Automated Testing of Healthcare Document Transformations in the PICASSO Interoperability Platform

Massimo Pascale, Marcello Roselli, Umberto Rugani
Codices s.r.l.
via G. Malasoma - 56121 Pisa, Italy
{m.pascale, m.roselli, u.rugani}@codices.com

Cesare Bartolini, Antonia Bertolino, Francesca Lonetti, Eda Marchetti
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche
via G. Moruzzi, 1 - 56124 Pisa, Italy
{cesare.bartolini, antonia.bertolino, francesca.lonetti, eda.marchetti}@isti.cnr.it

Andrea Polini
Dipartimento di Matematica ed Informatica, Università di Camerino
via Madonna delle Carceri, 9 - 62032 Camerino (MC), Italy
{andrea.polini}@unicam.it

Abstract

In every application domain, achieving interoperability among heterogenous information systems is a crucial challenge and alliances are formed to standardize data-exchange formats. In the healthcare sector, HL7-V3 provides the current international reference models for clinical and administrative documents. Codices, an Italian company, provides the PICASSO platform that uses HL7-V3 as the pivot format to fast achieve a highly integrated degree of interoperability among health-related applications. Given the XML structure of HL7-V3, PICASSO can exploit the XSLT technology to flexibly transform documents. However, Codices spends a large part of the PICASSO deployment workflow for manually validating the required XSL stylesheets. In this paper, we describe a pilot experience in test automation, based on the TAXI tool that applies systematic black-box techniques to generate a set of XML instances from a schema. Observed benefits to Codices development process are reported and discussed.

1 Introduction

Information systems are turning into the core asset of enterprises and organizations in every application domain, from Government to Healthcare, from Commerce to Education, from Banking to Telecommunications. "Interoperability" is today the enabling password and the crucial challenge for companies competing in the global market. The term in broad sense refers to *the ability of diverse systems and organizations to work together* [12], and may embrace different flavours, such as legal, political, social or technical [13]. Concerning interoperability among information systems, it entails syntactic and semantic levels. The former refers to the format of data, i.e., information systems belonging to different organizations can successfully exchange information; the latter is associated with their meaning, i.e., the interacting parties also agree on the interpretation of the exchanged data.

Nowadays syntactic integration is a reality. Software developers and service providers are aware that delivering data in an open specified format to facilitate their sharing with other companies is a mutual benefit. So in many domains, alliances have been established to foster data interoperability by jointly defining and maintaining a common syntax. The introduction of the XML language [1] and its fast diffusion has twisted such initiatives. XML provides the universally adopted notation for the standardization of content formats across domains. On the other hand, full semantic interoperability among communicating systems is still actively sought as the "holy Grail" [16].

Healthcare is a domain in which the need for automating information systems integration is particularly vital, for several reasons. For example, the same patient's historical data can be spread among different, geographically far, hospitals; within the same hospital or healthcare organization, systems dealing with disparate kinds of data need to interoperate, from the management of medical visit reservations to personnel accounting, from patient's clinical records to billing systems; and the exchanged information oftentimes carries on critical aspects.

Founded in 1987, *Health Level Seven* (HL7) [9] is commonly acclaimed as the reference organization for the development of international healthcare standards relative to clinical and administrative data. The current working version of HL7 standard is Version 3 (HL7-V3), which provides the Reference Information Model (RIM) for development. It also addresses semantic interoperability by explicitly defining the communication protocols, the messages to exchange, and the meaning of each information field.

HL7 is an international organization headquartered in the United States, but spread all over the world with affiliated national consortia, now almost thirty, that specialize the international standards to the local needs. In Italy, HL7 Italia [10] has been founded in 2003 as a mixed consortium of hospitals, large and small industries, and research centres. Codices is a SME[1] which is part of HL7 Italia.

Codices is actively involved in a coordinated effort among local and national government offices, to introduce HL7-V3 standard in the healthcare sector throughout the Tuscany region. In the past, the lack of a central organization in establishing the cooperation between systems managing clinical data has brought to a proliferation of heterogenous applications, making the task of introducing a regional standard for interoperability quite difficult.

In this context the experience presented in this paper takes place. Codices has developed an advanced platform to interconnect applications following differing standards, by using HL7-V3 as a pivot protocol. The platform is called PICASSO[2] (platform for interoperability and application cooperation in healthcare organizations and hospitals) and is the first Italian HL7-V3 platform. PICASSO relies on the XML technology for data transformation, on advanced algorithms for performance optimization and on the use of patterns design (like Message Broker and Remote Procedure Invocation)[11] for message exchange and component integration. It has already successfully operated in several projects. However, Codices developers were not satisfied with the validation stage at any new installation of a PICASSO enabled communication channel, which was performed manually.

