Preprint of article appearing in JVLC: Grundy, J.C, Hosking, J.G., Amor, R., Mugridge, W.B., Li, M. Domain-specific visual languages for specifying and generating data mapping system, Journal of Visual Languages and Computing, vol. 15, no. 3-4, June-August 2004, Elsevier, pp 243-263

Domain-Specific Visual Languages for Specifying and Generating Data Mapping Systems

J.C. Grundy, J.G. Hosking, R.W. Amor, W.B. Mugridge, and Y. Li
Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand

Abstract

Many application domains, including enterprise systems integration, health informatics and construction IT, require complex data to be transformed from one format to another. We have developed several tools to support specification and generation of such data mappings using domain-specific visual languages. We describe motivation for this work, challenges in developing visual mapping metaphors for different target users and problem domains, and illustrate using examples from several of our developed systems. We compare cognitive dimensions-based evaluations of the different approaches and summarise the lessons we have learned.

1. Introduction

One of the most common problems in computing is the need to keep multiple views and representations of the same information consistent [13, 16, 32]. Our work in this area has explored consistency management and data mapping approaches in domains as varied as software engineering tools [15], computer integrated construction [2, 3], clinical decision support [16], web services [44], adaptive user interfaces [17], visual notation mapping, and enterprise application integration [6, 18]. Much of this has focused on making consistency management more accessible, via domain-specific languages and tools to allow domain specialists to specify mappings using notations and metaphors appropriate and familiar to end users.

In this paper, we describe several domain-specific approaches to specifying mappings between different data representations, a significant solution component of the consistency management problem. These vary in the sophistication expected of the end user, from skilled programmer through to business analyst. In each case, we have chosen a metaphor for the visual mapping language appropriate to the user, and, in the process, have deliberately limited the expressive power of that language. We begin with an overview of the end user domains and metaphors used, before examining each metaphor in turn for tradeoffs and benefits. We then compare and contrast the solutions we have developed along with those in related work, and summarise the lessons learned from our experiences.

2. Motivation and Overview of Approaches

Complex information structures occur in a wide range of domains. Very often a domain requires the same information to be represented in a variety of ways. This occurs when different organizations have developed tools which use different data formats; when legacy systems and newly-developed systems must be integrated; when different organizations develop competing "standard" representations; and when quite different structures are needed for different tasks, e.g. data analysis versus data vizualisation [3, 4, 16, 11, 25]. To illustrate the nature of this "data mapping" problem, Fig. 1 shows two example messages representing medical patient treatment information (shown in an XML format, produced by parsing the original surface syntax) [16]. The left message encodes the treatment data using a "deep" structural hierarchy (Patient -> Visits -> Treatments). The right message encodes (mostly) the same data, but uses a flatter format.

To translate the messages we need to apply a variety of field-, record-, and record collection-level translations between these two representations of the patient treatment data. A

number of formulae, some dependent on source message content (e.g. the treatment cost and treatment units), need to be applied. Some structures in the messages repeat, such as the list of treatments required, and these may be organized in quite different ways e.g. a list of Treatments in the left message is grouped into Primary and Other treatments in the right message. To translate the right message into the left, we need to apply data mappings to convert the flat structure into the deeper hierarchical one. To do this several fields and collections must be merged or split e.g. the patient name, dates and address merged.

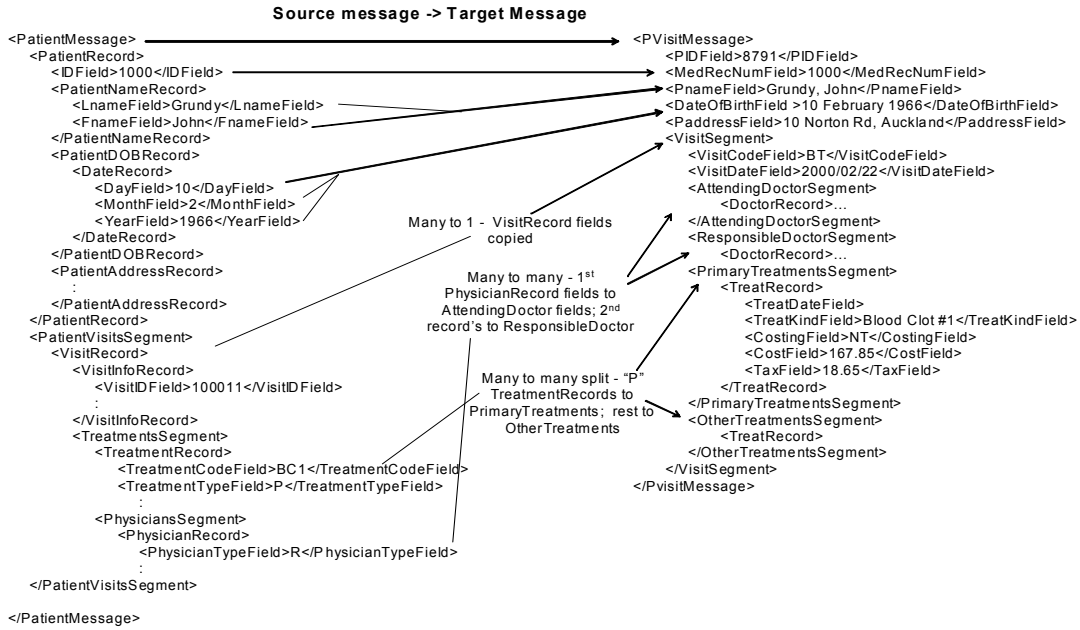


Figure 1. Illustration of mappings between XML formats

This example is typical of the sorts of data mapping problems we have come across in a variety of application domains. However, mappings are usually much larger, often with hundreds of records and fields [2, 6, 16]. This means developers need high-level support to express and manage their message mappings. Typical types of data element mapping [5, 25] include:

- Simple field to field equalities which may involve simple data format or unit conversion, e.g. the IDField and MedRecNumField in Figure 1 represent the same data.
- Simple formulae relating several fields on one side to one field on the other side, for example the FnameField and LnameField values are concatenated to form the PnameField
- Formulae relating multiple objects or records, possibly spanning parts of a type hierarchy, e.g. the name mapping above maps from a PatientNameRecord subrecord of a PatientRecord subrecord of a PatientMessage record to the PnameField of a PVisitMessage.
- Formulae mapping collections of objects or records on one or both sides. This could involve 1:n, n:1 or m:n object or record mappings, and involve merging, aggregation, selection, or partitioning of the collections involved. Some examples are shown in Figure 1.

Other mapping issues that need to be addressed include:

- The ability to handle specification of default values, when data is incomplete
- Specification of the directionality of a mapping specification (if not implicit in the language): can the same mapping specification work in either direction?
- Specification of the order in which data elements are mapped (if not implicit in the language, i.e. the flow of control needed to create a complete mapping correctly
- The need to use meta-data for data formats to assist in defining mapping specifications

- The need to translate data mapping specifications into efficient implementations, and whether these permit mapping of incremental changes from one side to the other (and how efficiently)
- The completeness of the mapping described, enabling a mapping system to determine whether partial information is being transferred.
- The syntax of the data repositories can be intrinsic to the mapping approach or explicitly specified by adjunct processes in the mapping tool e.g. CSV files or EDI messages translated into XML data structures.
- Transaction management for change propagation can be a characteristic of the mapping specification or handled individually within mapping environments.

