Turning a Suite of Modeling and Processing Tools Into a Production Grade System

Pascal Rivière[1], Olivier Rosec[1]

1 Caisse nationale d'assurance vieillesse (Cnav)
110 – 112 avenue de Flandre, F-75019 Paris
pascal.riviere@cnav.fr, olivier.rosec@cnav.fr

Abstract. The French national agency for old age pensions has evolved, along a systemic vision of all the issues to be covered, a generic set of tools to design and process structured file formats. This set of tools has turned into a production grade modeling and validating suite which relies for its distribution on the continuous integration of its various components. As such, continuous integration has here turned into a system which industrializes the production of systems of tools, each characterized by the file format it implements and the version under which it is released.

1 Introduction

Social security agencies have to rely on a massive influx of data to assess benefits and collect contributions. The data comes in thanks to structured file formats. Processing and validating these files is not easy, because of the many changes in their specification each year.

This is why the national agency for old age pensions (Cnav) wanted to avoid rewriting endlessly its file processing applications, and wanted to maintain a constant level of processing quality. A general approach was necessary.

To achieve this, the national agency has elaborated a generic set of tools [Rivière, Rosec 2013]. It was initially used on the N4DS standard, and then on the new DSN standard (N4DS = Norme de Déclarations Dématérialisées De Données Sociales, DSN = Déclaration Sociale Nominative).

This approach can be applied well beyond the initial problem domain: collecting data from payroll systems.

The suite of tools has been improved and extended along a systemic view of all issues to be covered when dealing with structured file formats: design, validation logic, testing logic, documentation both of the file formats and the semantics of the business data they carry.

2 What is an interchange standard?

Elaborating a generic set of tools for modeling interchange formats and for generating the corresponding validating components brings back to the original issue: defining a metamodel for an interchange standard. We define it by the following characteristics:

- It should implement a conceptual model of the business domain;
- It can be derived into a series of message models carrying the business data payload
- Each message is a hierarchy of data blocks which respect a certain structure and multiplicity (0,*)
- The structure of a block is defined as an ordered series of data elements which respect a certain syntax and multiplicity (0,1)
- The value and syntax of a data element is defined by a business datatype restricting one of four technical datatypes: string, decimal, enumeration, external referential
- Each technical datatype has facets which help define business datatypes: minimum and maximum length, a regular expression for string and decimal etc.
- Semantic consistency between data elements is enforced by rules.
- Because the interchange standard has to support the interchange scenarios defined by its user community, message models can themselves be inserted between a header and a footer composed of blocks linked to the applicative logic of the actual exchange system.

On top of this metamodel one can model many interchange formats. Here are the metrics for a few of the interchange formats modeled with the suite of tools.

Number of	*N4DS v01x08*	*DSN phase 1*	*DSN phase 2*
Data blocks	*124*	*29*	*33*
Data elements	*687*	*202*	*258*
Validating rules	*995*	*198*	*210*

The DSN message is the first for which a conceptual model was elaborated before working on the corresponding file formats. The idea is to derive the message model from the class diagram (below). It should be noted that such a derivation cannot be 100% deterministic and requires some choices [Elmasri, Navathe 2011].

