Generating EDI Message Translations from Visual Specifications

John Grundy[1], Rick Mugridge[1], John Hosking[1] and Paul Kendall[2]

[1]Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand
{john-g,rick,john}@cs.auckland.ac.nz

[2]Orion Systems Ltd
Mt Eden, Auckland, New Zealand
paul@orion.co.nz

Abstract

Electronic Data Interchange (EDI) systems are used in many domains to support inter-organisational information exchange. These systems require complex message translation, where data must be transformed from one EDI message format into another. We describe a visual language and support environment which greatly simplify the task of the systems integrator by using a domain-specific visual language to express data formats and format translations. Complex message translations are automated by an underlying transformation engine. We describe the motivation for this system, its key visual language and transformation engine features, a prototype environment, and experience translating it into a commercial product.

Keywords: domain-specific visual languages, message translation, visual environments, XML

1. Introduction

Electronic Data Interchange (EDI)-based systems exchange messages that codify organisational information [1, 17, 12]. For example, in Health Informatics a treatment provider's Information System might describe a patient, hospital visit information, patient treatment and treatment costs. It provides a set of EDI messages that are used to add, update and query this information. A health insurer or funding organisation requires this information to record the treatments, costs and reimbursements, but it uses a different set of EDI messages and information structures. In order to support EDI-based information exchange between these systems, the provider must supply the insurer/funder system with its expected message format, or the funder must translate the provider message(s) into its own EDI protocol. Similarly, data sent back to the provider from the funder must be appropriately converted. Often these message schema are very large and translation between them requires complex algorithms and code.

In different application domains a rich range of EDI-based messaging “standards” have arisen. For example in health informatics, well over 100 EDI standards are commonly used by different systems. The more systems that need to be integrated, whether intra- or inter-organisational, the more inter-EDI message mappings are required. Even with the advent of XML-based “standards” for various domains [1, 12, 13], there are many variants of document and message formats that need to be translated between [1]. Message-oriented Middleware (MoM) systems also typically require extensive message mapping to facilitate systems integration. Existing EDI, XML and MoM messaging and translation tools provide little high-level support for message translation, requiring developers to write complex program or scripting code. Most EDI message translations are done by hand-coded applications. Various XSLT-based translation tools exist, but these are have limited expressive power and require considerable effort to use. MoM integration tools also provide limited translation capabilities and limited visual formalisms, requiring scripting and coding.

We describe our work developing a proof-of-concept visual specification language and environment to provide general, high-level, automated EDI and XML message mapping facilities. We outline the motivation and key requirements for this integration support system, and give an overview of our tool-set which meets these. We describe the visual language we developed to facilitate EDI message mapping, with an outline of its underlying textual mapping language and mapping translation engine. We describe the dynamic visualisation support our environment provides to assist users debug their transformation specifications, and report on experience using it for EDI message and XML document mapping support. We conclude by evaluating our prototype visual environment and describe a commercial product developed based on this prototype.

2. Motivation

Many organisations use EDI-based messaging systems in order to exchange information. EDI systems use a message protocol to encode data queries, updates and processing requests in a form suitable for network transmission. These messages are made up of hierarchical record structures (messages, segments, records and fields), encoded into a serialised form for transmission. These messages form an asynchronous communication protocol between multiple systems: one system encodes and sends a message, another receives and decodes it, processes it, and encodes and sends back a response message. Many EDI protocols have been developed over the years for a large range of application domains. In order to facilitate systems integration, it is very often necessary to translate between different message protocols. For example, a health provider may use e.g. the UB92 protocol to encode patient treatment details, and must send this to a funder's system which only accepts 837a protocol-encoded treatment messages. These protocols encode (more or less) the same information, but in quite different ways.

It turns out that coding message translations, or "mappings", from one (or more) "source" messages to one (or more) "target" messages, is relatively complex. Often one has to translate between complex hierarchical structures, apply formulae to several field values and map sets of hierarchical record structures into other sets of hierarchical records. Key requirements for a system supporting mapping of EDI messages include:

- Separation of transport-level format from the hierarchical data schema of the message. Source EDI messages should be abstracted into an object-based representation for processing, and transport-level encodings of target messages generated from object representations when necessary.
- Specification of segment, record and field translations. Field-level translations are relatively straightforward, although complex formulae may be needed to merge multiple field values into one, split a field value into multiple etc. Records and segments may be grouped into collections, and complex collection to collection mappings may be required. Ideally, specifying these mappings should be done using abstract, structural representations rather than programmatic while/for etc loops.
- An integrated system incorporating automated message encoding/decoding and mapping facilities is highly desirable, rather than several separate, inconsistent and possibly incompatible components. Ideally both message format specification and message translation specification do not require direct programming but rather the use of visual tools that are easy to use and generate required message sourcing, sinking and transformations.

Most EDI solutions provide a set of predefined function libraries programmers use to encode and decode messages in particular EDI protocols [20, 21]. When translating between message formats, developers read a message using a protcol's API function, write program code to construct a new message, then generate its transport-level representation using another protocol API function. This means that a programmer is typically required to implement message mappings. Few high-level EDI message mapping systems have been developed, such as ETS, [8, 1]. These all suffer from using only low-level, textual representation of mappings or use of overly simple visual formalisms. Related Message-Oriented Middleware (MoM) systems use a similar approach, including MQ Series™ and Tuxedo™. Some message integration tools have been developed, including MQ Integrator™ [9]. These provide limited abstract message translation facilities, often requiring coding of translations. XML-based systems use "standardised" Document Type Definitions (DTDs) which can be used to form message-based protocols for systems integration [16, 5, 18]. Many XML translators have been produced, including XSLT, Seeburger's data format and business logic converter [15] and eBizExchange [14]. Most XML document translation systems use XSLT (XML Style Sheet-based Translations) [2, 22]. These suffer from a lack of expressive power in XSLT, especially for complex hierarchical collection mappings, and tools that only partially support visual mapping and XSLT script generation. Various Enterprise Application Integration (EAI) and database integration products have been developed to facilitate Business-to-Business electronic commerce. These include Vitria BusinessWare™ [19], BizTalk™ [6] and the Universal Translation Suite [3]. Some of these systems provide translation support for database, message and XML-encoded data using visual representations of mappings. However, these are typically limited to simple record structures, don't support the range of complex mappings inherent in most EDI message translations, and are relatively difficult to use.