Specification Language Name	**Target User Group**	**Target Application Domain**	**Assumed User Programming Ability**	**Data Structure Visualisation Metaphor**	**Mapping Visualisation Metaphor**
View Mapping Language (VML) [2]	Professional Programmer	Architecture & Engineering design tool integration	High	UML-like class diagrams	Iconic with wired attachments
Rimu Visual Mapper (RVM) [16]	Data Base Administrator	Health Sector Messaging Systems	Moderate	Hierarchical type structure	Mapping segments wired to elements
Form-Based Mapper (FBM) [26]	Business Analyst	Office automation	Low	Business forms	Wires between form elements

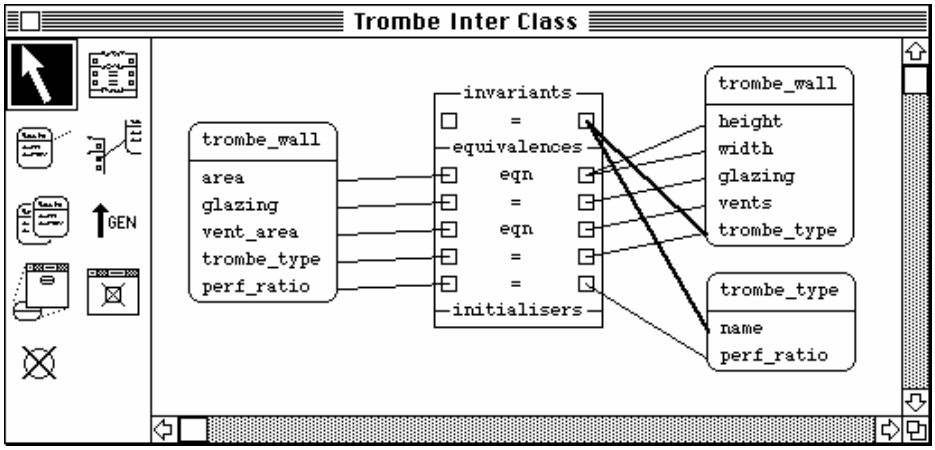
Table 1. Characteristics of the three visual mapping languages examined

Table 1 provides a summary of the characteristics of three domain-specific visual languages and their underlying metaphors that we have developed for data mapping specification. In each case, our approach is to visualise each schema's data structure side by side, with mappings specified by visually linking between elements, in an analogous fashion to Fig. 1, but at a type rather than instance level. The "links" between elements vary in expressive ability as discussed later. In each case we have developed tools for expressing specifications and generating implementations with the ability to "reuse" the visual specifications populated with instance data to visualize mapping instances at a comparable abstraction level to the specifications. For this paper we are not concerned with surface syntax, assuming schema or instances are represented in a DOM. Our tools use several approaches to parse surface syntax into this form.

3. The View Mapping Language

The View Mapping Language (VML), and its graphical form VML-G, aim at capturing relationships between design tool views [2]. The design tools studied for this language came from the architecture, engineering and construction (AEC) domains and need to interoperate to achieve a coordinated and consistent building design. A standard approach to achieve this is to develop an Integrated Project Database (IPDB) covering the set of data requirements of the tools being integrated. With such an IPDB all interoperating tools need only map from their internal representation to the IPDB, and vice-versa, to allow consistent information flows between the tools. Conceptually this is simple. However, the IPDB for such a domain is large (the Industry Foundation Classes standard for AEC [21] has over 500 classes) and representation of similar concepts in the design tools can vary considerably. Enabling the developers of an IPDB system (professional programmers) to capture the essence of the mapping required between representations and specify detailed correspondences between them was the main goal of VML.

VML is a high-level, declarative language for describing bidirectional correspondences between two arbitrary schemas. VML eschews notions of target and source schemas in mapping definitions. As far as practicable, a VML definition treats both schemas as equal partners. This is different to most other visual mapping approaches [5, 23, 16, 27, 37]. VML also removes many distinctions between entities and attributes, to allow mappings between them to be specified in the same way as attribute-attribute mappings. A VML mapping consists of a specification of the schemas to be mapped between, and a set of correspondences between entities and attributes (called *inter_class* specifications) to describe how the mapping is to be achieved. An *inter_class* definition details: entities from the schemas that take part in the mapping; an optional set of conditions which must hold to apply the mapping (*invariants)*; relationships between data in the entities (*equivalences*); and, optionally, initial values for attributes when an object is created (*initialisers)*. The invariants serve two purposes in the mapping: when mapping in one direction they constrain the set of objects for which a particular mapping will be applied: in the opposite direction they provide initial values for attributes in the newly created objects.




```
inter_class([trombe_wall],[trombe_wall, trombe_type],
         invariants(trombe_wall.trombe_type = name),
         equivalences(area = height * width,
                      glazing = glazing,
                      vent_area = sum(vents=>(height * width)),
                      trombe_type = trombe_type,
                      perf_ratio = perf_ratio)
                                          ).
```

Figure 2. VML-G (top) and VML mapping for a simple correspondence between classes

VML-G uses a single icon to represent an *inter_class* definition (see centre icon in Figure 2). This has three sections corresponding to the three sections in an *inter_class* definition. These allow *invariants*, *equivalences* and *initialisers* to be grouped into localized areas in the icon and provide a visual separation of these distinct functions. The other icon type in VML-G denotes an entity taking part in the mapping with the *inter_class* (e.g. the other icons in Figure 2).

Creating a mapping within the visual environment consists of placing an *inter_class* icon between icons for the entities involved in a particular mapping. Entity icons specify the name of the entity (with optional schema and version information) and attribute and method names defined in the entity. Each individual equation, function, or procedure has a row in the *inter_class* icon. Each row end has a box to which the attributes and entities involved in the mapping element are connected. Wiring from an attribute or entity to a box connects it into that equation. Wiring an attribute or entity to a section label in an *inter_class* icon creates a new equation for that attribute or entity. Each row has a symbol defining the type of mapping being defined. These are:

= a direct equality between an attribute (or entity) in one schema and that in the other (perhaps with type conversion). These 1:1 mappings are distinguished as they occur often.

eqn the attributes or entities in one schema are related to attributes or entities in the other schema through an equation (i.e. an enforceable constraint) that is not a 1:1 equivalence.

func the attributes or entities wired together are mapped via a functional (declarative) mapping.

proc a procedure maps between the specified attributes or entities. Different procedures are needed to map in each direction as in general a procedure is not automatically reversible.

Fig. 2 shows a VML-G and equivalent VML specification of a mapping between sets of classes. A 1:1 correspondence is specified between every *trombe_wall* object in one schema and every *trombe_wall* and *trombe_type* in the other schema. All attributes in one class map directly to attributes in the alternate classes (and vice-versa). There is a single invariant which matches *trombe_wall* and *trombe_type* objects with an equivalent value in the specified attributes.