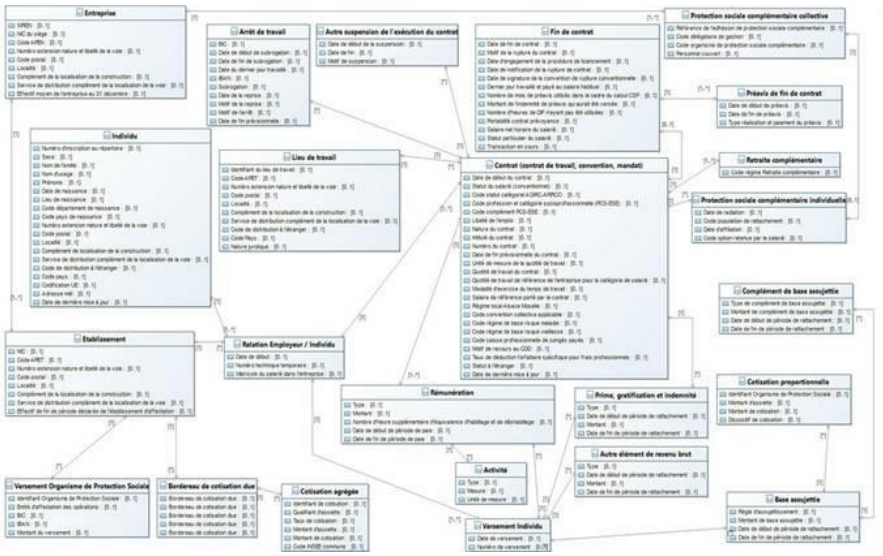

Fig. 1. DSN phase 2 class diagram

The figure below shows how various message models can be inserted between header and footer blocks.

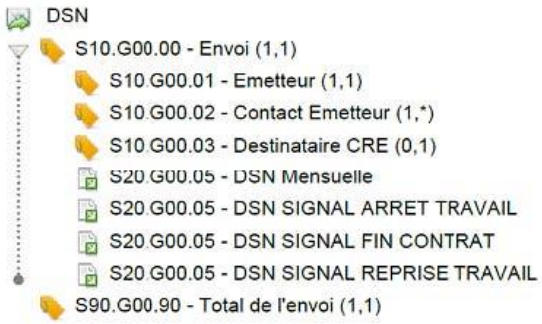

Fig. 2. DSN phase 2 Message Hierarchy

The figure below shows how the biggest DSN message model, the monthly DSN, is represented as a hierarchy of data blocks in the Designer.

Fig. 3. DSN phase 2 Monthly Declaration Hierarchy

Each data element is described in detail in the .pdf export of the specification.

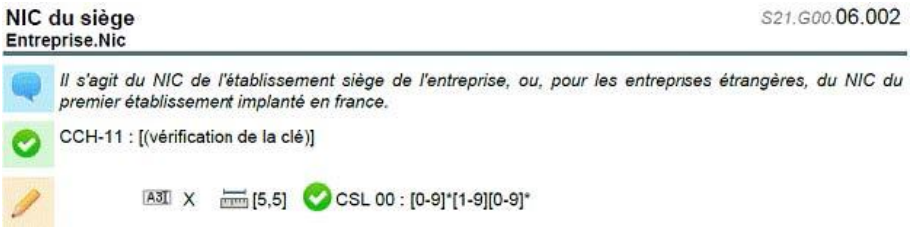

Fig. 4. PDF export for a data element

3 Tools, what for?

The generic set of tools covers the following functionalities (corresponding tool between brackets):

- Modeling an interchange format and defining its validating rules (Designer)
- Documenting the data carried by the interchange format (Semantic Repository)
- Generating the specification of the interchange format as a .pdf + various .csv exports (Designer)
- Generating validating components which can be integrated in a process (Parser within the Designer, Validating building block composed of Knowledge Base cum Validator)

- Generating a validating tool with a graphical user interface for a local compliance check before sending files to the remote processing platform (Autovalidator)
- Testing the generated components (Tester)

This series of tools does not owe its birth to a careful plan. One should rather think of it as an emergence. It is through the efforts of the national agency to organize the original system of goals and means and the various artefacts supporting them into a coherent whole that new tools kept arising until the present list was obtained.

How do these tools operate? This is what this part of the paper is about.

3.1 The Designer

This tool is at the heart of the suite. As previously noted, the Designer generates:

- The specification of the interchange format which will be disseminated throughout the user community for implementation (.pdf, .csv export, XML schemas)
- But also the knowledge base which will be read by the Validator to check whether the actual files sent by the users comply with the specification

As a modeler of data structures, datatypes and messages, the Designer is fairly classical. But as a repository of validating rules relying on first-order predicate logic, it is not.