3. Overview of the Encore System

In order to provide EDI and XML system developers and integrators with appropriate tool support for message management and translation, Orion Systems Ltd (www.orion.co.nz), a leading health industry EDI solutions provider, has developed the Symphonia™ product suite. The aim is to minimise the requirement for professional programmers to be used in implementing message mapping by providing an appropriate domain specific visual language that business modellers can use. Figure 1 illustrates the key architectural components of this system. Visual tools are used to specify EDI message formats (including XML encoded-messages), message translations (mappings), and message control flow (i.e. message exchange sequences between Symphonia™ components and external EDI- and XML-based systems, including exception handling). Without Symphonia™, a very large program would have to be hand-coded to read source EDI and XML messages, decode and translate these into target messages, and handle message sequencing.

Researchers at the University of Auckland developed a proof-of-concept message mapping system to augment Orion's message control and EDI-to-objects software tools, comprising three key components. A visual mapping tool imports Symphonia™ or XML-encoded message definitions (a), allowing developers to abstractly specify message mappings with a visual notation, generates mapping language programs (b), and dynamically visualises running message mappings (c). A compiler for a textual mapping language takes programs generated by the visual mapping tool and generates a set of binary encoded mapping functions (d). A mapping engine reads objects encoding EDI messages (e), maps source object formats to target object formats based on the mapping function binary code, and generates target EDI and XML message object representations (f).

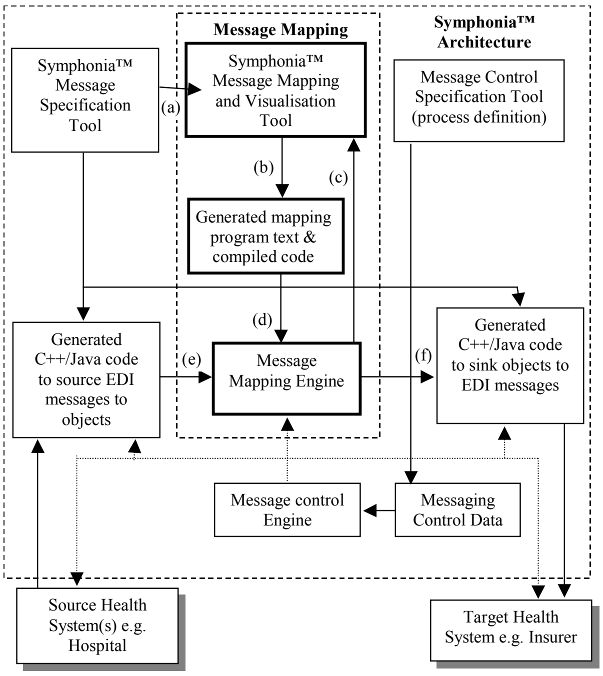
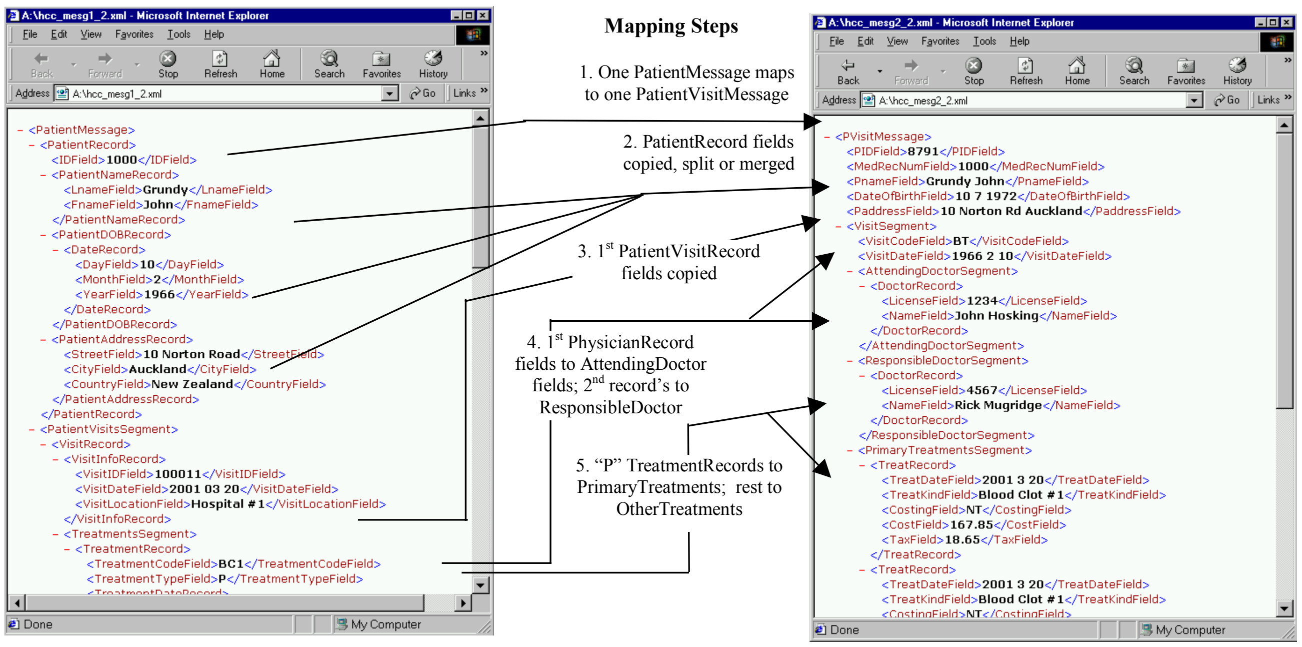