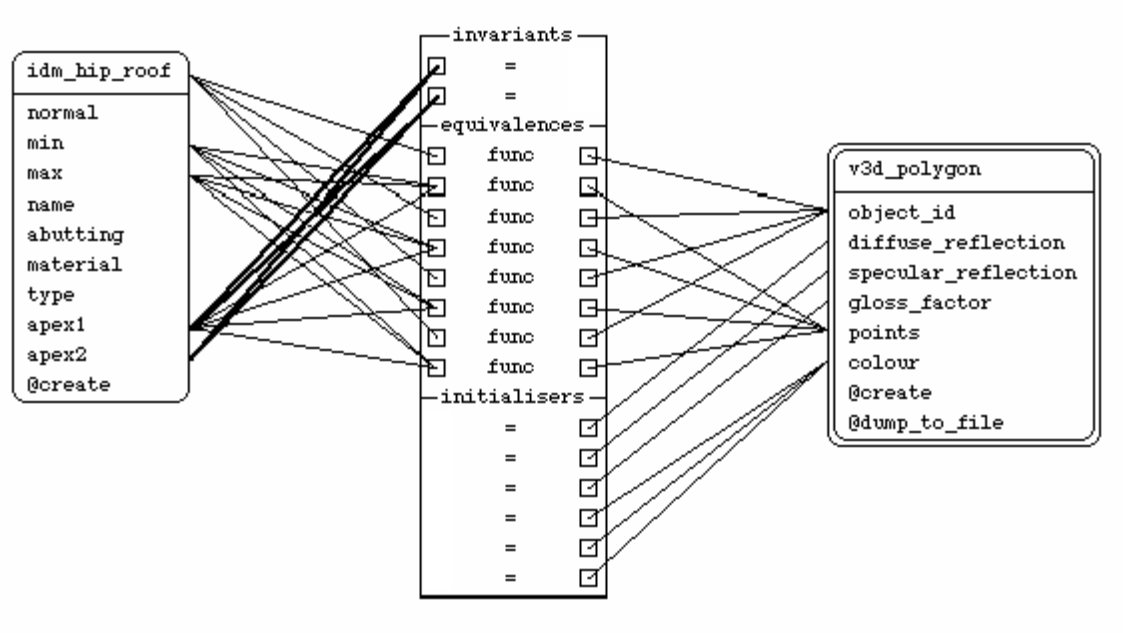


Figure 3. VML mapping with complex wiring

Fig. 3 shows a more complex mapping between each object of type *idm_hip_roof* and a collection of objects of type *v3d_polygon* (collections have a double line round the icon). In this mapping *invariants* on attributes of *idm_hip_roof* constrain when this mapping may be applied. *Initialisers* specify how to instantiate *v3d_polygon* objects. A constant number of *v3d_polygon* objects (four) are mapped from an *idm_hip_roof* object, controlled via references to objects in the set of *v3d_polygons*. It is straightforward to gain a high level understanding of the mapping from this specification and thus check its completeness (e.g., 'Why aren't the *normal*, *name*, *abutting*, *material* and *type* attributes mapped?'). However, mapping detail is hidden (e.g., *func* does not show the type of mapping being prescribed and *eqn* formulae are not visible). Users can open a mapping element to view and edit this textual detail. The example also shows how a mapping specification can become visually untidy if there are many overlapping links, a limitation on the efficacy of this metaphor. This can be addressed via multiple views of parts of the *same* mapping.

A complete textual mapping specification covering the entire mapping can be created for use in an IPDB. Our IPDB framework includes an interpreter for VML mappings allowing data to be mapped between the IPDB and a design tool, either by batch or incremental update of the data stores. Inconsistencies can be viewed, and appropriate action taken to ameliorate conflicts.

4. The Rimu Visual Mapper

The Rimu Visual Mapper (RVM) targets mappings between health message formats, such as those in Fig. 1 [16]. The normal approach to this problem is to develop a custom translator in a language such as C++. Such translators are very tedious to write, involving tens of pages of error prone coding by experienced programmers. Our aim in developing RVM, collaboratively with Orion Systems Ltd, was to minimise professional programmer involvement, and dramatically reduce the time and cost to develop and maintain such translators. Target RVM users are Database Administrators (DBAs) maintaining information systems that health messages are exchanged between. DBAs are familiar with data structuring and formatting issues, but have less programming experience than users of VML. Specifically, they have some understanding of procedural programming, including parameters, and familiarity with spreadsheet expression programming but not object-oriented or declarative programming expected for VML users.

Health messages consist of hierarchical record (type) structures, with repeating and optional elements. Inheritance is not used in the type definitions, nor associated behavioural code, in contrast to VML. The mapping task is thus one of data translation, however, the structural manipulations can be complex, particularly those involving repeated elements.

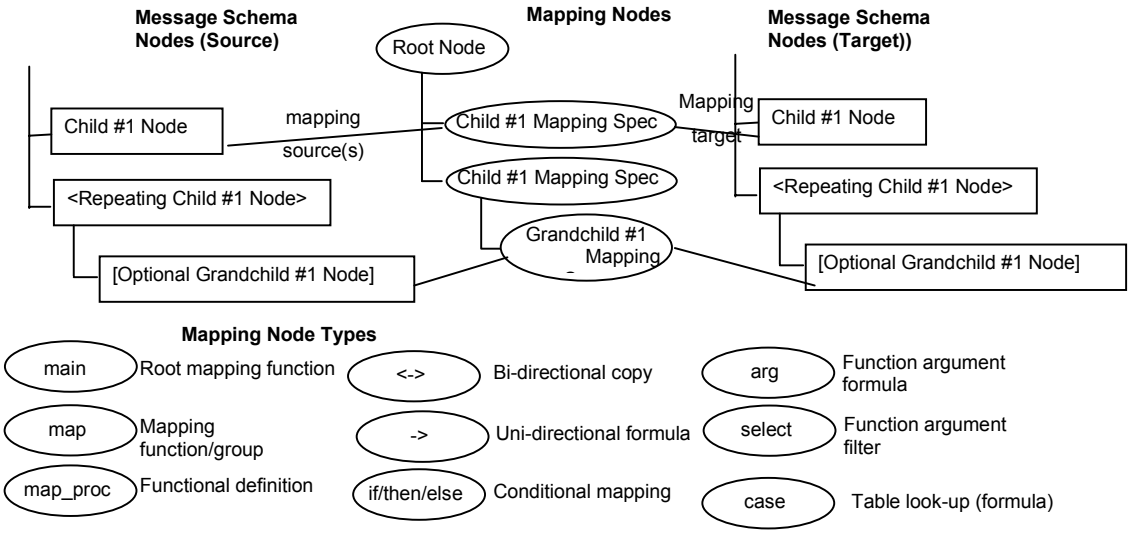


Figure 4. Basic RVM notation

Fig. 4 shows the basic elements of the RVM mapping notation, and Figs 5 and 6 show examples of its usage. The main features of the notation are:

- The visualisation metaphor chosen is a tree showing the hierarchical record structure, one element per line, indented to reflect the type hierarchy. Target end users are used to viewing message data structures in a hierarchical form, so this is a natural metaphor for them.
- Angle brackets and square braces indicate that elements are repeated or optional respectively
- Element mappings are also represented as a tree, with each element representing a mapping between one or more data elements on each side of the mapping.
- An element's mapping direction is shown by an arrow; simple equivalences are bidirectional.
- The ordering of mapping elements in the tree determines their order of application, meaning an explicit procedural sequencing of mappings, rather than VML's declarative approach.
- Levels in the mapping element tree allow for grouping of related mappings.
- If-then-else and case elements allow for conditional mappings.
- Mapping functions defined separately from the main mapping provide procedural abstraction. These take as parameters types from each side of the mapping. Function applications take

"instances" of the types as actual parameters. If a source is a collection, the mapping either selects one item and transforms it, or multiple items which it transforms one-by-one.

Figure 5 screen (1) shows two message schema (those from Fig. 1) and several mapping specifications in a prototype of RVM. The "main" mapping node (a) groups all mappings from one message to another. The first child mapping (b) groups a sequence of mappings specifying how to translate PatientMessage patient information into a PVisitMessage's fields. Node (c) specifies that MedRecNumField and IDField are (bidirectional) copies. The PIDField value is set by a default value generated by a calculation (d). The PnameField value concantenates the PatientNameRecord's LnameField and FnameField values (e). Spreadsheet-style formulae specify these unidirectional calculations. The DateRecord fields are merged to DateOfBirthField by a unidirectional function application (f). This is defined in (2); the concatenation formula is in the formula bar at the bottom, and the function can be reused to translate other dates. Another function specifies the reverse mapping (g), where DateOfBirthField is parsed to obtain the DOB.