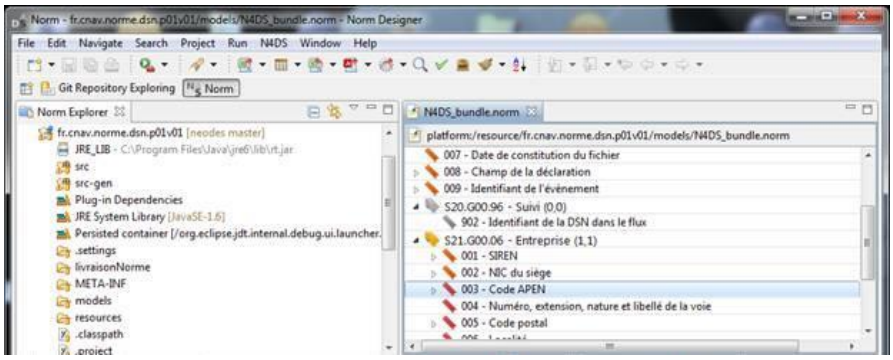

Fig. 5. Top part of the Designer UI

In the modeling perspective, the screen is split up between the project view (top left), the outline view of the current model (bottom left), the view displaying the current model element (top right) and the view showing the properties of the part of the model currently in focus (bottom right).

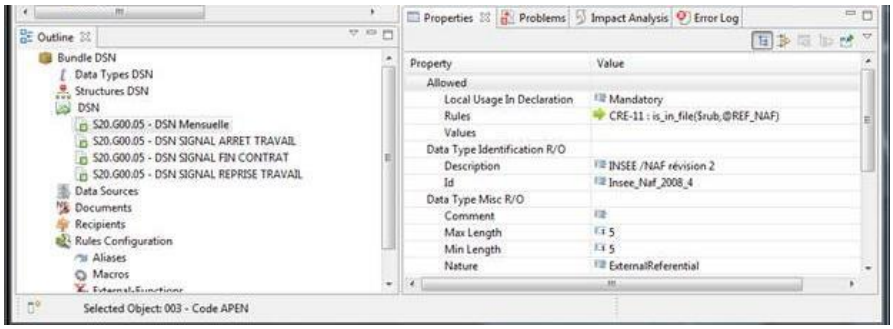

Fig. 5. Bottom part of the Designer UI

Rules are modeled under the data element to which they are attached in the functional specification for the file format.

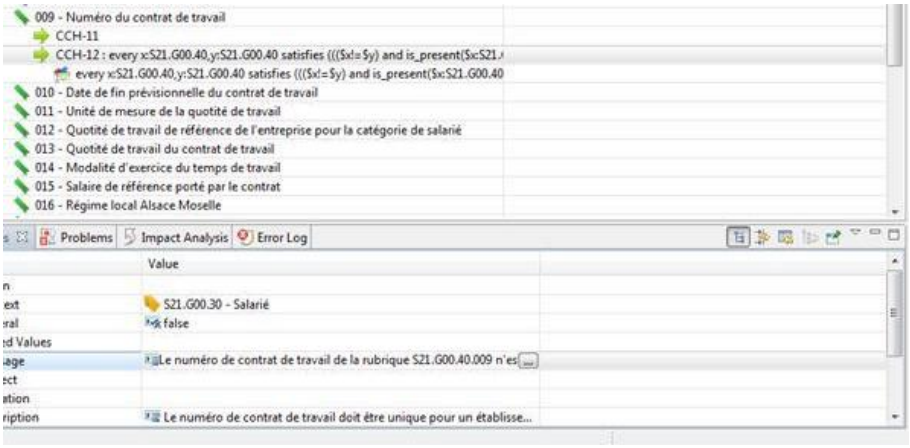

Fig. 5. Rule as displayed in the Structures and Properties views

A rule has a subject and a context, that is, a position within the message tree from which all paths to the data elements called by the rule will be calculated. The textual DSL corresponding to the functional rule is entered in the model through a widget.