Message Mapping
Symphonia™ Architecture
Symphonia™ Message Specification Tool
(a)
Symphonia™ Message Mapping and Visualisation Tool
Message Control Specification Tool (process definition)
(b)
(c)
Generated mapping program text & compiled code
(d)
Generated C++/Java code to source EDI messages to objects
(e)
Message Mapping Engine
(f)
Generated C++/Java code to sink objects to EDI messages
Message control Engine
Messaging Control Data
Source Health System(s) e.g. Hospital
Target Health System e.g. Insurer

Figure 1. Orion Symphonia™ System Architecture.

The visual mapping tool is described in more detail in the following sections. Its key features include: the ability to visualise EDI and XML message structures; the ability to visually specify complex mappings between message formats; the generation of mapping language program text; the invocation of a compiler and reporting of error messages against visual mapping elements; and the dynamic visualisation of partially-mapped message instances to assist developers debug mapping specifications as they are being run. An evaluation of this visual mapping environment is given, along with a comparison of the proof-of-concept mapping system to the commercial product recently developed by Orion Systems.

4. Message Mapping Process

To illustrate the specification of message mappings with this tool, Figure 2 shows two example messages representing health informatics data (for simplicity, shown in XML format in IE 5.5). The message on the left encodes patient treatment information using a deep hierarchy (Patient->Visits->Treatments). The message on the right encodes (mostly) the same data, but using a flatter-format. To translate the left message into the right we need to apply field-, record-, segment- and record collection-level mappings to transform the structure. To translate the right into the left, we apply mappings to convert the flat structure into the deeper hierarchical one. In addition, several fields and collections of fields must be merged or split. A number of formulae, some dependent on message content, need to be applied. This is reasonably typical of many of the EDI and XML mappings we have used the Symphonia™ system for in the health industry, and is common for many mappings in other domains. Most EDI and XML messages are much larger however, often with hundreds of segments, records and fields.


A:\hcc_mesg1_2.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History
Address A:\hcc_mesg1_2.xml
Go Links
Done
My Computer
Mapping Steps
1. One PatientMessage maps to one PatientVisitMessage
2. PatientRecord fields copied, split or merged
3. 1st PatientVisitRecord fields copied
4. 1st PhysicianRecord fields to AttendingDoctor fields; 2nd record's to ResponsibleDoctor
5. "P" TreatmentRecords to PrimaryTreatments; rest to OtherTreatments
A:\hcc_mesg2_2.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History
Address A:\hcc_mesg2_2.xml
Go Links
Done
My Computer

```
- <PatientMessage>
  - <PatientRecord>
      <IDField>1000</IDField>
    - <PatientNameRecord>
        <LnameField>Grundy</LnameField>
        <FnameField>John</FnameField>
      </PatientNameRecord>
    - <PatientDOBRecord>
      - <DateRecord>
          <DayField>10</DayField>
          <MonthField>2</MonthField>
          <YearField>1966</YearField>
        </DateRecord>
      </PatientDOBRecord>
    - <PatientAddressRecord>
        <StreetField>10 Norton Road</StreetField>
        <CityField>Auckland</CityField>
        <CountryField>New Zealand</CountryField>
      </PatientAddressRecord>
    </PatientRecord>
  - <PatientVisitsSegment>
    - <VisitRecord>
      - <VisitInfoRecord>
          <VisitIDField>100011</VisitIDField>
          <VisitDateField>2001 03 20</VisitDateField>
          <VisitLocationField>Hospital #1</VisitLocationField>
        </VisitInfoRecord>
      - <TreatmentsSegment>
        - <TreatmentRecord>
            <TreatmentCodeField>BC1</TreatmentCodeField>
            <TreatmentTypeField>P</TreatmentTypeField>
```

```
- <PVisitMessage>
    <PIDField>8791</PIDField>
    <MedRecNumField>1000</MedRecNumField>
    <PnameField>Grundy John</PnameField>
    <DateOfBirthField>10 7 1972</DateOfBirthField>
    <PaddressField>10 Norton Rd Auckland</PaddressField>
  - <VisitSegment>
      <VisitCodeField>BT</VisitCodeField>
      <VisitDateField>1966 2 10</VisitDateField>
    - <AttendingDoctorSegment>
      - <DoctorRecord>
          <LicenseField>1234</LicenseField>
          <NameField>John Hosking</NameField>
        </DoctorRecord>
      </AttendingDoctorSegment>
    - <ResponsibleDoctorSegment>
      - <DoctorRecord>
          <LicenseField>4567</LicenseField>
          <NameField>Rick Mugridge</NameField>
        </DoctorRecord>
      </ResponsibleDoctorSegment>
    - <PrimaryTreatmentsSegment>
      - <TreatRecord>
          <TreatDateField>2001 3 20</TreatDateField>
          <TreatKindField>Blood Clot #1</TreatKindField>
          <CostingField>NT</CostingField>
          <CostField>167.85</CostField>
          <TaxField>18.65</TaxField>
        </TreatRecord>
      - <TreatRecord>
          <TreatDateField>2001 3 20</TreatDateField>
          <TreatKindField>Blood Clot #1</TreatKindField>
```

Figure 2. Examples of EDI formats to be mapped (shown in XML).

5. Message Mapping Specification

Our visual mapping tool provides a domain-specific visual language for representing message mappings. This includes source and target message structure representation and mappings between source and target segments, records, fields and collections. The mapping tool focuses on structure mapping semantics, not transport-level EDI or XML parsing and unparsing (which is handled by the Symphonia™ Message Specification tool-generated C++ or Java transport-level decoding/encoding code).