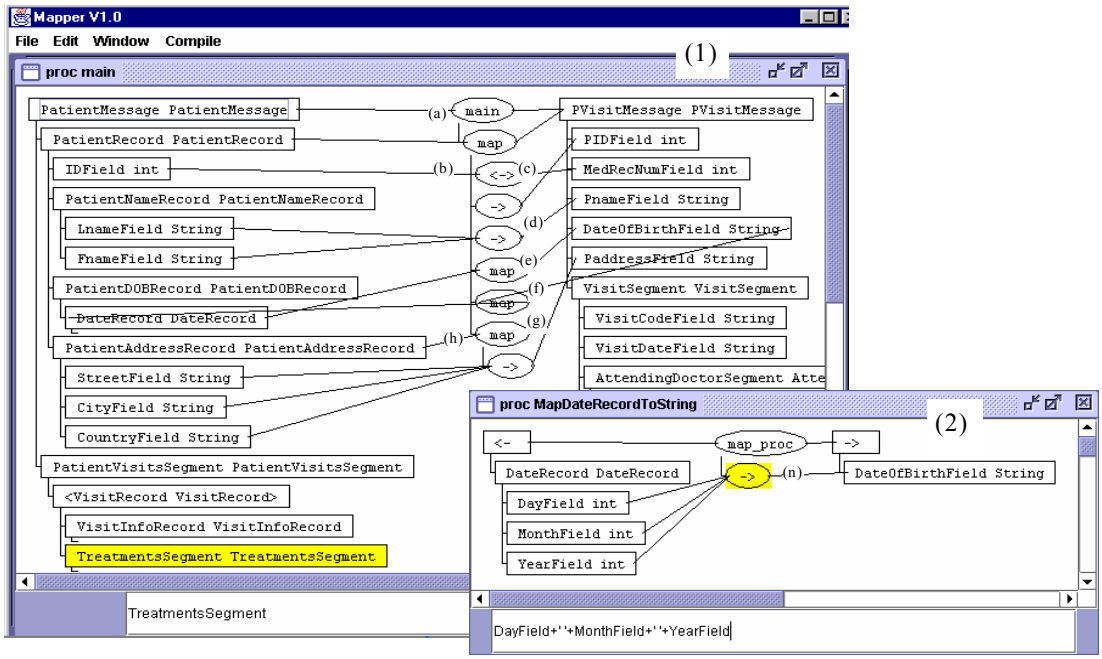


Figure 5. Example mapping specification using RVM

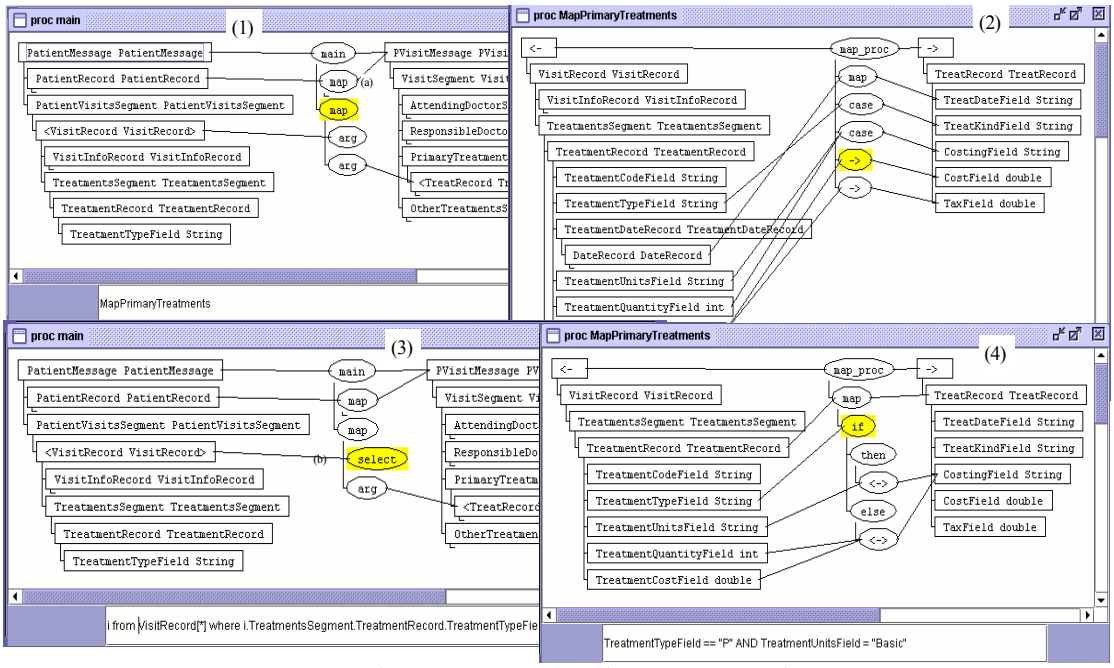


Figure 6. More complex RVM mappings

More complex mappings involve collections and conditional logic. For example, Fig. 6 screen (1) shows how PatientMessage records are translated into multiple PvisitMessage records by a mapping function applied to each VisitRecord in the PatientVisitsSegment, producing a TreatRecord in the target message's PrimaryTreatmentsSegment (a). The function definition is in (2). This uses two lookup tables to map treatment codes. Expressions on function argument nodes specify selection or filtering operations on collection arguments. For example, Fig. 6 screen (3) is an alternative to Fig. 5 screen (1), showing mapping of only "P" treatments in the source message to the PrimaryTreatments segment in the target message. The first input argument (b) now includes a selection filter over source VisitRecords, with the mapping function only applied to source records matching the criteria. Fig. 6 screen (4) shows a conditional mapping node where differing target cost information is calculated depending on a source field value.

When a mapping specification is complete, our visual mapping specification tool generates a textual "mapping language" encoding the full mapping specified by the user. This language is further compiled to a tree-structured byte code, which is interpreted by a mapping engine to automate message transformation. The message translation process is shown in Fig. 7. Mappings can be visualised as they are applied to debug them, as shown on the right of Fig. 7. This reuses the mapping specification views, annotating elements with message instance data, with navigation facilities to step through collections.

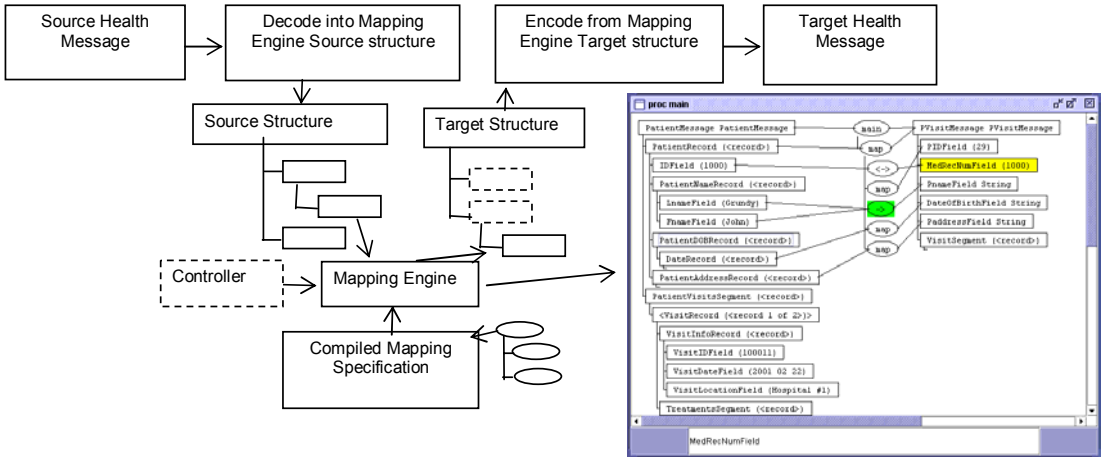


Figure 7. Message mapping engine processing

5. The Form-Based Mapper

Our final data mapping system, the Form-based Mapper (FBM) targets non programmers in the domain of business integration [26]. Most "E-business" requires data transformation from one business's data format to another's [4, 10, 38]. Business analysts are the professionals responsible for understanding and developing business processes; they have knowledge of what one business's information needs are and how to map them onto another's data. However, programmers typically implement business-to-business data exchange mechanisms.