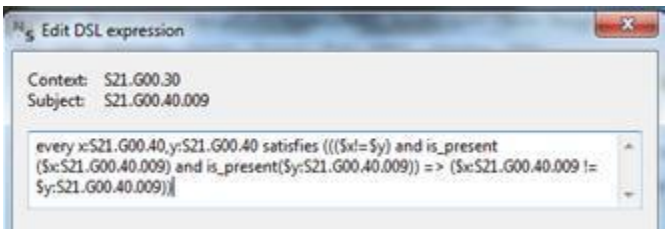

Fig. 5. Rule as displayed in the DSL widget

A message will be returned to the user in the report stream if the processed file does not satisfy the rule.

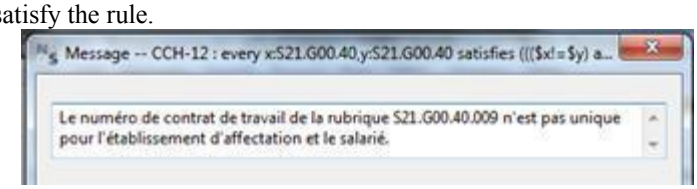

Fig. 5. Rule user message

When the knowledge base is compiled, the rule will be implemented as a Java class along with the set of utility classes loading and unloading at runtime the data elements called by the rule.

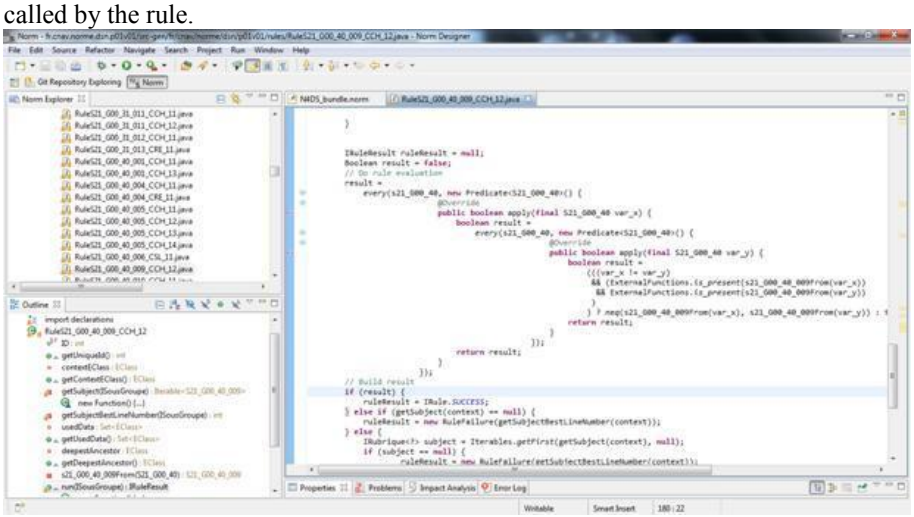

Fig. 5. Java implementation of a rule

3.2 The Validator

When a file is processed by the Validator, it goes through several stages:

- Conversion into XML
- Syntactical validation
- Semantic validation

As the file is processed, the Validator emits a report stream. The screenshot below shows the console and part of the report as transformed from XML to HTML by a local script.