The visual mapping tool is used to represent message structures in a predominantly hierarchical form (as EDI and most XML messages are strongly oriented towards hierarchical structure). Developers import message schema definitions into the tool from Symphonia™ Message Specification tool-generated EDI protocol definitions or from XML documents, XML DTDs or XML Schema definitions. The source message schema is placed on the left, the target on the right, using rectangles ("nodes") to represent each message, segment, record and field item in the message "schema" (see Figure 3). Child nodes are linked to their owning parents. Note, however, that many mappings may be "run backwards" and our compiler allows mapping specifications to be generated for translations left-to-right or right-to-left. Any number of views of parts of a mapping specification are supported by the tool. In EDI messages, some parts of a message repeat and some are optional, and this is indicated in the visual tool (see Figure 3). Schema nodes can be shown or hidden by developers as required, in order to manage the complexity of mapping specifications.

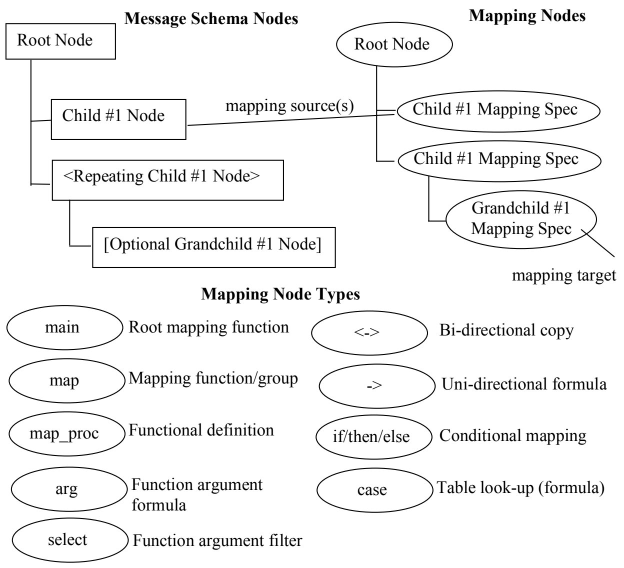
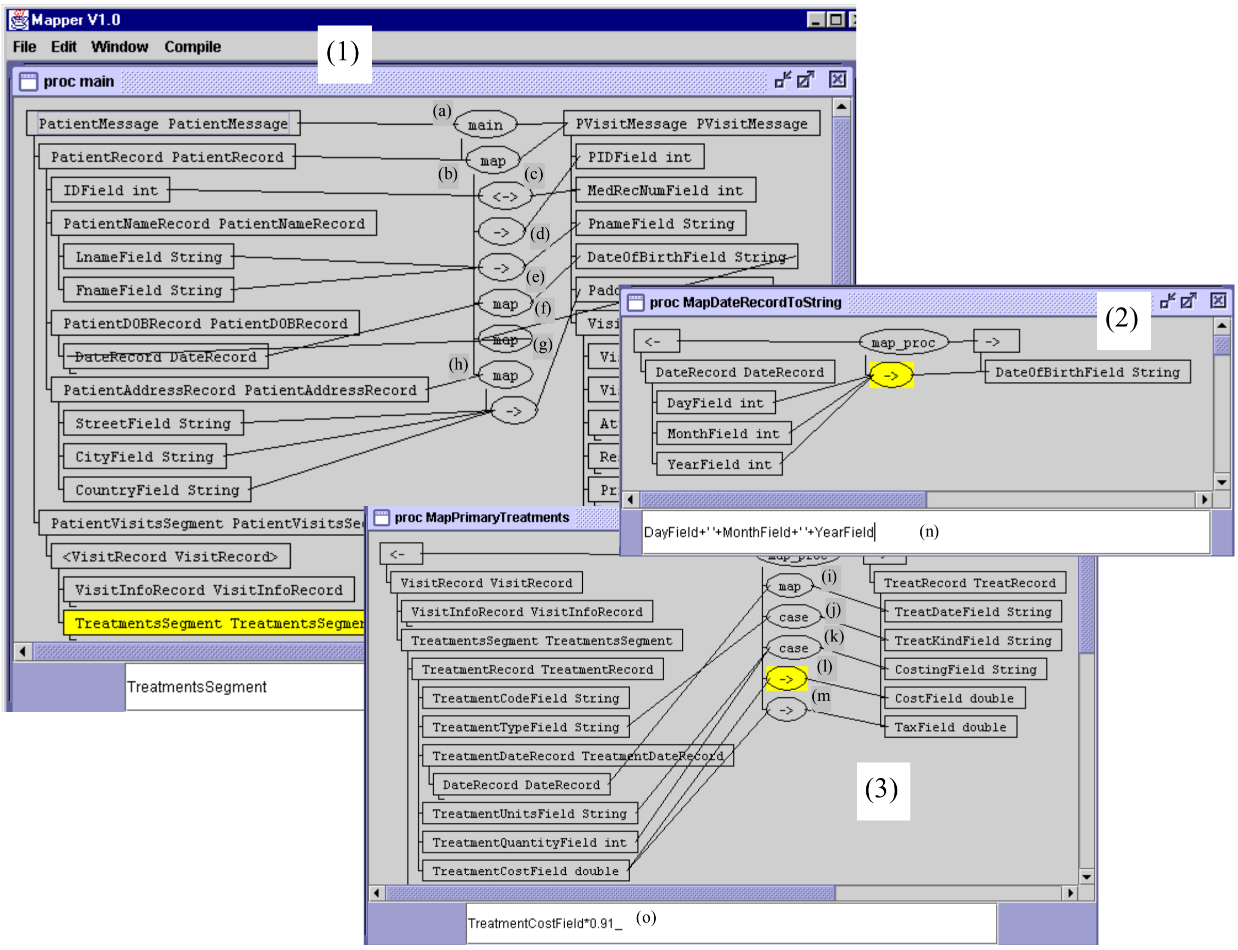


Figure 3. Mapper Visual Language.