Our aim in developing FBM [26] was to eliminate programmers from this task. Business analysts, typically have minimal programming skills; most are familiar with spreadsheet and possibly simple database programming, but little else. Accordingly we have adopted a metaphor based on business forms. Data structures are represented as business forms, and mappings are specified using a "business form copying" metaphor, with spreadsheet-style data dependency calculations. This mimics what happens when a paper form is sent from one business or unit to another and information from the form is copied into the receiving business' data entry screens.

Fig. 8 shows the mapping specification process used by FBM. Meta-data from enterprise systems (1) is extracted to describe source and target data. A default "business form" is generated for each data model (2). A business analyst can modify these default form layouts to better-conform to physical business forms (hard-copy or computer screen). The analyst specifies correspondences between fields or groups on source and target forms by direct manipulation (3). These correspondences and associated formulae are used to generate data mapping code in the form of XSLT scripts, Java programs or 3rd party data mapping tool code (4).

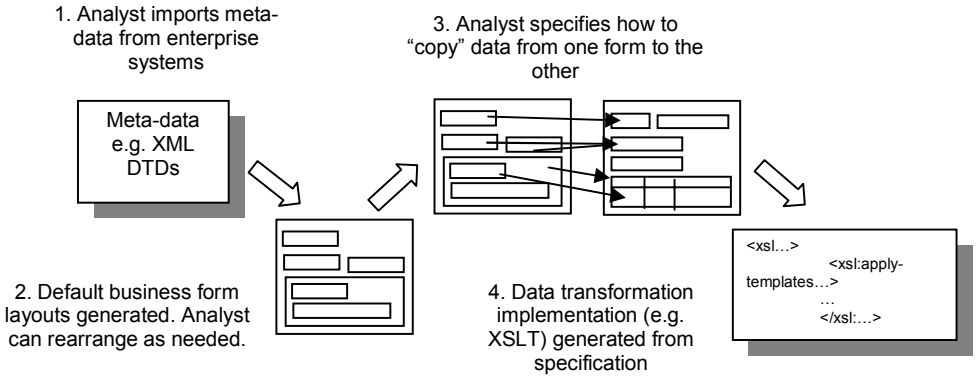


Figure 8. Business form-based data mapping specification and code generation.

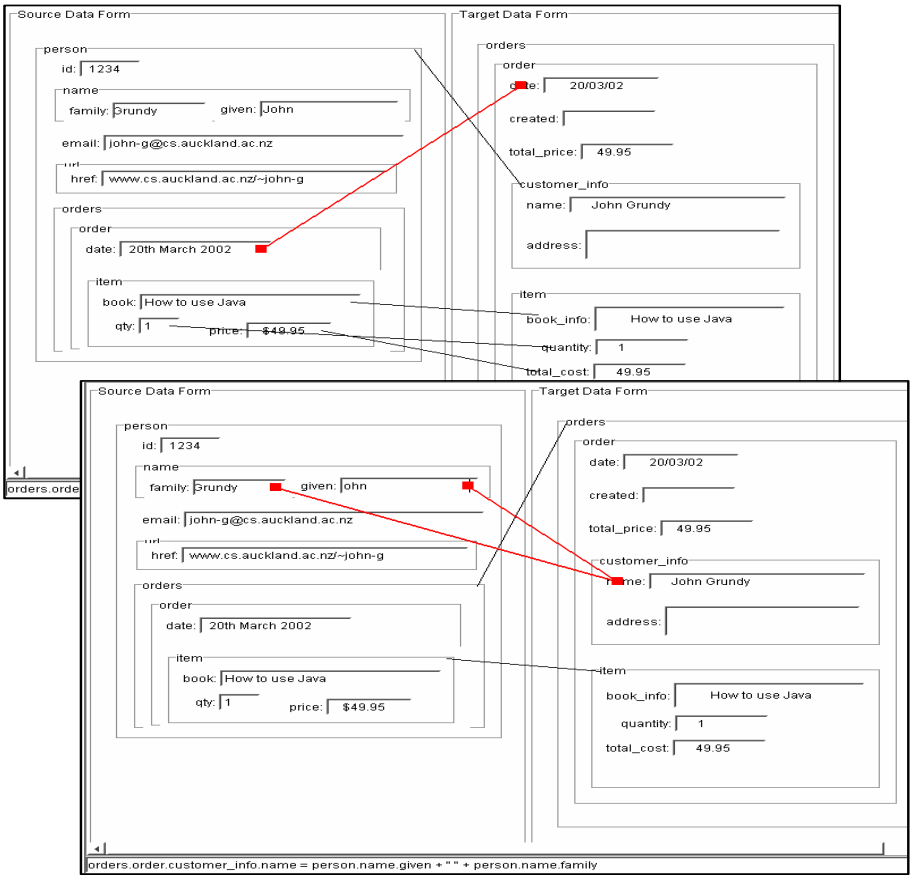


Figure 9. Simple (top) and more complex (bottom) data mapping examples.

Fig. 9 shows FBM in use, with two Order data structures viewed side by side as business forms. Unlike VML and RMV, forms are populated with actual data, to provide a more concrete metaphor. A programming-by-demonstration approach is used to specify source and target data structure correspondences. The analyst drags and drops connections between one or more fields or field groups in the source form and one or more fields or field groups in the target form to specify data transformation mappings. Fig. 9 shows the following different types of mapping:

- 1:1 copies possibly with type or name conversion, for example *qty* to *quantity*, *price* to *total cost* and *date* to *date*, the latter with a type conversion. In these cases a simple drag and drop between fields, possibly with a conversion formula, is sufficient to specify the mapping.
- 1:1 relationships between groups of fields. E.g., the 'person' record has been mapped to the 'customer_info' record.
- m:1 mappings, where several fields may be required to determine the value of one field, e.g. the mapping of 'family' and 'given' fields of the 'name' entity are mapped to the 'name' field in the target form. To specify this, the business analyst selects the two source fields and then drags to the target field, creating a 2 to 1 linkage and enters a formula to specify the calculation (in this case concatenation with a separator). As with RVM, these mappings are usually of a simpler form than their inverse, which typically require some form of parsing.
- n:m mappings, of any combination of fields and structures. For example, the order record in the source form maps onto a whole target form. This implies a single customer with multiple

orders in the source form will generate multiple target "forms" (target order data records). In this example, multiple source order items map onto the same number of target order items. Many domains involve more complex structural mappings (e.g. selection from source group to form target, m:1 or 1:m structural transformations). In FBM analysts specify correspondences between one or more source fields or groups and one or more target fields or groups. They may then qualify the correspondence indicating selection from source groups by indexing or filtering and qualify the target by specifying collection indexing or construction. In our tool the analyst uses library functions (similar to spreadsheet list/array manipulation functions) acting on source field(s) or group(s) to produce result values for the target.

