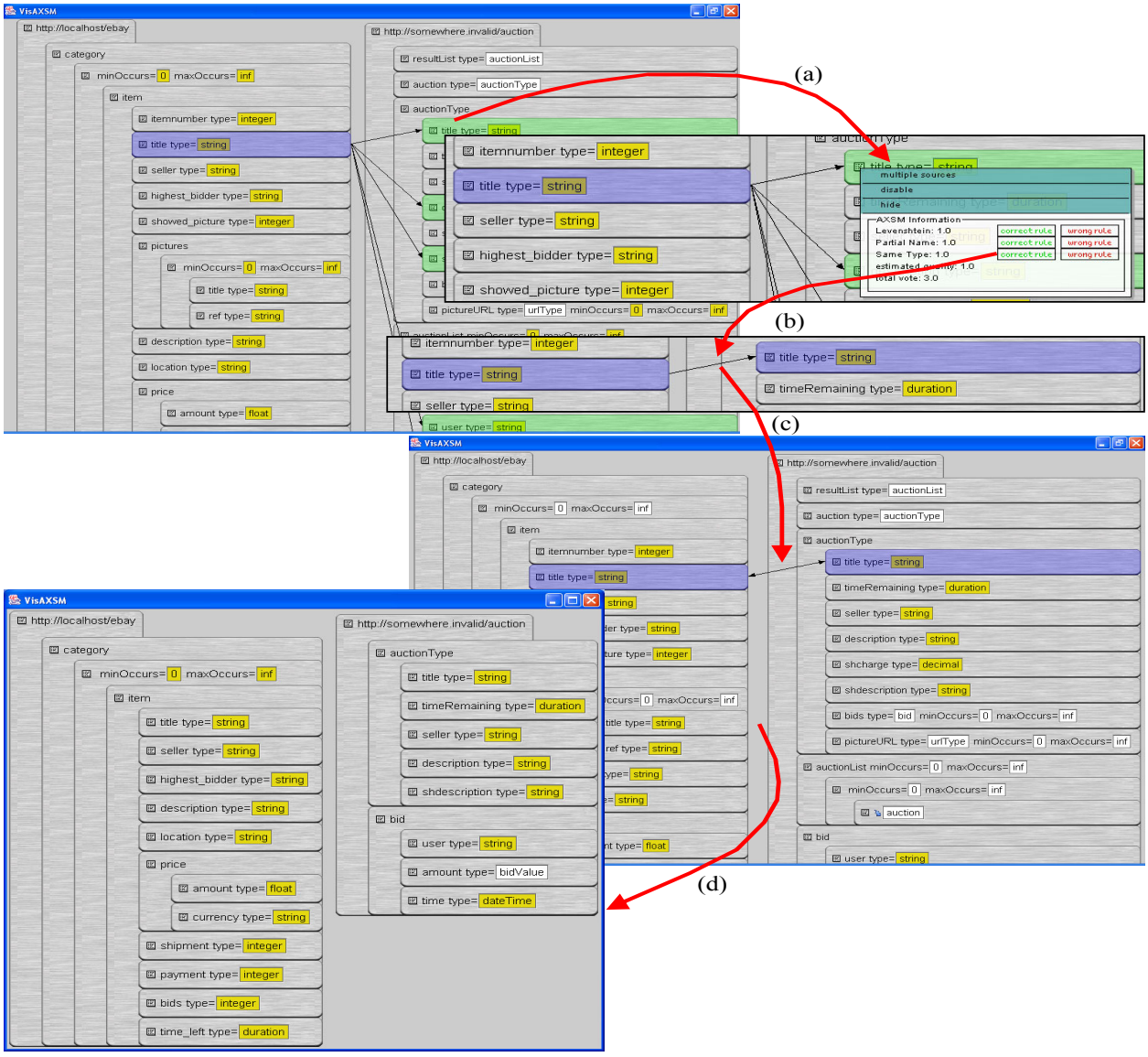

Figure 5: Assigning elements in VisAXSM.