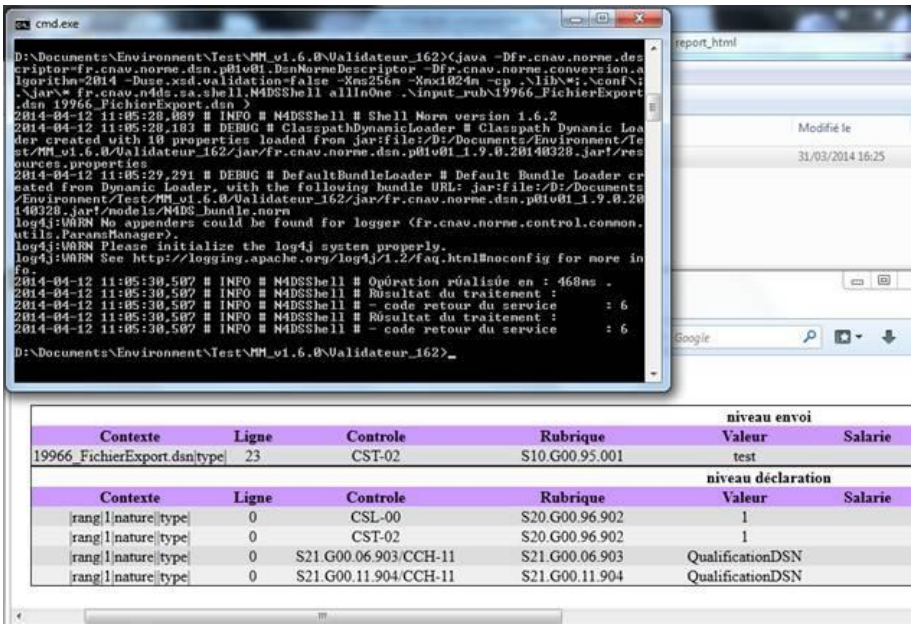

Fig. 5. Command line view of the Validator against backdrop of HTMLized report

The Validator has been improved: the report stream can be customized to the needs of any user community, thanks to an architecture which now separates events logging and report serialization

The Validator has been rolled out in production for the DSN.

A POC has been made to validate file formats circulating between old age pensions agencies to consolidate data about pensioners claiming the minimum entitlement.

3.3 The Tester

Work is under way on a testing tool which should:

- Validate the initial knowledge base against the specification it implements
- Ensure non-regression on validating components before roll-out
- Generate semi-automatically test suites.

The Tester is an RCP tool which loads a test suite file into a series of test cases which are then automatically executed and parsed to compare the obtained result with the expected result.

3.4 The Auto Validator

Users can now check whether their files comply with the specification before sending them to the processing platform.

The Autovalidator tool reads the same knowledge base and executes the same validating logic as the Validator in production.

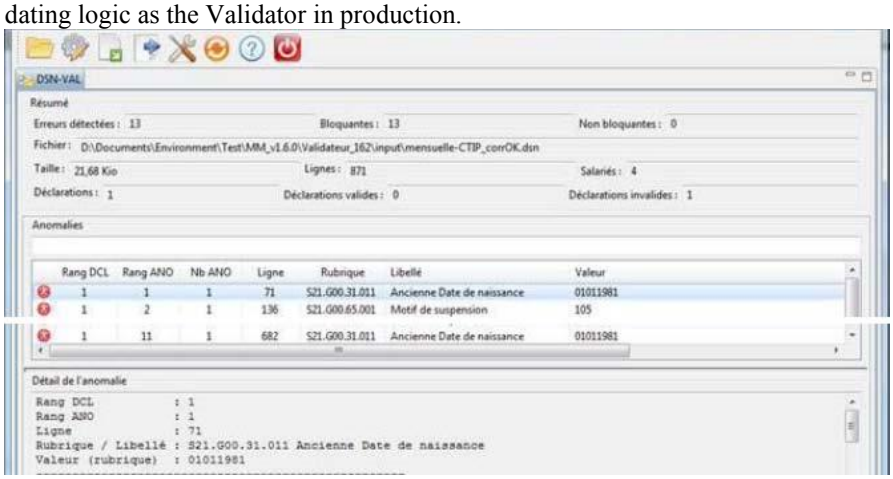

Fig. 5. Screenshot showing local validation with DSNVAL tool

Anyone can download the Autovalidator tool for the DSN, DSNVAL, on the Internet: http://www.dsn-info.fr/precontrole-dsn-val.htm

3.3 The Semantic Repository

A Semantic Repository now links data definitions coming from the conceptual data model of the business domain to their implementation in the file format specification. A query tool helps navigate through the Repository, using various approaches: selection criteria, full text research, keywords...

Fig. 5. Welcome page of DSN Semantic Repository

4 Implementation.

All the tools call on the facilities offered by the Eclipse Modeling Framework.