6. Discussion

Here we survey related work and then compare and contrast our visual mapping languages using the well known Cognitive Dimensions framework.

6.1 Related Work

System integration has become a crucial problem in many domains, leading to the development of a wide variety of solutions [4, 38, 45]. The majority of data transformation specification techniques are "programmer-centric" focussing on supporting professional programmers. Many use custom-coded transformation modules or components that take considerable design, implementation and testing by expert programmers [12, 11, 16]. For example, most EDI solutions provide a set of predefined function libraries programmers use to encode and decode messages in particular protocols [11, 41]. Translating between message formats involves reading a message using a protocol's API, writing code to construct a new message, and then generating its transport-level representation using another protocol API. The few high-level EDI message mapping systems developed, such as ETS [1], suffer from using low-level, textual representation of mappings or overly simple visual formalisms. Message-Oriented Middleware (MoM) systems, including MQ Series™ and Tuxedo™, use a similar approach.

Many data integration products address similar problems to our VML mapper. These include Openlink Virtuoso™ [23], the Universal Translation Suite [10], and various virtual database integration [5, 27, 42, 46] and Enterprise Application Integration (EAI) tools, such as Aditel [1], eXcelon [12], Vitria BusinessWare™ [43] and BizTalk™ [11]. Some of these provide translation support for database, message and XML-encoded data using visual representations of mappings, but are limited to simple record structures and are relatively difficult for non-experts to use. Most database-oriented integration products use table-centric specification metaphors. Message-oriented products tend to adopt tree-structured visualisation metaphors, which are neither as declarative as VML nor as analyst-centric as the form-based mapper.

Message integration tools analogous to RVM include Data Transformation Manager, BizTalk [14] and MQ Integrator™ [22]. These provide limited abstract message translation facilities, typically requiring coding of complex translations. Most enable non-expert programmers and some non-programmer end users to specify message data transformations. Many adopt a tree-based mapping metaphor with drag-and-drop between tree nodes, similar to RVM. Most do not however try to capture the structure of the mappings themselves, but rather simply visualise these as links between tree nodes and leaves.

Many recent systems use scripted solutions to support easier evolution of transformations. Many use XML-based transformations [31, 39] with "standardised" DTDs and XML Schema [39, 47]. Many XML translators have been produced, including the W3C standard XML Style Sheet Transformation (XSLT) scripting language, Seeburger's data format and business logic converter [37] and eBizExchange [33]. BizTalk™ [14] and BusinessWare™ [43] also provide visual XML-based message mapping support. Most XML document translation systems use XSLT [7, 47]. These suffer from a lack of expressive power (especially for complex hierarchical mappings) and only partially support visual mapping and XSLT script generation. In addition, these tools use

programmer-centric metaphors, usually tree-based drag-and-drop with transformation expressions. These approaches do not suit many end-users, such as business analysts. XSLT provides a degree of declarative transformation support, but not to the degree of VML.

Several visual language approaches address similar themes to our work. These include SIML-based XML document representation using transformation by graph-grammars [48], algorithm animation using formal mapping specifications between structures [35], and VXT, an XML document transformer [36]. The first two use rule-based specification of mappings between structures, encoding these rules as graph transformation-based specifications or decorated abstract syntax trees. These contrast with our structure-based linking and formulae, which are more readily visualised using source/target linking approaches. VXT provides a document transformation visual language metaphor, which is quite different to our work in that it uses a visual pattern-matching approach rather than explicit structural linkages. Milicev [29] describes a system that has similar expressive power to VML but uses extended UML diagrams including conditional and iterative constructs to specify transformations.

FBM incorporates programming by demonstration (PBD). PBD systems deduce task specifications from user interactions. These are then automated or partially automated [9]. Such systems have been applied to a wide range of problem domains, including MetaMouse [30], Masuishi's report generator [28] Sugiura's Internet Scrapbook [40] and XSLByDemo [20]. A key to the success of PBD systems is the use of real-world metaphors by which users demonstrate actions and computer applications reflect learned operations to users.

6.2 Evaluation

We have undertaken usability trials of our three environments, but for the purposes of this paper it is more useful to evaluate and compare them using the Cognitive Dimensions framework [19]. The significant results of this are as follows, categorised by dimension.

Abstraction Gradient: VML has a relatively small number of visual abstractions: entity icons, inter-class definition blocks, and mapping links. There is some extra complexity in the parts of an inter class definition (invariants, equivalences, initialisers), but this is straightforward for the targeted users. This makes it easy to design and understand a mapping definition at a high level, but, hides the complexity of the formulae used to specify mapping equations and functions. These provide a large number of primitives, including difficult concepts such as bijections, pointer referencing, and escape to code for function specification.

RVM is a medium-level abstraction system. Mapping functions and groups represent potentially complex aggregates of primitive transformations and require knowledge of procedural abstraction. Collection mappings succinctly capture complex, iterative transformations without the need for iteration variables and other complex notation. Use of procedural abstraction was a deliberate choice for RVM, due to end user familiarity with the paradigm, and also as processing of messages requires explicit specification of control flow, for example to avoid multiple passes through a sequential stream of records.

The key abstraction used in FBM is the business form, a concrete visual metaphor comprising primitive form elements (labels, text fields, check boxes, etc) and groups of primitives. These abstractions map onto meta-data elements, though the user can create further abstraction groups if required. Links between fields represent formulae converting source data item(s) and group(s) to target data item(s) and group(s).

In each case, we have chosen a set of abstractions familiar to the targeted end user group. It is interesting to observe that the number of visual abstractions increases inversely to the capability of the end user. This may seem counter intuitive, but is explained by the escape to underlying richer textual formalisms available with RVM and particularly VML.

Closeness of mapping: VML's graphical notation extends from a notation that is familiar to the targeted end users, UML class icons. The inter class definitions and mapping links are intuitive in this context. There is, however, a large step from this visual form to the underlying

textual language. RVM's primitive visual elements represent simple field-level transformations in, explicitly representing source and target dependencies. Mapping functions, iterations and groupings represent high-level, aggregated dependencies of target schema items to source items using abstractions familiar to the target users. FBM uses a business form metaphor. Its visual representation thus maps directly onto business analysts' cognitive model of their problem domain. Allowing user editing of the generated form layout supports even closer mapping as analysts can tailor the layout to be closer to the actual forms they are familiar with. A major difference between the languages is that VML allows mapping of method calls from source to target, a feature omitted from FBM and RVM as a concept thought too difficult for target users.

*Consistenc*y: VML's entity and inter class nodes and RVM's schema and mapping nodes are distinguished by basic shape differences. The left-to-right connectivity of source/mapping/ target nodes is preserved throughout for both. RVM's hierarchical schema and mapping node links use the same layout and visual representation. FBM's source and target form representations both use the same visual form elements. All inter-form element links are rendered consistently, presenting a potential problem as discriminating between simple and complex mappings may be desirable for the business analyst.

Diffuseness/Terseness: The VML visual notation is terse, with only three main elements, and less than a dozen minor elements. The underlying textual notation is, however, far richer, and hence more diffuse. RVM is also quite terse, using a small set of visual icons and connectors and relying on labelling to distinguish different mapping operations and abstractions. In contrast to these more abstract approaches, FBM employs a more verbose visual language that can include elements not directly used in the mapping process e.g. business form layout groups, labels, lines and boxes and images. In contrast, mapping specifications using meta-data renderings such as trees and entity-relationship diagrams seldom include elements not directly used in the meta-data mapping specification. The use of a concrete form-based metaphor in our approach necessitates a less terse notation to support the desired visual metaphor.