The developer may also specify a formula to apply to convert the source value to the target value if required. The developer may also specify a mapping is "bi-directional" i.e. source may be mapped to and from target. This is shown in example (c). Several source elements may map to a single target element and a single source element to multiple target elements respectively. The user can specify multiple source or target mappings by saying an accepted suggestion is not the only source or target element for the mapping (multiple sources/targets). The matching agents are then re-run and the user may accept another target element for an already-mapped source, or may specify for a different source element the same target element as already mapped to another source element. The multiple source mapping is used in our example for 'auctionType.shdescription', containing the target schema merged information of 'location', 'shipment' and 'payment'.

The final result of this mapping process can be shown in one of VisAXSM's different views, for example displaying only all resolved elements in both of the XML schemas (d). These mappings can now be stored in AXSM's XML-based format and then be used by external tools to generate mappers between the two schemas.

6. Architecture and Implementation

A high-level illustration of VisAXSM's architecture is shown in Figure 6. XML Schemas and data files are parsed and stored within the environment in an extended DOM data structure. Similarly, a data structure holds mapping

correspondences i.e. what elements in the source schema correspond to those in the target. This data structure also provides the context for the analysis agents i.e. what parts of the source or target schema they should focus on. Each mapping item in this data structure records not only which source and target schema elements are related but also: the weighting of the mapping (via votes from multiple agents); whether the user has accepted or rejected the suggested mapping; and display information (shown, hidden, hide/show if another element is hidden/shown, etc).

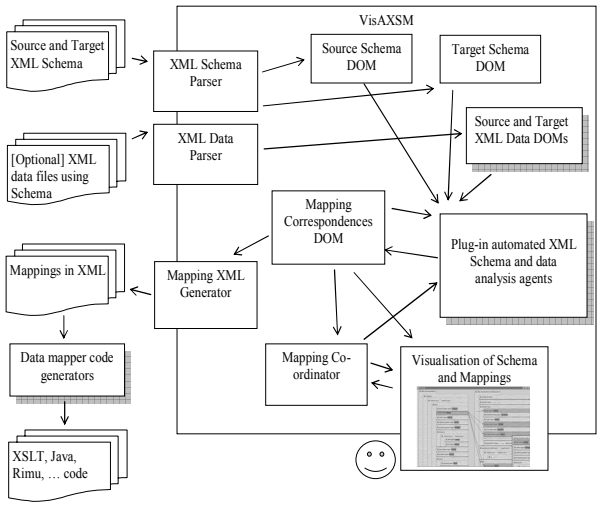

Figure 6: High-level VisAXSM architecture.

The plug-in analysis agents take schemas and/or data information as input, along with the current schema mapping information, and update the schema mapping data structure with their suggested new mappings as necessary. They associate a "weight" against each suggestion they add, along with, where possible, the formula they think may be needed to convert the source to target value. A mapping co-ordinator determines the order in which to invoke the agents, the parts of the source and target schema to offer the agents, and aggregates the results produced by all agents to form an overall "vote" for each suggested mapping. The co-ordinator requests the schema visualisation component to display the current focus sub-schema and associated mapping correspondences to the user after all agents have processed this "mapping context". User interaction updates the mapping context e.g. accepting or rejecting suggestions, changing the elements to focus on, etc and the co-ordinator re-runs the agents to update the mapping correspondences.

We used Java to implement the VisAXSM environment. The Java XML parser and XML DOM APIs were used to manage XML-based import, export and data management. We implemented a wrapper around the standard DOM management functions to provide a range of additional searching and information access functions to simplify the matching agent implementation . We designed an API for matching agents and also for the extended DOM functions to make implementing and adding new agents as easy as we could. Agents are each given the same current mapping context as DOMs which hold source and target schema subsets. However, data value matching agents must query the source and target XML data files loaded by VisAXSM as they need, as pre-computing the parts they want to search is too expensive and varies between different agents.