We use a simple visual language to describe the message mappings. Oval "mapping nodes" are arranged hierarchically in the centre of a mapping view. Each node typically has a source message schema node and target schema node. A mapping node thus specifies a translation of source information into target information. For example, indicating the presence of a segment in a source message requires a corresponding segment in a target message; a source field should be copied into a target field; a formula be applied to one or more source fields to obtain a target field; or a group of source records be translated into a group of target records (possibly with a different hierarchial organisation of sub-records and their fields). When the mapping engine applies mapping transformations it runs these hierarchically and in top-to-bottom order i.e. top-level "message map" first, which creates the target message body; and then subsequent mappings. If a source is a collection (repeating set of records), the mapping either selects one item from the collection and transforms, or multiple items from the collection which it transforms one-by-one. Note that some mapping nodes may have multiple source nodes where a formula is applied to these to calculate an output, and some no source node if a default value is put into the target message field.

Figure 4 (1) shows two message schema (those from Figure 2) and some simple mapping specifications. The “main” mapping node (a) groups all mappings from one EDI message to another. The first child mapping node (b) below groups the mappings specifying how to copy PatientMessage patient information to a PVisitMessage's patient fields. The first node of this (c) specifies that MedRecNumField is copied from IDField, and is a bi-directional mapping (<->) i.e. can be applied the same when mapping in either direction. The PIDField value is defaulted (auto-generated) by an external function call (d). The PnameField value is a merge of the PatientNameRecord’s LnameField and FnameField values (e). The DateRecord fields are merged into one DateOfBirthField value by a local mapping function call (f). Another reusable mapping function call says what to do when translating in the reverse direction (g), in this case the DateOfBirthField will have to be parsed and split to obtain the separated DOB record's field values. Figure 4 (2) shows the date mapping function definition we can reuse instead of redefining this common mapping functionality. Address record fields are merged (h). Figure 4 (3) shows some other basic mappings used when translating treatment details. The TreatmentDate transformation (i) is specified using the mapping function in Figure 4 (2), the TreatKind and TreatCosting are looked up in tables using source record values (j, k), and TreatRecord cost fields are calculated from a single source value (l, m).

Formulae are written in our textual mapping language, as shown the bottom window text areas (n, o). This includes local variables, conditional and iteration statements and so on. Our mapping system allows developers to code complex, reusable mapping functions both with the visual editor or with this textual scripting language. They can also, if necessary, call external functions written in Java or C++ using mapping nodes.