In this paper we present a joint experience between Codices and a CNR research group in introducing automated tool support for the validation of transformations performed within PICASSO. The proof-of-concept case study reported here showed promising results and has been quite inspiring for further integration approaches for automation.

The paper is structured as follows: in the next section, we provide further details about PICASSO and its functioning. In Section 3 we present the goal of the pilot experience. We then describe the adopted technical solution in Section 4 and its application within the PICASSO platform in Section 5. Further considerations and lessons learnt are discussed in Section 6 and general conclusions are drawn in Section 7.

2 Background

An overview of the current status of the healthcare sector in Tuscany showed that many of the applications are still using expensive integration solutions. These are generally custom-made and unable to provide interoperability of a software with another coming from an environment committed to different integration approaches. The heterogeneity of the integration approaches is a big obstacle against cooperation among such applications and seriously compromises data transfer and exchange.

As said in the Introduction, HL7-V3 defines a reference model for the healthcare information production and a protocol that can be used for implementing semantic interoperability among software applications in the healthcare structures. To solve the interoperability problem, for each interaction between two applications this protocol defines: the message exchange communication patterns; the message format; in the message body, the meaning of each information field.

Today, the most advanced software companies use older versions of HL7 (HL7 version 2.x with x=3,4,5) because there is upward compatibility among HL7-V2 releases. On the contrary, a compatibility between HL7-V2 and HL7-V3 protocols is not guaranteed. So, the same effort is needed for integrating an HL7-V3 application with either an HL7-V2.x application or an application not conforming to HL7 protocol. This implies a low diffusion of the HL7-V3 protocol.

With respect to traditional integration platforms,

[1] Small and Medium Enterprise.

[2] PICASSO is an acronym from the Italian "Piattaforma per l'Interoperabilità e la Cooperazione Applicativa nelle Strutture Sanitarie ed Ospedaliere".