The VisAXSM GUI is implemented using Swing components with overlay lines drawn to represent the mapping correspondences. We also developed a prototype HTML-based user interface using Java Server Pages, to experiment with delivering the mapper functionality via a web browser rather than as a desktop application. The mapping specification XML file format produced by VisAXSM is currently a bespoke representation. We developed this as we could not find any current standard XML representation that captures the range of information about mappings we need i.e. source/target elements, formulae to convert source value(s) to target value(s), whether the mapping is accepted or rejected by the user, and its ranking weight. We have experimented with data mapper code generation by using XSLT transformation scripts to convert the saved schema mapping correspondences to data mapping code. This code implements a data mapper program, which takes an XML data file in the source schema format and converts it to an XML data file in the target schema format. We used Java as the target data mapper programming language, but could use XSLT itself or a third-party data mapping engine [10].

7. Discussion

7.1 Evaluation

We have applied our VisAXSM prototype to several data mapping problems, using XML Schema with both small (a couple of dozen) and larger (well over one hundred) elements. We have also applied the tool both to XML Schema that are very similar i.e. many close correspondences, and to those that are quite different i.e. with elements that are more difficult to determine matches between and with many items that do not map between source and target data formats. For smaller tests we used schemas representing auction items and order invoices.

In our largest test we used two different approaches that specify Bibtex records in XML Schema, from bibtexml.sourceforge.net and www.authopilot.com/xml/ respectively. The schemas are large (400 and 1000 lines). However, these two schemas are quite similar, especially in the choice of element names, which makes it easier for the mapper to find mapping candidates. There is one big structural difference: the source schema has an element called "nonStandardField" that can store name-value pairs that do not belong to the basic set of fields for each entry (e.g. ISBN numbers for books are stored this way). The target schema doesn't use such a generic approach but has

specific elements to hold these values. This difference occurs often (in each type of publication) but is always exactly the same. There are also some minor differences in representation between both schemas and a complete mapping is not possible. One example of this is that the editors of a publication are a string in the target schema but a nested structure (with each name having its own element) in the source. As there is no matching agent that automatically detects lists, this has to be mapped manually.

	source.xsd to target.xsd		**nyberg.xsd to bibtexml.xsd**		**auction.xsd to ebay.xsd**	
Stat.	Absolute	Relative	Absolute	Relative	Absolute	Relative
1.	73	-	5689	-	63	-
2.	3	-			8	-
3.	9	0.50	72	0.80	8	0.57
4.	9	0.64	72	0.36	8	0.40
5.	11	0.78	430	1.00	10	0.90
6.	12	0.67	190	1.00	11	0.50

1. Total number of candidates: Total number of initial matching suggestions (candidates) the system has generated.
2. Covered elements without correct mapping: Elements from either source or target that have suggestions, but none are correct. This is disturbing to the user as it requires consideration of many options without benefit.
3. Source elements covered: All source elements for which the correspondence with the highest certainty is the correct one. This means that processing the set of suggestions for this element just involves marking the first one as correct.
4. Target elements covered: same for target elements. Note this number can only be different from source number if multi-element rules are involved.
5. Source elements covered manually: number of source elements covered by the largest possible mapping that is achievable using manual tools.
6. Target elements covered manually: The same for the target side.

Table 1: Example mapping results [4].

Table 1 shows some examples of mapping statistics with the invoice, auction and bibtex example schemas. The relative column shows the percentage of covered elements compared to the total number of elements in the schema. In the large bibtex case study, manually starting by making the root level "entry" elements correspond reduces the search space of 5689 down to 2327, with a large number of source and target elements able to be automatically mapped.

While VisAXSM is a proof-of-concept prototype, trial users have found the tool effective and straightforward to use. The ability to selectively hide and show different parts of source and target schemas to manage complexity is useful whether or not the mapping agents are used. Both a Swing-based GUI interface and JSP-based web interface were prototyped. The former has proved to be more effective for larger schema, as it provides better control of schema elision and higher-level visualisation of correspondences between schemas. Further refinement of the user interface is looking to provide more automated display and hiding of schema items and mappings.