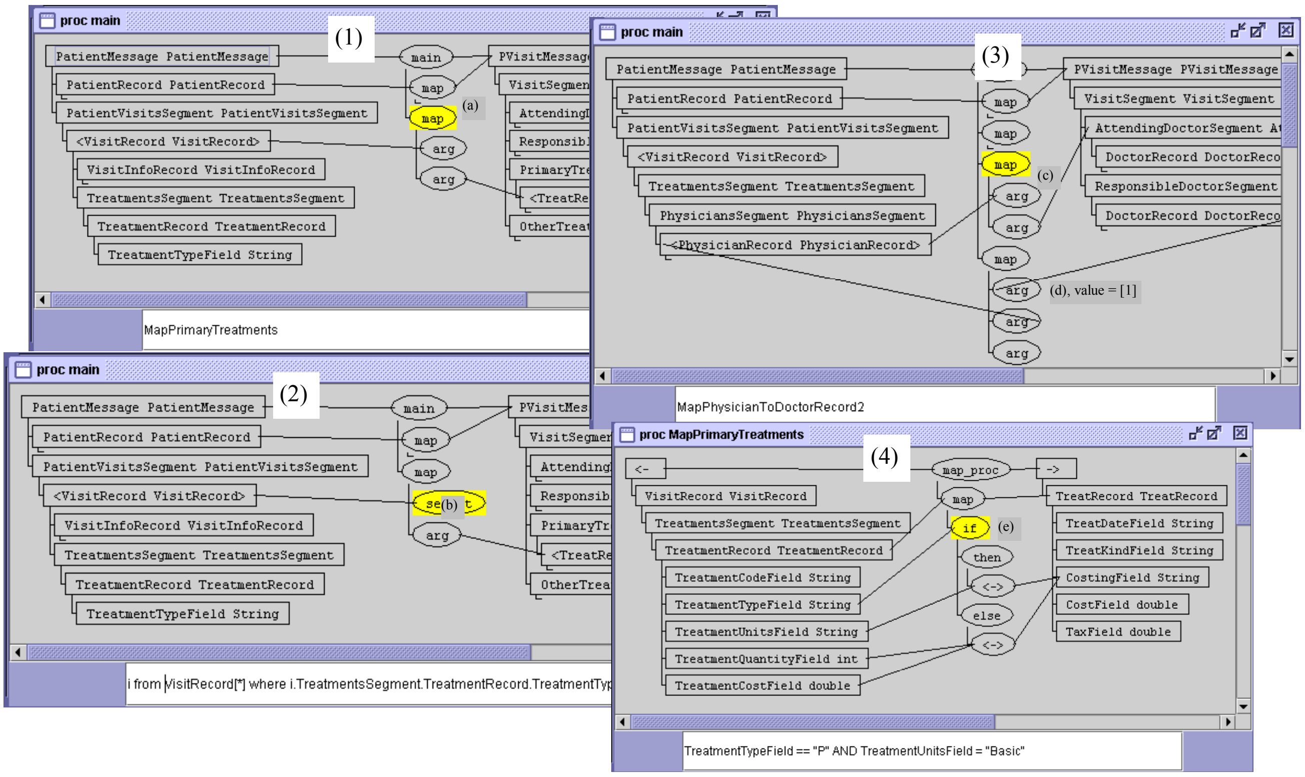


Mapper V1.0
File Edit Window Compile
(1)
proc main
PatientMessage PatientMessage
PatientRecord PatientRecord
IDField int
PatientNameRecord PatientNameRecord
LnameField String
FnameField String
PatientDOBRecord PatientDOBRecord
DateRecord DateRecord
PatientAddressRecord PatientAddressRecord
StreetField String
CityField String
CountryField String
PatientVisitsSegment PatientVisitsSe
<VisitRecord VisitRecord>
VisitInfoRecord VisitInfoRecord
TreatmentsSegment TreatmentsSegmen
main
map
<->
->
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
PVisitMessage PVisitMessage
PIDField int
MedRecNumField int
PnameField String
DateOfBirthField String
TreatmentsSegment
(2)
proc MapDateRecordToString
<-
map_proc
DateRecord DateRecord
DayField int
MonthField int
YearField int
DateOfBirthField String
DayField+'+MonthField+'+YearField
(n)
proc MapPrimaryTreatments
VisitRecord VisitRecord
VisitInfoRecord VisitInfoRecord
TreatmentsSegment TreatmentsSegment
TreatmentRecord TreatmentRecord
TreatmentCodeField String
TreatmentTypeField String
TreatmentDateRecord TreatmentDateRecord
DateRecord DateRecord
TreatmentUnitsField String
TreatmentQuantityField int
TreatmentCostField double
case
(i)
(j)
(k)
(l)
(m
TreatRecord TreatRecord
TreatDateField String
TreatKindField String
CostingField String
CostField double
TaxField double
(3)
TreatmentCostField*0.91_
(o)

Figure 4. Some simple message mapping examples expressed in our visual mapping specification prototype tool.

More complex message transformations involve transformations of groups of records into similar or different structures and the use of conditional logic during transformations, sometimes doing content-sensitive message transformations. For example, Figure 5 (1) shows how all PatientMessage treatment records can be translated into multiple PvisitMessage primary treatment records by applying a mapping function to each VisitRecord in the PatientVisitsSegment, producing a corresponding TreatRecord in the target message's PrimaryTreatmentsSegment (a). The definition of the mapping function called by this mapping is the one shown in Figure 4 (3) above. Sometimes we want to selectively map information. Argument nodes are used to specify such restrictions on arguments passed to mapping functions. For example, Figure 5 (2) is an alternative to Figure 5 (1), showing the mapping of only treatments marked "P" in the source message to the PrimaryTreatments segment in the target message. The first mapping function input argument (b) now includes a selection filter over the source VisitRecords, with the mapping function only being applied to each source record matching the selection criteria. The selection argument (with its formula shown in the message bar) specifies only TreatmentRecords with TreatmentTypeField == "P" are mapped to PrimaryTreatmentsSegment TreatRecords. Other TreatmentTypeField values are mapped to OtherTreatmentsSegment TreatRecords.

Sometimes we want to map one element of a collection into a single target element (or vice-versa). Figure 5 (3) shows first the mapping of the first Physician record in the source message into the AttendingDoctor record in the right-hand side message. The top argument node (c) has value "[0]", indexing the first source PhysicianRecord. The reverse mapping is of the ResponsibleDoctor record to the second Physician record in the left-hand side message. The second arg node (d) has value "[1]", creating a second target PhysicianRecord when this mapping is done. Sometimes we selectively apply mapping processing. For example, Figure 5 (4) shows different target costing information being calculated depending on a source field value, using if/then/else constructs in the mapping specification (e).



(1)
proc main
PatientMessage PatientMessage
PatientRecord PatientRecord
PatientVisitsSegment PatientVisitsSegment
<VisitRecord VisitRecord>
VisitInfoRecord VisitInfoRecord
TreatmentsSegment TreatmentsSegment
TreatmentRecord TreatmentRecord
TreatmentTypeField String
main
map
(a)
arg
MapPrimaryTreatments
(2)
se(b) t
i from VisitRecord[*] where i.TreatmentsSegment.TreatmentRecord.TreatmentTyp
(3)
PVisitMessage PVisitMessage
VisitSegment VisitSegment
AttendingDoctorSegment
DoctorRecord DoctorRecord
ResponsibleDoctorSegment
PhysiciansSegment PhysiciansSegment
<PhysicianRecord PhysicianRecord>
(c)
(d), value = [1]
MapPhysicianToDoctorRecord2
(4)
proc MapPrimaryTreatments
map_proc
VisitRecord VisitRecord
TreatRecord TreatRecord
if
(e)
then
else
TreatmentCodeField String
TreatmentTypeField String
TreatmentUnitsField String
TreatmentQuantityField int
TreatmentCostField double
TreatDateField String
TreatKindField String
CostingField String
CostField double
TaxField double
TreatmentTypeField == "P" AND TreatmentUnitsField = "Basic"

Figure 5. Complex mapping examples.

6. Mapping Language and Engine

When a mapping specification is complete, our visual mapping specification tool generates a textual "mapping language" encoding the full EDI/XML message mapping specified by the user. An example is shown in Figure 6. This language includes message typing, conventional programming constructs and a number of record collection iteration, selection, replication and creation constructs. It also allows external functions (in Java or C++) to be invoked for type conversions, field parsing etc. This language is automatically compiled to a byte code for our message mapping engine.

```
type PatientMessage = struct {
  PatientRecord PatientRecord;
  PatientVisitsSegment PatientVisitsSegment;
};
type PatientRecord = struct {
  int IDField;
  PatientNameRecord PatientNameRecord;
  PatientDOBRecord PatientDOBRecord;
  PatientAddressRecord PatientAddressRecord;
};
…
map main(<- PatientMessage PatientMessage,
        -> PVisitMessage PVisitMessage)
{
  PatientMessage.PatientRecord.IDField <->
        PVisitMessage.MedRecNumField;
  ExternalGeneratePatientID(,PVisitMessage.PIDField);
  Concat(PatientMessage.PatientRecord.
        PatientNameRecord.LnameField, ' ', …
  );
  DOBRecordToDateOfBirth(PatientMessage.PatientRecord.
        PatientDOBRecord,PVisitMessage.DateOfBirthField);
  MapTreatmentRecordToTreatRecord(select(I from
        in.PatientVisitsSegment.VisitRecord.TreatmentsSegment.
             TreatmentRecord[*] where I.TreatmentSegment … ));
…
}
```

Figure 6. Some generated mapping language code.