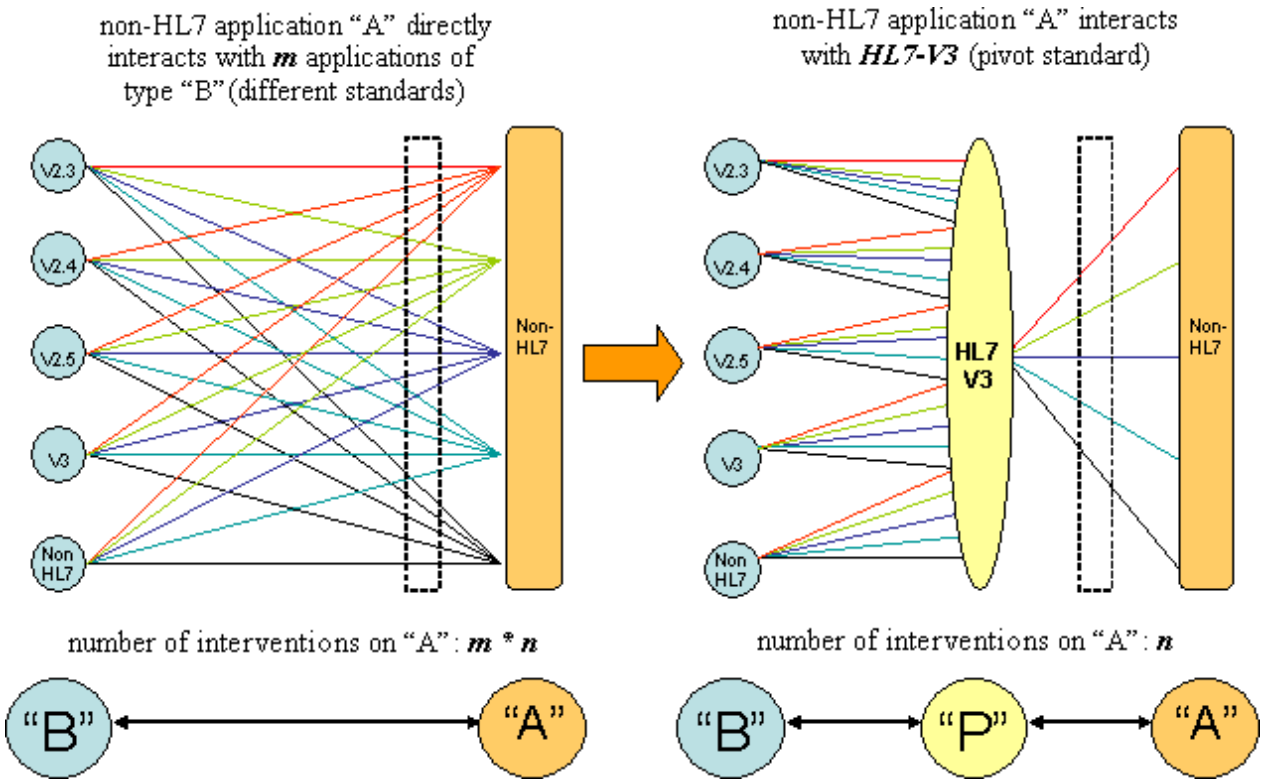

Figure 1. Traditional platform *vs* PICASSO

PICASSO reduces the number of system interactions by using the HL7-V3 as a pivot protocol. In particular, as we can see in the left part of Figure 1, in a traditional platform, an application "A" that needs to communicate with m different applications "B" has to exchange $n \times m$ messages, where n are the different types of messages. Using PICASSO (see the right part of the Figure 1), to communicate with m applications "B", an application "A" uses only n messages addressed to the HL7-V3 protocol. PICASSO will take over the burden of mapping the n messages into the proper messages for the m existing applications.

Thus, PICASSO drastically reduces the complexity of the point-to point approach to the transformation of the existing platforms by using easier and automated transformation steps. In particular, the automation process foresees the definition of three XML reference schemas: the HL7-V3 central schema and two interface schemas called *XML-Int-S* and *XML-Int-R* (where Int stands for Interface and S and R for Sender and Receiver, respectively). Note that these schemas are defined only once and for all on the basis of the processed events and the standard format used by the sender and receiver applications.

Thanks to the adopted XML notation, schemas can be employed for a quick transformation by means of an XSL stylesheet. XSL stylesheets are defined to transform XML-Int-S into HL7-V3 and HL7-V3 into XML-Int-R. The XML-Int-S and XML-Int-R schemas define an intermediate format and are used in PICASSO for reducing the complexity of the components (called Connectors) that transform the application data format into and from the XML-Int intermediate format.

A sketch of the PICASSO transformation framework is presented in Figure 2, where an interaction between two generic applications named Sender and Receiver is addressed. This general schema can be detailed to deal with different real interaction scenarios. The sender or receiver applications, for example, could be compliant with HL7-V2.x, or they could even not be compliant with any standard. Specific application pairs will require different implementations of the connectors.

3 Goal of the experience: Automated validation of XSL stylesheets

We have described the attractive features of PICASSO in terms of standard compliance and ease of instantiation of new integration channels. In addition to such business values, another important and competi-

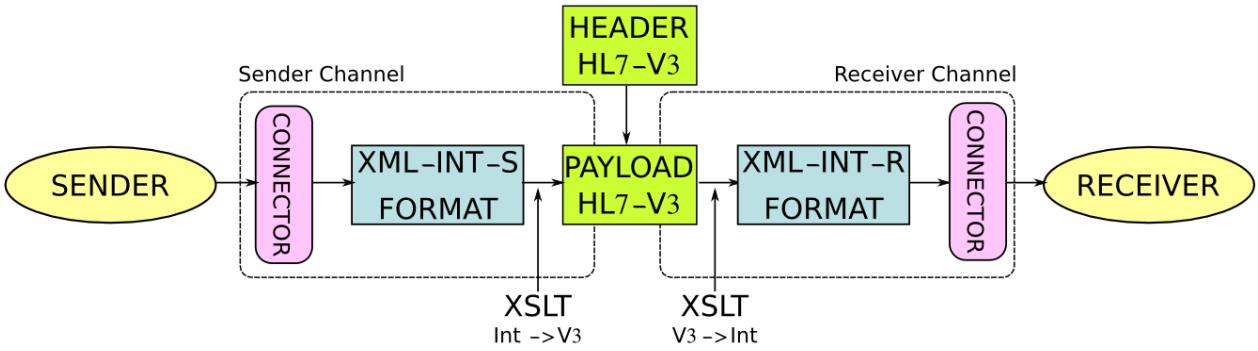

Figure 2. PICASSO transformation framework

tive result for Codices is the strict deadline in setting up an application channel within the PICASSO framework: the company's policy is to guarantee a complete setup in *no longer than three days*. This requires a very skilled personnel, a good organization of activities as well as the adoption of advanced technologies and extensive automation in the development process.