7.2 Advantages and Disadvantages

Our experiences with VisAXSM have shown that mappings between schemas with many close correspondences can be done surprisingly quickly, even if the schemas are very large. A business example with over one hundred elements in each source and target schema is able to be completely mapped within minutes using VisAXSM. If agents that determine matches with a high weighting repeatedly find good matches, users can reduce the search space rapidly. This is because most agents work best when restricted to small schema subsets. Typically, after a few high-level record matches are made, the large number of remaining matches are found accurately.

Experiments with VisAXSM have shown that the user can allow the tool to automatically accept suggested matches from agents when even a moderately high correlation is reached between the agents. However, the quality of suggested mappings can vary greatly depending on the current mapping focus, similarity of the schemas, and weightings of mapping agents in use. The user can always review some or even all of the mappings using the visual display at any time, hence they can reject any they find that have been inadvertently marked "correct" when in fact the user knows they are not, and this forces VisAXSM to re-run agents on the new subsets of unmapped elements. We found the approach of having plug-in agents worked well, and we were able to add new agents from time to time to the tool with no impact on the tool or other agent implementations. The overall approach appears very promising for data mapping problems where there is reasonable closeness between the schemas being mapped (i.e. most elements in each schema map to the other and names, types and record structures are substantially similar or the same).

Our approach encounters problems in expected circumstances – when most schema names, types and record structures are very different. We found that the agents either couldn't make any suggestions or their correlation was very low, particularly when generated and non-generated schemas were compared (e.g. mapping element "Field027" and "PatientName" fails for all agents except the data matchers, which can't be suitably focused on subset schema elements). We argue that mapping schemas that contain generated names is beyond the scope of our approach. Other problems were encountered with schemas with huge variations in naming conventions and record organisational structure. However, it is important to realise that our tool can still be successfully used to manually visualise parts of the schemas and to specify accurate mappings. We found the agents provide little useful suggestions in these circumstances and the user ends up manually specifying most of the correspondences. This was one of the problems we were trying to overcome so our approach could be considered unsuccessful in this case.

7.3 Future Work

Our plug-in approach to extending the matching agents proved successful. However, the agent co-ordinator currently has little knowledge of the characteristics of available agents and the ordering of agent invocation could

be enhanced. This would have the advantage that if an agent determines candidate mappings of good likelihood, agents executing after it can use this to inform and constrain their own processing, improving the quality of their weights.

We have only experimented with fairly basic data mapper code generation from these mapping specifications to date, using XSLT transformation scripts. This needs further investigation to demonstrate that very complex data mapping implementations can be successfully generated from the specifications produced by VisAXSM.

The current version of VisAXSM does not directly support more complex mappings or operations, (e.g. merging of strings and converting numbers to strings cannot be easily expressed or represented). Instead, the user must provide a formula which will carry out the required data conversion but the mapping correspondence looks the same as any other. As these kinds of mapping operations are common, the tool should provide some higher-level representations of such field-level transformations.

We are developing a concept of schema element "rendering plug-ins", similar to matcher agent plug-ins but providing new visual element representational and manipulation support. The idea is to allow these rendering plug-ins to be placed on the screen and be used to represent complex mapping operations, different kinds of schema elements, to provide context-sensitive tailored interaction, etc. The benefit of using plug-in rendering units is both improved direct visual feedback to the user and support for extensible schema element presentation and manipulation within VisAXSM. An example of using this approach would be in developing matching agents with real-time simulation, where developers can create mappings, add mapping functions and see the results of their mapping correspondences live on the screen with example data.