The Designer is built on top of the Eclipse platform. An effort has been made to organize its plug-ins into coherent features. It has moved from a mere graphical modeler to a developing tool which federates into a specialized plug-in all the projects an interchange format relies on for its successful implementation: modeling projects, and Java projects too in the guise of the functions extending the textual DSL, the few remaining rules written directly in Java, the Java serialization for the report stream.

A class known as a Norm Descriptor calls the validating logic persisted in the knowledge base. The knowledge base itself is a Java archive organized in layers according to the processing stages of the Validator.

In the models layer, one finds the files which describe the syntax of the model the actual files have to comply with, in terms of data blocks, elements and types.

A Java project carries the logic for the semantic validating rules and the report serialization.

Report logic relies on a model which is shared by the Designer and Validator.

The conversion algorithm which transforms the input files into XML has evolved at the beginning of 2014 to throw an error when a data block or element is out of sequence with regard to the “covering message” (the superset of all message models for a file standard). It thus contributes to the validating logic. Previously it generated extra

data blocks elements to generate a sequence of XML elements which would always be valid, but this strategy led to misleading error reports at the syntactical stage.

This is the most recent change brought about by the need to bring the Validator to production grade level, after the replacement of the initial off-the-shelf engines. The semantic validation was ported from Saxon and Schematron to a Java API in 2012 for performance, and the syntactical validation was ported from Xerces to the same Java API because users did not understand report messages phrased in XML constructs.

5 Continuous integration.

Originally, continuous integration covered only the suite of tools which were then more or less used on a standalone basis by the unit which specializes in the generation of the format models and validating components within the national agency.

But as the Validator was being prepared for production, it was considered safer to include in the testing strategy the knowledge base itself.

Now continuous integration encompasses all source control repositories, whether for tools or file formats, and builds to roll out fully tested components can be specialized according to the phase and version of the file format for which deliverables must be shipped.

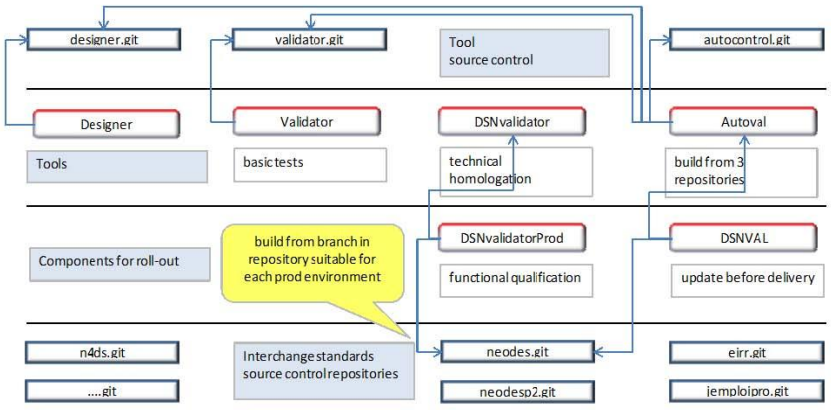

Fig. 5. Code repositories called by builds according to tools and deliverables

Continuous integration has become a system which helps produce systems of tools for structured file formats, each system of tools and deliverables being specialized into a given file format and released under a certain version.

Genericity still prevails thanks to the separation between models representing business rules and transformations implementing the processing logic.

One could almost describe the system of code repositories and build scripts as an autopoietic machine perpetually regenerating tools and deliverables.

6 Conclusion

The suite of tools is used intensively and successfully for the DSN project.

The national agency wants to make it yet again more generic in order to generalize its use for the validation of most or all structured data interchanges between other institutions and itself.

It is also investigating the possibility of turning the suite of tools open source in the hope it nurtures a new way of thinking about and working with structured file formats.

References

1. Pascal Rivière, Olivier Rosec, “Model-Based Interchange Formats: a Generic set of tools for Validating Structured Data against a Knowledge Base”, CSDM 2013
2. R. Elmasri., S.B. Navathe, “Fundamentals of database systems”, Addison-Wesley 2011