In the current process, Codices has identified validation as the weak stage. In particular, the testing of the XSL stylesheets involved in a channel implementation (see Figure 2) is perceived as a critical and time consuming step because still entirely manually executed. The particular nature of XSL stylesheets, containing a set of directives destined to be interpreted by an XSLT engine, requires a varied and specific set of test cases for verifying all the possible situations, that most of the times cannot be easily derived. Moreover, this testing process is by its very nature human-intensive and error-prone, if manually performed, because XSL is not purposefully conceived for human manipulation. We evaluated from past experiences that testing a 25-element XSL stylesheet takes approximately half a day, using a test suite around 50 or 60 cases. This is clearly a consistent portion of the three days available, and could easily result in a problem in the case of particularly complex stylesheets.

Our goal is therefore *the introduction of automatic support for the validation of the XSL stylesheets which transform to and from the HL7-V3 format*. We have performed a field overview and found some existing tools for XSL stylesheet testing. For example, UTF-X [2] is an extension to the JUnit Java unit testing framework. It provides functionality for XSL stylesheet unit testing, strongly encouraging a test-first-design principle (which is far from our process workflow). Other tools include tennison-tests [4], which allow users to write unit tests in XML and exercise XSLT from Ant; while Alster [3] is an XSLT unit testing framework which supports the creation and execution of XSL stylesheets containing specially marked test templates. However, also such tools are not suitable either, since testing remains heavily based on preparatory manual settings; this means that the correctness of the results eventually depends again on the expertise of tester.

In contrast, our target is to push automation in the testing stage as much as possible, so as to drastically decrease the time and effort required by the verification of the XSL stylesheets. The framework shown in Figure 3 describes the ideal approach we want to adopt for the PICASSO interoperability platform. Starting from a reference XML Schema (XML Schema A), an Instance Generator is used to automatically derive XML instances to be provided to the XSLT engine. The latter transforms the set of XML files into different XML files according to the rules defined within an XSL stylesheet. It is worth noting that the pure functional behavior of an XSLT engine allows to derive a really simple tester that has to take one single XML instance at a time and directly pass it to the XSLT engine.

Every time an XML instance is passed to the XSLT engine, another XML document will be created. Also, in this case it is not mandatory to have an XML Schema defining the target structure (actually, XSLT also supports output formats other than XML). Nevertheless, since our purpose is to test the correctness of the transformation rules, the availability of an XML Schema for the target format (XML Schema B) provides a simple oracle checking if the transformed instances conform to the target schema. As shown in Figure 3, this scenario leads to a *completely automated framework* to check if an XSL stylesheet is correct.

4 Experimented technology: TAXI

XML Schema is a formal, computer-readable specification for the input data domain. By processing the schema, a set of conforming XML instances can be de-