Visibility and Juxtaposability: All three have good juxtaposibility with the ability to have multiple views open side by side, displaying different parts of the same mapping specification, and environment navigation options make it straightforward to locate related elements and views. In VML, the formulae for the mappings are specified separately and must be navigated to in separate windows. Use of elision to display the formulae within the icons would have been an alternative. In RVM, poor visibility occurs when reverse-mappings or non-hierarchical message schema references are present. Both result in source/target lines crossing over icons and other connectors, obscuring specifications. Some complex, structural mappings where formulas are used, based on source field values, to specify sub-record groups to map, can't be directly represented visually (but can be expressed in our textual mapping language and encapsulated in a mapping function node).

FBM has explicit inter-form element links providing good visibility, but the links between form elements and element groups to the underlying meta-model is hidden. When the user modifies form layout e.g. by adding or rearranging grouping, this linkage is blurred and not visible in the visual form-based visualisation nor tree-based structure views. Two views are supported in our tool: concrete form-based and tree-based structure visualisations, which are viewed side-by-side. Sub-views are supported using the tree-based view to select a portion of the form for display, but multiple views displayed simultaneously are not currently supported. The FBM views can get very large and some basic elision facilities and multiple views are provided to users. However, currently these result in loss of context of the form elements and mappings.

Viscosity: Changing mapping links in VML has low viscosity, although there is a need to be able to order mappings in the inter class icon to minimise link crossovers. Individual inter class definitions usually only involve a small number of entities, so making space for additional entities is not an issue. The hierarchical layout of RVM provides some difficulty when changes are made. These may necessitate tree manipulation operations that are currently not well

supported in the environment. FBM has a low viscosity to small changes, where the time to make a change is nearly always equivalent to the initial specification time (wiring time and specification of the mapping equation).

Hidden Dependencies: This type of dependency may occur in VML due to the declarative nature of the specification language. The order of evaluation of mappings is dependent upon the specifications which exist and not visible within the described mappings. However, as a declarative language, the fact that evaluation order changes is not significant as long as the mappings are resolvable. What may be significant is that some mappings may become resolvable with changes or additions to the mapping specification and this is not visible from the specification environment. RVM's procedural abstractions create hidden dependencies (separation of function from application), which are minimised through environment navigation aids. FBM's highly concrete metaphor minimises hidden dependencies for the user.

Error-proneness: The wiring approach of VML should reduce the chance of errors in mapping specification, but the hidden nature of the equations comprising a mapping will increase error-proneness by reducing opportunities to validate the equations, functions and procedures which are utilised for the mapping specification. Similarly, the formulae in both RVM and FBM provide significant opportunity for inadvertent errors.

6.3 Implementation techniques

VML was implemented using our MViews framework for constructing multiple view visual editing environments [15], which allowed the environment to be rapidly developed. This framework is implemented using an object oriented extension to Prolog, which was excellent for implementing the declarative features of the underlying mapping implementation.

The prototype RVM environment was implemented using JComposer [16], a meta tool resulting from further development of our MViews framework (including lessons learnt from development of VML). This allowed simple experimentation with notational features and rapid generation of the resulting environment. Additional back-end code generates textual mapping language code. The compiler and Rimu Mapping Engine are written in Java, the later consisting of a threaded interpreter for generated byte code. Orion has developed a commercial version of the mapping tool, Rhapsody [34]. This preserves the metaphor, mapping language and engine architecture, and most visual specification and visualisation techniques. Modifications include phasing out conditional nodes, better navigation, better support for non-hierarchical field references and additional visual annotations.

FBM was developed using Java's Swing GUI API and JAX XML parsing API, and hence required more programming effort than VML or RVM. The tool allows users to import meta-data from XML-encoded data files or DTDs. The meta-data is used to generate a basic form layout with simple heuristics used to generate form elements and element groupings using Java Swing components. We used the Java Swing component event-passing mechanism and transparent panel overlays to intercept user interaction with the generated form components for drag and drop. We generate XSLT-based transformation scripts to implement the data mapping specifications.

7. Conclusions and Future Research

We have presented three domain-specific visual languages addressing the problem of specifying mappings between different schema. The mappings differ in their intended target user group, and hence have adopted different data visualisation metaphors. Each metaphor builds from concepts familiar to the specific target group, and closely related to the business problem they are trying to solve: OO design diagrams for programmers; tree-based schema representations for DBAs; and business forms for business analysts.

The resulting Visual Languages are quite different in nature, yet have surprisingly similar expressive ability for the relatively simple XML-based structures. An upcoming piece of work will attempt to characterise more formally the expressive ability of visual languages for mapping

to provide a deeper understanding of what is forsaken by various choices of formalism. Our three visual languages are unified by the use of a wiring metaphor to express mapping relationships. This appears to us to be a natural way to represent such relationships, but has the problem of "clutter" for large mapping specifications, which must be solved by multiple views or elision.

In each case, the visual language hides much of the complexity of the mapping, in particular the manipulation of collections. This has the advantage that it allows users to concentrate on the high level design of the mappings, i.e., what elements are related to one another, and understand at a gross level the structural manipulations involved in collection mapping. However, it means that complex mappings can be difficult to depict or comprehend from the visual representation, requiring careful examination of formula boxes and textual renderings to understand and debug. We are skeptical, however, that provision of convincing visualisations of complex collection manipulations is achievable for the two less programmer-able target groups. For skilled programmers, an underlying visual formalism, such as a Prograph-style dataflow language [8], may prove an option, but it is questionable whether this has a significant advantage over the textual code for such a target group.

There is an interesting tradeoff with these visual notations between having efficient representations of schema in order to describe a mapping and utilising standard representations of schema that will be immediately comprehensible to users. For example, EXPRESS-G [24], used in engineering domains to represent schema, is poorly suited to the overlay of a mapping notation, yet the introduction of a new representation for schema in this domain purely to support mapping specifications will meet with resistance.

It is clear that the support environment offered alongside the visual notation has a major impact on its usability. The support for data and formula views in FBM is an important aspect of its usability, as is the partial mapping evaluation functionality to view exemplars of transformed data. Multiple overlapping views of partial mappings with consistency maintained across them provide some controls for managing large schema with the notations. Further environment support for the visual languages comprising libraries of common mappings and automated matching of schema portions may also be an important adjunct to the visual notation.

We are interested in exploring other metaphors for the mapping problem. In current work, we are examining approaches to specifying the mapping of visual notations from one to another, together with approaches for specifying office automation (with attendant data mappings).

References

1. Aditel. http://www.aditel.org
2. R. Amor, A Generalised Framework for the Design and Construction of Integrated Design Systems, *Ph.D. thesis,* Department of Computer Science, University of Auckland, Auckland, New Zealand, 1997, 350pp.
3. R. Amor, J. Hosking and W. Mugridge, ICAtect-II: A Framework for the Integration of Building Design Tools, *Automation in Construction, 8(3),* 1999, 277-289.
4. G. Alonso, U. Fiedler, C. Hagen, A. Lazcano, H. Schuldt and N. Weiler, WISE: business to business e-commerce. Proceedings Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises, RIDE-VE'99, Los Alamitos, CA, USA.IEEE CS Press, 1999, 132-139.
5. C. Batini, M. Lenzerini and S.B. Navathe, A Comparative Analysis of Methodologies for Database Schema Integration, *ACM Computing Surveys, 18(4), December*, 1986, 323-364.
6. J. Blackham, P. Grundeman, J.C. Grundy, J.G. Hosking and W.B. Mugridge, Supporting Pervasive Business via Virtual Database Aggregation, In Proceedings of Evolve'2001 – Pervasive Business, Sydney, Australia, March 15-16, DSTC Press, 2001.
7. D. Cheung, S.D. Lee, T. Lee, W. Song and C.J. Tan, Distributed and scalable XML document processing architecture for E-commerce systems. *Proceedings of the Second International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*. IEEE, 2000, 152-157.