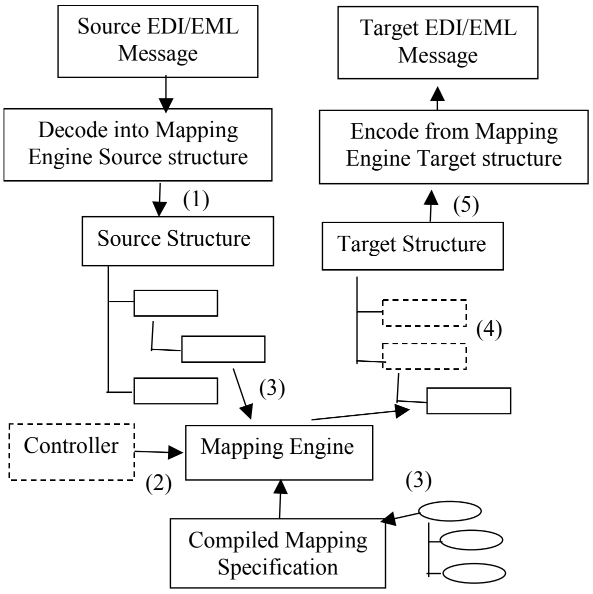


Figure 7. Message mapping engine processing.

When deployed, the mapping engine uses the compiled byte code to automate EDI and XML message transformation. Figure 7 illustrates the basic process of message transformation. A source message is read by the mapper (supplied by the generated transport-level decoding classes) into a source message data structure (1). The mapping specification for this message to a target message is requested by the Symphonia™ message controller (2). The mapping engine traverses the compiled mappjng specification hierarchically, running each mapping function and then its sub-mapping functions in turn (3). The source message records and fields can be read in any order, and the target message can similarly be constructed in any order, its values put into a target message data structure. Place-holders are used by the mapping engine for not-yet transformed target message parts if necessary (4). Once mapping is complete, a transport-level target message is constructed from the mapping engine target data structure, using the generated message encoding classes.

7. Mapping Visualisation

Our visual mapping tool can be used to visualise in-progress message mappings and to step through mappings as they are applied to debug them. Actual message data is shown in the visual views to aid this process. Figure 8 shows the visual tool in use during message mapping debugging. The mapping is shown part-way through, after compiling the specification, selecting "Step Map", opening the source XML-encoded EDI message and stepping through the first three mappings. Some target fields have values e.g. patient information (a) while others have not been assigned values yet (b). Groups of records can be displayed (c), one group item at a time by the use of pop-up menus associated with schema nodes (d).

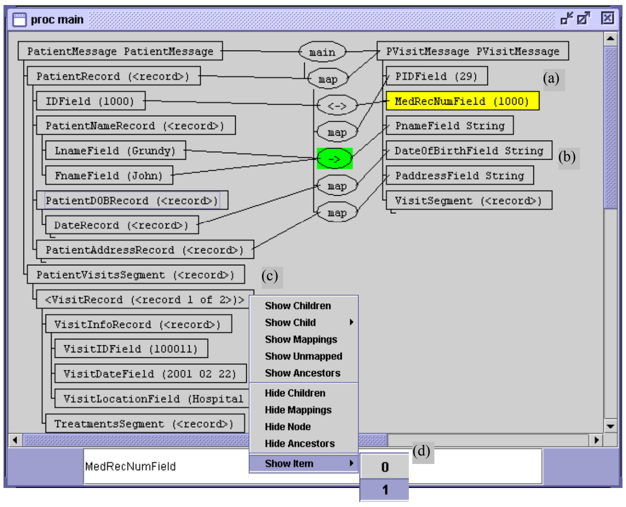


Figure 8. Visualising partially mapped data.

8. Experience

Translation of EDI (and other) message formats is complex [4, 5, 21]. Typically large, complex programs are written in languages like Java and C++ to do this. These are difficult to understand and debug, but their chief weakness is that they take a long time to develop and are hard to maintain. Existing message and document translation tools do help, but most are limited to fairly basic data and structural translations. Most use textual programming or scripting.

Our mapping environment provides a high-level, expressive visual language with which EDI system integrators can readily specify complex mappings, particularly conditional and structural mappings. A visual formalism is used for mapping structure and a textual language for formulaic information. These specifications are compiled and run by a mapping engine, with visualisation and step-through of mapping supported in a tightly integrated way, visualisations using the same mapping visual language. The expressive power of this system is demonstrated by noting that a hand coded UB92 to 837 health EDI message mapping program took over 3 months to build, whereas a visual mapper-specified and implemented equivalent took less than a week. We implemented our prototype using Java, which proved very successful, being able to leverage Java's XML and Symphonia-generated EDI message parsing. We have run many performance tests with a range of message formats, sizes and number. Performance of our mapping engine is excellent, with around 30,000 moderately complex EDI messages being able to be translated per minute.

We carried out a cognitive dimensions framework-based evaluation of our visual mapping tool to assess its visual language's adequacy for EDI and XML message translation developers [7]. This evaluation framework characterises features of visual languages using several dimensions, which helps indicate how users will find the language in use. The results are summarised below.