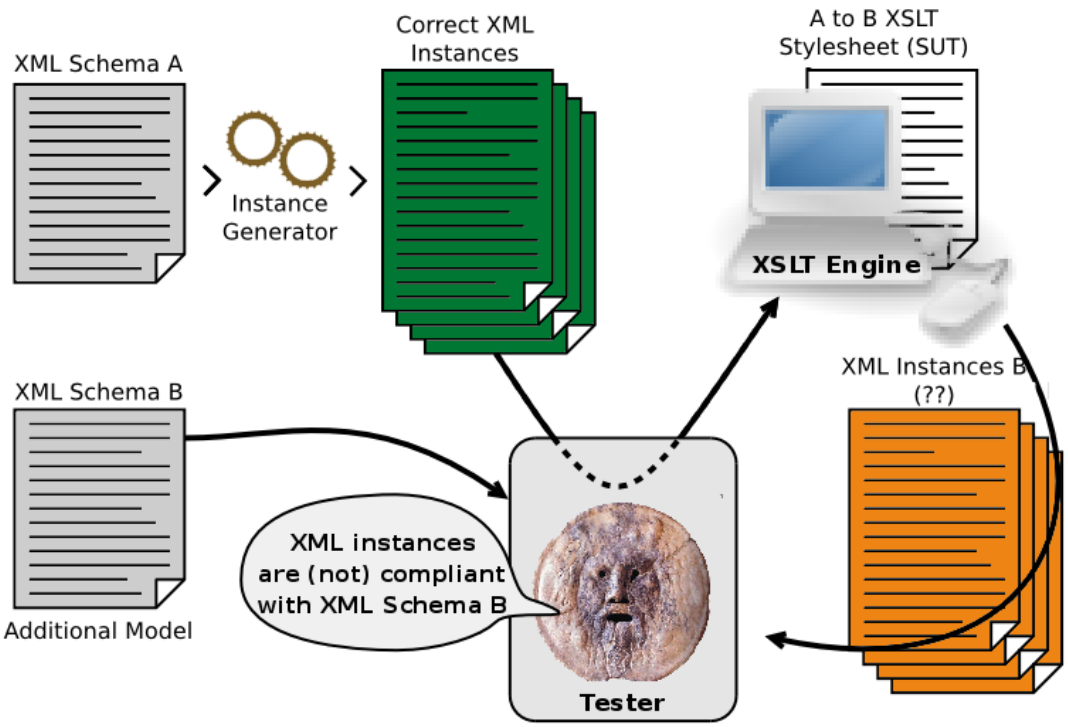

Figure 3. Fully automated XSL stylesheet validation

rived, and these can be used as test case entries for systematic black-box testing of the application. The execution of the application over the generated test instances can be used for conformance testing, i.e., to test how the application behaves on automatically generated conforming instances. Some proposals for deriving such instances are available in literature, for example [14, 7]. However, before fully integrating an automatic tool in our channel development process, the evaluation of the possible improvements was required.

In this section we briefly describe a possible implementation of the Instance Generator (as referred in Figure 3), useful for the automated derivation of conformance test suites based on an XML Schema specification. Among the applications available we decided to start our evaluation from the TAXI tool [7]. In the rest of this paper we report the results obtained.

TAXI (Testing by Automatically generated XML Instances) [6, 7] is a tool able to generate compliant XML instances from a given XML Schema. It has been conceived so as to cover all the interesting combinations of the schema by adopting a systematic black-box criterion. For this reason, TAXI applies the well-known Category Partition (CP) technique [15] to the XML Schema. CP provides a stepwise intuitive approach to identify the relevant input parameters and environment conditions and combine their significant values into an effective test suite. A detailed description of TAXI functioning and architecture is outside the scope of the present paper. We refer to [6, 7] for a more accurate description.

In the following, we only present the main activities of the TAXI component, which are more relevant for the understanding of this case study. TAXI activity starts with the analysis of an input XML Schema. In case *choice* elements are included into the schema, a set of sub-schemas are derived by selecting a different child from each *choice* element. In case of nested *choice* elements, a combinatorial explosion of *choice* children is performed. This ensures that the set of sub-schemas represents all possible structures derivable from *choices*.

The implementation of CP requires the analysis of the XML Schema and the extraction of the useful information. Element occurrences and types are analyzed and the constraints are determined from the XML Schema definition (normally an XSD file). In particular, boundary values for *minOccurs* and *maxOccurs* are defined. If specific values are defined for *minOccurs* and *maxOccurs*, they are used as boundary values, whereas if a *maxOccurs* attribute has an "unbounded" value, a user-defined default number is used. Exploiting the information collected so far and the structure of the (sub)schema, TAXI derives a set

of intermediate instances by combining the occurrence values assigned to each element.

The final instances are derived from the intermediate ones by assigning values to the various elements. Two approaches can be adopted, depending on the contents of the database embedded within TAXI:

- if a given element has a selection of possible values stored in the database (or in the schema itself through an *enumeration*), a random value is selected from those. This allows to have meaningful values, albeit fictional, in the instances. Storing data in the database is an operation that must be done manually: currently TAXI supports no means of loading data from an external source, although this might be a feature in a future version of the software. However, populating the database is an operation which must be done only once for all instances to be generated, and, since the database is persistent, the data can be reused throughout multiple executions of the test suite;

- if a set of possible values is not provided for the element, its value is randomly generated in each instance. This approach generates instances whose values are meaningless and illegible, however they are still compliant with the data types and their restrictions, making the instances still compliant with the schema. Of course, this approach does not require the initial population of the database and can be immediately adopted;

- obviously, it is possible to partially populate the database, assigning value sets to some elements or attributes and leaving the others empty for random generation. This mixed approach is less expensive than the full population and allows to focus a major attention on the most critical parts of the schema.