8. Summary

Identifying data mapping correspondences between two complex schemas and implementing a data mapping system to convert between them is very challenging. We have developed a proof-of-concept prototype, VisAXSM, which uses a combination of automated schema analysis agents and user interaction to address some of the problems in this domain. XML Schema are inspected by a number of agents, each incorporating a different heuristic and producing a set of candidate mapping correspondences from elements in the source schema to elements in the target. The user reviews these suggestions, presented in a high-level graphical form, accepting or rejecting them as necessary. These user interactions constrain the remaining search space and focus the agents on unmapped subsets of the schemas for further analysis. Once this process is complete, data mapper implementation program code can be generated from the final mapping specifications. These programs convert XML data in the source schema format to the target schema format. Applying our prototype to several example data mapping problems has shown it to be a promising approach to data mapping specification.

References

[1] Aditel Corp. ETS for Windows™, www.aditel.be, viewed June 2001.

[2] Altova, http://www.altova.com/products_mapforce.html

[3] Amor, R.W., and Hosking, J.G. 'Mappings: the glue in an integrated system'. In Scherer, R.J. (ed) Product and process modelling in the building industry, Rotterdam, The Netherlands, A.A. Balkema Publishers, 117-123, 1995

[4] Bossung, S., Semi-automatic discovery of mapping rules to match XML Schemas, Department of Computer Science, The University of Auckland, 71pp

[5] Data Junction Corp, Universal Translation Suite™ General Information, www.datajunction.com, viewed May 2001.

[6] Goulde, M.A. Microsoft's BizTalk Framework adds messaging to XML. *E-Business Strategies & Solutions*, Sept. 1999, pp.10-14

[7] Grundy, J.C., Bai, J., Blackham, J., Hosking, J.G. and Amor, R. An Architecture for Efficient, Flexible Enterprise System Integration, Proc 2003 Intnl Conf on Internet Computing, Las Vegas, June 23-26 2003, CSREA Press, pp. 350-356.

[8] Grundy, J.C., Hosking, J.G., Amor, R.W., Mugridge, W.B., and Li Y., Domain-Specific Visual Languages for Specifying and Generating Data Mapping Systems, JVLC, *15:3-4, 243-263, 2004.*

[9] Grundy, J.C., Hosking, J.G., and Mugridge, W.B., Inconsistency Management for Multiple-View Software Development Environments, IEEE Transactions on Software Engineering, 24(11), November 1998, 960-981.

[10] Grundy, J.C., Mugridge, W.B., Hosking, J.G. and Kendall, P., Generating EDI Message Translations from Visual Specifications, Proc 2001 IEEE ASE Conf, San Diego, CA, 26-28 Nov 2001, IEEE CS Press.

[11] IBM Corp, MQ Series Integrator, www.ibm.com, viewed May 2001.

[12] Li, Y., Grundy, J.C., Amor, R.A., and Hosking, J.G., A data mapping specification environment using a concrete business form-based metaphor, Proc IEEE HCC'02, Arlington, USA, 3-6 September, 2002, IEEE CS Press,158-167

[13] Lincoln, T., Spinosa, J., Boyer, S., Alschuler, L., HL7-XML progress report. In Proceedings of XML Europe '99, Alexandria, VA, USA, 1999, pp.733-736.

[14] OnDisplay, CenterStage eBizXchange, www.ondisplay.com.

[15] Rahm, E., Bernstein, P.A., A survey of approaches to automatic schema mapping, The VLDB Journal 10: 334-350 2001, Springer Verlag

[16] Seeburger Corp, SEEBURGER data format and business logic converter, www.seeburger.de/xml/, viewed May 2001

[17] Stoeckle, H. , Grundy, J.C. and Hosking, J.G., Approaches to Supporting Software Visual Notation Exchange, Proc 2003 IEEE HCC, Auckland, New Zealand, Oct 2003, IEEE, 59-66.

[18] Su, H., Kuno, H., Rudensteinern, E.A., Automating the Transformation of XML Documents, Proc Workshop on Web Information and Data Management, 2001.

[19] Vitria Technolgy Inc, Vitria BusinessWare, www.vitria.com.

[20] Wallin, G., A new look at EDI healthcare. *Health Management Technology*, vol.20, no.5, June 1999.