Abstraction Gradient. Our visual mapping language is a medium-level abstraction system. Mapping functions and groups represent potentially complex aggregations of primitive mapping transformations. Collection mappings succinctly capture complex, iterative transformations.

Closeness of mapping. Our primitive visual elements represent simple field-level transformations in our textual mapping language, explicitly representing source and target dependencies. Mapping functions, iterations and groupings represent high-level, aggregated dependencies of target schema items to source items.

Consistency. Schema and mapping nodes are distinguished by basic shape differences. Hierarchical schema and mapping node links use the same layout and visual representation. The left-to-right connectivity of source/mapping/target nodes is preserved throughout.

Diffuseness/Terseness. Our prototype visual mapping language is quite terse, using a small set of visual icons and connectors and relying on labelling to distinguish different mapping operations and abstractions.

Hard Mental Operations. Understanding mapping specifications requires understanding the mapping engine semantics: hierarchical and iterative mapping traversal.

Role-Expressivenes. Model/view separation and multiple view support makes it possible to create modularised models, with each view of the model displaying a single related group of entities from the model. The view can be given a relevant name indicating the role the group of entities has in the model.

Visibility and Juxtaposability. Our mapping tool has good juxtaposibility with the ability to have multiple views open side by side, displaying different parts of the same mapping specification. Poor visibility occurs when revesre-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 visual mapping function nodes).

The success of our prototype has lead to Orion Systems Ltd developing a commercial version of the tool, recently released in a Beta version. This preserves the visual tool metaphor, mapping language and engine architecture, and much of the visual specification and dynamic visualisation techniques. Some modifications include the phasing out of conditional nodes in the visual mapping specifications and use of textual if/case statements; the introduction of various navigation aids, better support for non-heirarchical field references and additional visual annotations. We are investigating the extension of this system to support database sourcing and sinking of information i.e. in addition to EDI messages, allowing developers to specify mappings to and from database tables. While we have applied both the proof-of-concept and commercial mapping tools to health informatics, they are not restricted to this domain. We are investigating the application of the Symphonia™ system to a wide range of Business-to-Business electronic commerce domains.

9. Summary

Systems integration is a challenging task, particularly in application domains where a wide variety of complex "standards" for representing inter-organisational information exchange have grown over time. We have developed a message translation system for EDI-based and XML-based applications, which has been applied to and shown its worth in the health informatics domain. Key contributions are its intuitive visual representation of inter-schema message mappings, full integration with a powerful textual mapping language, an efficient mapping engine and ability to dynamically visualise, using the visual mapping views, running message mappings. A commercial product developed from our proof-of-concept system has been developed and wide interest in this product is being shown in the health systems integration industry.

References

1. Aditel Corp. ETS for Windows™, www.aditel.be.
2. Cheung, D., Lee, S.D., Lee, T., Song, W., Tan, C.J. Distributed and scalable XML document processing architecture for E-commerce systems. In *Proceedings of the Second International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*. IEEE CS Press, 2000, pp.152-157.
3. Data Junction Corp, Universal Translation Suite™ General Information, www.datajunction.com.
4. Emmelhainz M.A. *Electronic Data Interchange: A Total Management Guide*, Van Nostrand Rein-hold; New York; 1990.
5. Estes, D. Disciplined XML, *EAI Journal*, Jan. 2001, www.eaijournal.com.
6. Goulde, M.A. Microsoft's BizTalk Framework adds messaging to XML. *E-Business Strategies & Solutions*, Sept. 1999, pp.10-14.
7. Green, T.R.G and Petre, M, Usability analysis of visual programming environments: a 'cognitive dimensions' framework, Journal of Visual Languages and Computing 1996 (7), pp.131-174.
8. Huemer, C. and Tjoa, A.M. Meta Messages in Electronic Data Interchange (EDI), In *Proceedings of the Third IEEE meta-data conference*, April 1999.
9. IBM Corp, MQ Series Integrator, www.ibm.com.
10. Liou, D.M., Huang, E.W., Chen, T.T. and Hsiao, S.H. Design and implementation of a Web-based HL7 validation system. *Proceedings 2000 IEEE EMBS International Conference on Information Technology Applications in Biomedicine*, IEEE CS Press, 2000, pp.347-352.
11. 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.
12. McLure ML, Moynihan JJ. Organizing for EDI (healthcare industry). *Healthcare Financial Management*, vol.49, no.1, Jan. 1995, pp.90-93.
13. Morgenthal, J.P. XML: The New Integration Frontier, *EAI Journal*, Feb. 2001, www.eaijournal.com.
14. OnDisplay Corp, CenterStage eBizXchange, www.ondisplay.com.
15. Seeburger Corp, SEEBURGER data format and business logic converter, www.seeburger.de/xml/.
16. Spencer H. XML standards for data interchange. *Imaging & Document Solutions*, vol.9, no.9, Sept. 2000, pp.15-17.
17. Swatman, P.M.C., Swatman, P.A., Fowler, D.C. A model of EDI integration and strategic business reengineering. *Journal of Strategic Information Systems*, vol.3, no.1, March, 1994, pp.41-60.
18. Sokolowski, R., Expressing health care objects in XML, In *Proceedings of the 1999 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, IEEE CS Press, 1999 pp, 341 –342.
19. Vitria Technolgy Inc, Vitria BusinessWare White Paper, www.vitria.com.
20. Wallin, G. A new look at EDI healthcare. *Health Management Technology*, vol.20, no.5, June 1999.
21. Wing, H., Colomb, R.M., Mineau, G. Using CG formal contexts to support business system interoperation. 6th International Conference on Conceptual Structures, Berlin, Germany, 1998, Springer-Verlag, pp.431-8.
22. XML.org, XML and XSLT, www.xml.org.