A special care during the final instance derivation is devoted to the *all* elements. Every time an intermediate instance contains an *all* construct, a random sequence of its children elements is chosen for generating the final instance. This new sequence is then used during the assignment of values to each element.

To make the generation more flexible, TAXI also provides different test strategies to pilot the instance generation. These include the coverage of all the possible (sub)schemas, or occurrence combinations, or value combinations, or simply the generation of a predetermined, user-defined number of instances.

5 Validation automation in practice

In this section, we outline our experience in using TAXI to test the correctness of the XSL stylesheets used within PICASSO.

For this purpose, we use a specific feature built in TAXI, which we refer to as the "transformation and validation" of a set of XML instances. Given a set of source XML instances (which may have been generated by TAXI itself from an input XML Schema), this feature comprises the following steps:

1. a user specifies an XSL stylesheet and a target XML Schema;
2. TAXI picks each instance in sequence, transforms it according to the rules in the stylesheet, and stores the output XML instance in a file;
3. TAXI validates the output instance against the target XML Schema;
4. finally, a log is generated that describes which of the transformed instances were conforming and which were not, and details the validation errors for the non-conforming ones.

This "transformation and validation" feature allowed us to carry out the whole experimentation according to the process described in Figure 3 within the TAXI environment and without the need of external tools.

We now consider the experiment setup. Structurally speaking, from an interface perspective there are three possible types of healthcare applications:

- HL7-V3
- HL7-V2 (several revisions of the protocol)
- non-HL7 (potentially infinite data structures)

This leads to a total of 9 categories of communication channels, which would correspond to the 9 cells of Table 1 below. This table also shows which types of XSL stylesheet are not needed in the PICASSO framework (marked with an X). Since we are using HL7-V3 as a pivot, every communication must be adapted to the HL7-V3 protocol. Therefore there is no need to make direct transformations between different revisions of HL7-V2 or proprietary protocols; additionally, it is pointless to have the transformation from HL7-V3 to HL7-V3 (identical transformation). This leaves a total of 4 conceptually different XSL stylesheet: non-HL7 to and from HL7-V3, and HL7-V2 to and from HL7-V3.

As an example, let's consider two communicating applications, A which is compliant with the HL7-V2

	non-HL7	HL7-V2	HL7-V3
non-HL7	X	X	
HL7-V2	X	X	
HL7-V3			X

Table 1. Communication channels

standard, and B which uses a proprietary format; the communications cross a pivot channel, called P, based on HL7-V3. In our example, A performs a remote procedure call (RPC) request to B. The full communication channel is made up of a total of four transformations:

1. HL7-V2 to HL7-V3 for transforming the data of A's request into P's format;
2. HL7-V3 to the proprietary format for delivering the request to B;
3. proprietary format to HL7-V3, to make the response compliant with P;
4. HL7-V3 to HL7-V2 to deliver the response data to A.

Each of these steps requires a separate XSL stylesheet. The four stylesheets are autonomous and require separate development processes, however these processes are structurally identical. For this reason, our experience focuses only on one of these transformations, specifically the first one ($A \rightarrow P$).

As explained in Section 2, the interfaces are first converted to XML formats which are as close as possible to a predefined schema, to ease the process of developing the stylesheet. Therefore, the XSLT transformation applies to the XML-Int format and HL7-V3. In our example, we used the XML-Int-S corresponding to an application compliant with HL7-V2. The experimentation was conducted in two separate steps.

5.1 Initial development

In the first phase, we wanted to uproot every possible bug or problem. A transformation error might be related to bugs either in the XSL stylesheet or in the interface schema. Our purpose was to detect both types of errors. To evaluate the benefits of the automated approach versus the traditional manual testing, we decided to carry out both and compare their results. So, after developing preliminary versions of the schema and the stylesheet, we initially executed our usual set of manual tests (as described in Section 3). These were successful, which led us to the conclusion that this part of the channel was correct.

Then we switched to the automated testing based on TAXI. As outlined in Section 4, we had the two options of either populating the database with meaningful values or leaving it empty. The former choice produces "neat" instances which actually resemble operative data, but it requires an "onset time" which may take up to a few hours, while the latter option produces less readable instances but saves up time. We chose the latter.

During this first trial, we generated a total of 50 instances using TAXI, and transformed them using the XSL stylesheet. The output of the transformation and validation showed us that there were some unchecked errors, which had not emerged during the manual testing. Upon deeper analysis, we noticed that some of the validation errors were due to problems in the XSL stylesheet, while the others were related to problems in the XML-Int-S schema. We then proceeded to fix the errors discovered and went on to the second step.