8. P.T. Cox, F.R. Giles and T. Pietrzykowski, Prograph: a step towards liberating programming from textual conditioning, in *Proceedings of the 1989 IEEE Workshop on Visual Languages,* IEEE CS Press, 1989, 150-156.
9. A. Cypher, (ed) *Watch what I do: Programming by demonstration*, Cambridge, Mass, MIT Press, 1993.
10. Data Junction Corp, http://www.datajunction.com/.
11. M.A. Emmelhainz, *Electronic Data Interchange: A Total Management Guide*, Van Nostrand Reinhold, 1990.
12. eXcelon Corp, eXcelon B2B Integration Server White Paper, http://www.exceloncorp.com/.
13. A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer and B. Nuseibeh, Inconsistency Handling in Multiperspective Specifications, *IEEE Transactions on Software Engineering*, vol. 2, no. 8, 1994, 569-578.
14. M.A. Goulde, Microsoft's BizTalk Framework adds messaging to XML, *E-Business Strategies & Solutions*, Sept. 1999, 10-14.
15. J.C. Grundy, J.G. Hosking and W.B. Mugridge, Inconsistency Management for Multiple-View Software Development Environments, *IEEE Transactions on Software Engineering*, Vol. 24, No. 11, November, IEEE CS Press, 1998, 960-981.
16. J.C. Grundy, W.B. Mugridge, J.G. Hosking and P. Kendal, Generating EDI Message Translations from Visual Specifications, In *Proceedings of the 2001 IEEE Automated Software Engineering Conference*, San Diego, 26-29 Nov, IEEE CS Press, 2001, 35-42.
17. J.C. Grundy and W. Zou, Building multi-device, adaptive thin-client web user interfaces with Extended Java Server Pages, accepted as a chapter in *Cross-Platform and Multi-device User Interfaces*, Wiley, 2003.
18. J.C. Grundy, J. Bai, J. Blackham, J.G. Hosking and R. Amor, An Architecture for Efficient, Flexible Enterprise System Integration, In Proceedings of the 2003 International Conference on Internet Computing, Las Vegas, June 23-26, CSREA Press, 2003.
19. T.R. Green and M. Petre, Usability analysis of visual programming environments: a 'cognitive dimensions' framework, *Journal of Visual Languages and Computing,* (7), 1996, 131-174.
20. M. Hori, T. Koyanagi, K. Ono, M. Abe, *XSLByDemo*, www.alphaworks.ibm.com/tech/xslbydemo, 2000.
21. IAI, International Alliance for Interoperability http://www.iai-international.org/, 2003.
22. IBM Corp, *MQ Series Integrator*, http://www.ibm.com/.
23. K. Idenhen, Introducing OpenLink Virtuoso: Universal Data Access Without Boundaries, White paper, http://www.openlinksw.com/.
24. ISO/TC184, Part 11: The EXPRESS Language Reference Manual in Industrial automation systems and integration - Product data representation and exchange', *International Standard, ISO-IEC, Geneva, Switzerland, ISO 10303-11,* 1992.
25. W. Kim and J. Seo, Classifying Schematic and Data Heterogeneity in Multidatabase Systems, *IEEE Computer, 24(12), December*, 1991, 12-18.
26. Y. Li, J.C. Grundy, R. Amor and J.G. Hosking, A data mapping specification environment using a concrete business form-based metaphor, In *Proceedings of the 2002 International Conference on Human-Centric Computing*, IEEE CS Press, 2002, 158-167.
27. E.P. Lim and R.H.L. Chiang, The integration of relationship instances from heterogeneous databases. *Decision Support Systems*, vol.29, no.2, Aug, Publisher: Elsevier, Netherlands, 2000, 153-167.
28. T. Masuishi and N. Takahashi, A Reporting Tool Using Programming by Example for Format Designation, in Liebermann H (ed) *Your Wish is my Command*, Morgan Kaufman, 2000.
29. D. Milicev, Automatic model transformations using extended UML object diagrams in modeling environments, *IEEE ToSE, 28(4),* 2002, 413-431.
30. D. Maulsby and I.H. Witten, MetaMouse: an instructable agent for programming by demonstration, in Cypher A (ed) Watch what I do, MIT Press, 1993.
31. J.P. Morgenthal, XML: The New Integration Frontier, *EAI Journal*, Feb., http://www.eaijournal.com/, 2001.
32. B. Nuseibeh, Towards a framework for managing inconsistency between multiple views, in *Proceedings of Viewpoints'96,* ACM Press, San Francisco, 1996, 184-186.
33. OnDisplay Corp, http://www.ondisplay.com/.
34. Orion Systems Ltd, Rhapsody Integration Engine, http://www.orion.co.nz/rhapsody_overview.htm.

35. M. Pereira and P. Henriques, Visualisation/animation of programs based on abstract representations and formal mappings, In Proceedings of the 2001 IEEE Human-Centric Computing Conference, Sept 5-7, Stresa, Italy, IEEE CS Press, 2001, 373-380.
36. E. Pietriga and V. Zumer, VXT: Visual XML Transformor, Proceedings of the 2001 IEEE Human-Centric Computing Conference, Sept 5-7, Stresa, Italy, IEEE CS Press, 2001, 404-405.
37. Seeburger Corp, http://www.seeburger.de/xml/.
38. J.A. Senn, The evolution of business-to-business commerce models: the influence of new information technology models. Proceedings of International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems, Piscataway, NJ, USA, IEEE CS Press, 1999, 153-158.
39. H. Spencer, XML standards for data interchange. *Imaging & Document Solutions*, vol.9, no.9, Sept. 2000, 15-17.
40. A. Sugiura, Web Browsing by Example, in Liebermann H (ed) *Your Wish is my Command*, Morgan Kaufman, 2000.
41. P.M.C. Swatman, P.A. Swatman and D.C. Fowler, A model of EDI integration and strategic business reengineering. *Journal of Strategic Information Systems*, vol.3, no.1, March, 1994, 41-60.
42. E.M.A. Uchoa and R.N. Melo, HEROS: a framework for heterogeneous database systems integration. Database and Expert Systems Applications. 10th International Conference, DEXA'99, Lecture Notes in Computer Science Vol.1677, Berlin, Germany, Springer-Verlag. 1999, 656-667.
43. Vitria Technolgy Inc, http://www.vitria.com/.
44. P. White and J.C. Grundy, Experiences Developing a Collaborative Travel Planning Application with .NET Web Services, In Proceedings of the 2003 International Conference on Web Services, Las Vegas, June 23-26, CSREA Press, 2003.
45. H. Wing, R.M. Colomb and G. Mineau, Using CG formal contexts to support business system interoperation. In *Proceedings of the 6th International Conference on Conceptual Structures*, Berlin, Springer-Verlag, 1998, 431-438.
46. E. Wu, A CORBA-based architecture for integrating distributed and heterogeneous databases, Proceedings Fifth IEEE International Conference on Engineering of Complex Computer Systems, Los Alamitos, CA, USA, IEEE CS Press, 1999, 143-152.
47. XML.org, *XML and XSLT*, http://www.xml.org/.
48. K. Zhang, D.-Q. Zhang and Y. Deng, A visual approach to XML document Design and Transformation, In Proceedings of the 2001 IEEE Human-Centric Computing Conference, Sept 5-7, Stresa, Italy, IEEE CS Press, 2001, 312-319.