5.2 Fully automated testing

In this second step, we wanted to test the final versions of the schema and the stylesheet with a higher number of test cases. The manual testing was avoided this time, because building a lot of test cases would have required too much time, something that cannot be afforded by Codices in an actual production.

TAXI, applying the Category Partition algorithm and setting an upper limit for unbound cardinality elements, predicted a total of 131072 possible instances of the XML-Int-S schema. Once established the number of instances to be run, one of the functionalities of TAXI is the possibility of applying a test strategy, based on pair-wise approach [8, 5], for picking the instances that maximize the fault detection capability.

Thus, for the second step, we decided to run a test session of 200 instances, and this time the validation passed on all instances. This was enough for us to assert that the schema and the stylesheet were acceptable. However, in a production context, a higher number of test cases might be used; if the source schema is not exceedingly complex, even an exhaustive testing of all the possible data structures could be carried out, in still less time than our manual testing of 50–60 instances.

Several benefits have emerged from this experience:

- an in-depth testing using a smart algorithm to select the useful instances likely provides better results than a manual testing, because all the significant parts of the structure can be exploited. There is no guarantee that the errors would have been detected without a systematic approach;

- the improvement in time is priceless. The automated methodology required a matter of seconds to evaluate the number of possible structures and generate 200 instances, and a similar time to transform and validate all the instances and produce the output log. Even if we had taken the time to populate the database, which might have taken a time between 30 and 60 minutes, the results are astounding when compared to the half day or so required for a manual test with around 50 instances;

- the final conformance testing with a high number of test cases is simply not affordable with a manual approach (much less one which exploits all possible data structures). From this perspective, the automated methodology offers a much higher degree of warranty that the results are correct.

6 Experience feedback

The experience collected over the years evidences that the current Codices processes are effective. However, our success is greatly due to the strong expertise of the developers and their profound knowledge of the application environment. Skilled personnel can focus immediately on the peculiarities of each XSL stylesheet, and consequently develop a useful set of XML instances, each one able to stress different key parts of the transformation. However, due to time constraints (a deadline of three days for the whole channel), it is not possible to verify all the transformations implemented in each stylesheet, and only a limited number of instances can be developed. Moreover, this manual process is critical, because when a novel application is encountered it may clearly require unpredictable learning cycles and error-prone tuning processes.

For such reasons, we have planned to push test automation as far as possible. From the experience reported here, evidence has been gained that the application of advanced tools for automatic XML instances derivation can improve the performance of the Codices' validation phase in a significant way. We have not carried out a formalized empirical assessment to get precise quantitative estimations of the gains in time reduction, or of the improvements in test coverage or effectiveness. We are aware that from an academic perspective a formal experiment would have produced added value. However, this was not in scope with our objective, and would have meant a lot of additional effort and cost with little return for the company.

To us, the hands-on experience with TAXI represented a very good starting point towards making validation faster, predictable and more effective. The gains achieved by the usage of the tool, at practically no cost, are so evident that we did not need to get precise numbers to be convinced. Not only has the testing time for a single stylesheet been drastically reduced from about half a day to a few minutes, but we also get an improvement of the effectiveness of the test suites. Thanks to the test strategy implemented in the TAXI tool, the XML instances have been generated systematically, considering two different kinds of variability: in their structure and in the values assigned to the various attributes and elements composing the instance.

When combined, these two factors guarantee that the XML instances used in the testing phase adequately cover the full spectrum of the whole possible XSL input domain and highlight a huge variety of possible problems (many of which had never been revealed before).

This positive experience has persuaded Codices to introduce the automated framework introduced in this paper as part of the channel development process. This will reduce the time required for channel setup, and improve the competitivity of Codices in developing interoperability solutions within the healthcare sector. Moreover, the skill and knowledge of the developers will be better focused on the other critical aspects of the platform implementation, with a consistent improvement in terms of quality of the released products.

7 Conclusions

Testing is always reported, particularly by software companies, as one of the most expensive activities in software production. As a consequence, companies are more and more eager of having tools and techniques that can support the different testing activities.

In this work we showed how a technique and a tool (TAXI) developed within an academic context have been applied in an industrial setting with minimal effort and interesting results. The peculiarities of the selected application domain were suited for defining a completely automatic testing phase based on the availability of XML-based descriptions for the exchanged data.

To fulfill the vision of an open, interconnected world, the key notion is to guarantee an interchange format as well as the trustworthiness of data and quality of the applications using those data. Codices' PICASSO framework for healthcare interoperability focuses on the consistent exchange of data among systems using different data formats. To achieve this, an important component of the framework is devoted to transforming documents from a format into another, both of which are specified using the XML Schema language.

TAXI's functionality is to automatically derive correct XML instances starting from an XML Schema. The generation is carried out according to powerful and long-established test strategies which guarantee the derivation of a set of instances meaningfully distributed over the input domain. When validating XSLT transformations, the testing process can be fully automated, since a reliable oracle is readily available as the XML Schema corresponding to the format of the document expected as output.

The experience reported here has been a win-win endeavor for both the involved industrials and the researchers. The noticeable improvements in terms of time and effectiveness of the testing phase have convinced Codices of the opportunity to introduce the TAXI tool into their deployment workflow. The adoption of TAXI can drastically improve their processes, shifting from a testing phase typically made up of ad-hoc instances, hand-coded by Codices personnel, to a completely automated one.

For the CNR researchers, the opportunity of experimenting TAXI on PICASSO has highlighted several weaknesses of the tool when scaling up to a schema taken from a real application domain, and paved the way for many possible improvements. It may well be that using an XML Schema as the oracle is not sufficient to reveal all possible problems; we believe it is useful to further enrich the oracle and augment the constraints to be checked, for instance using XPath, while still maintaining a completely automatic approach. As explained in Section 4, the population of the database is a critical step. Since information systems are more and more hungry for semantic interoperability, this step could fruitfully interact with domain ontologies to breed the database with meaningful input values.

Together, we plan further experiments in the near future, to build the basis for a wide-scale dissemination within HL7 Italia of the benefits of highly automated integration through PICASSO.

Acknowledgements

This work was partially supported by the TAS^3 Project (EU FP7 IP No.216287) and by ART DECO (Adaptive infRasTructure for DECentralized Organizations), an Italian FIRB (Fondo per gli Investimenti della Ricerca di Base) 2005 Project.

The authors wish to thank Paolo Marcheschi of the CNR Institute of Clinical Physiology for triggering this experience and for inspiring discussions.

References

[1] W3C extensible markup language (XML). `http://www.w3.org/XML/`, 1996.

[2] Unit Testing Framework - XSLT. `http://utf-x.sourceforge.net/`, 2004.

[3] Alster-XSLT Unit Testing Framework. `http://alster.sourceforge.net/`, September 2006.

[4] tennison. `http://tennison-tests.sourceforge.net/index.html`, July 2006.

[5] Pairwise testing. `http://www.pairwise.org/`, accessed Feb. 9, 2009.

[6] A. Bertolino, J. Gao, E. Marchetti, and A. Polini. Systematic generation of XML instances to test complex software applications. In *Proceedings of Rapid Integration in Software Engineering, (RISE 2006)*. LNCS 4401, September 2006. Geneve, Switzerland.

[7] A. Bertolino, J. Gao, E. Marchetti, and A. Polini. Automatic test data generation for XML Schema-based partition testing. In *Proceedings of International Workshop on Automation of Software Test 2007 (ICSE'07 companion)*, Minneapolis, Minnesota, USA, May 2007.

[8] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: An approach to testing based on combinatiorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, July 1997.

[9] Health Level Seven. `http://www.hl7.org/`, accessed Oct. 9, 2008.

[10] HL7 Italia. `http://www.hl7italia.it/`, accessed Oct. 9, 2008.

[11] G. Hohpe and B. Woolf. *Enterprise Integration Patterns*. Addison-Wesley Professional, Reading, MA, 2003.

[12] Interoperability. Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Interoperability`, Accessed Oct. 9, 2008.

[13] P. Miller. Interoperability - What is it and Why should I want it? *Ariadne*, (Issue 24), June 2000.

[14] J. Offutt and W. Xu. Generating test cases for web services using data perturbation. In *Workshop on Testing, Analysis and Verification of Web Services*, Boston Mass, July 2004.

[15] T. Ostrand and M. Balcer. The category-partition method for specifying and generating functional tests. *Communications of ACM*, 31(6), 1988.

[16] M. Uschold and M. Gruninger. Ontologies and semantics for seamless connectivity. *SIGMOD Rec.*, 33(4):58–64, 2004.