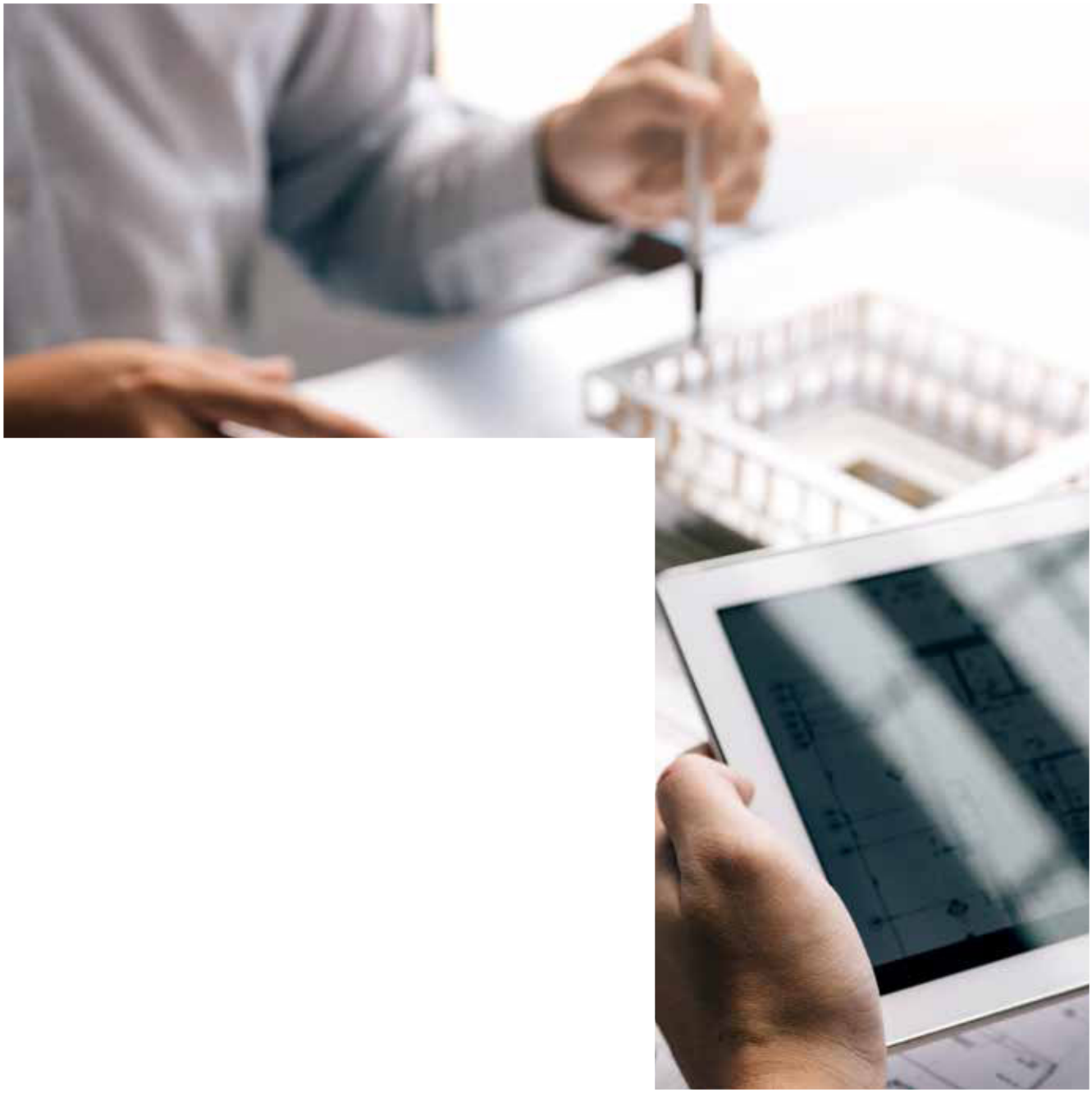


Canonical Data Model as Single Source of Truth for Documentation and Implementation

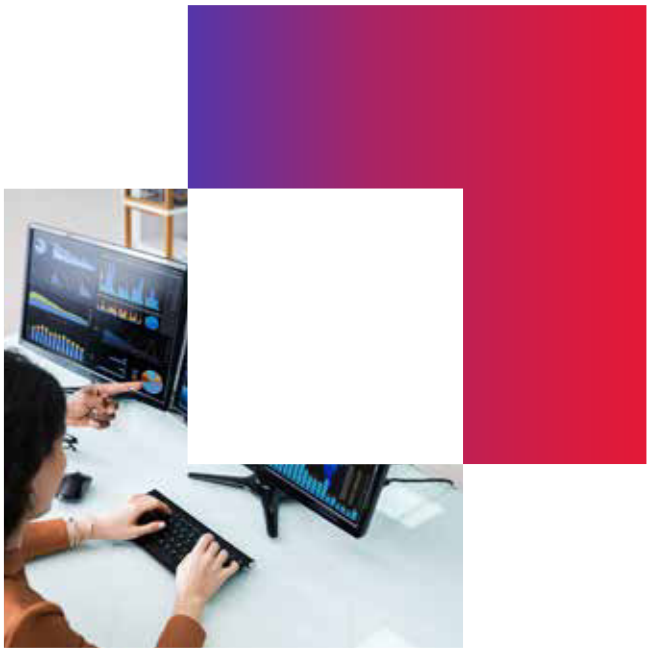
Derive multiple implementations alongside comprehensive business documentation from a single source

Introduction

Enterprises exchange data, both among their own applications and with external parties. Defining, maintaining, documenting and enforcing a data model is a challenge that involves many stakeholders and diverse technologies and, let's face it, is not always a complete success. This white paper describes an approach for continuous integrated data modeling across business functions and technology silos.

The classic approach

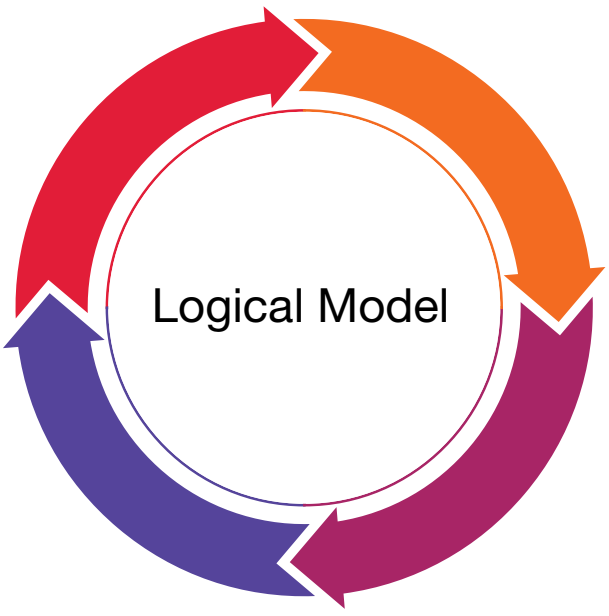
In order to send business data from one system to another, the details of the data exchange need to be described in a way that is verifiable for business stakeholders and suitable for technical implementation. This process typically starts with business subject matter experts working together with architects to define and document what is known as a logical model, which describes the “what” of the data to be exchanged independently of the “how” of technology. On the basis of this, solution architects then derive a representation that suits the technology they are responsible for, e.g., a relational database (physical model). This physical model is handed over to developers for implementation. Changes are supposed to be fed back into the logical model, and the cycle begins again.



The challenge

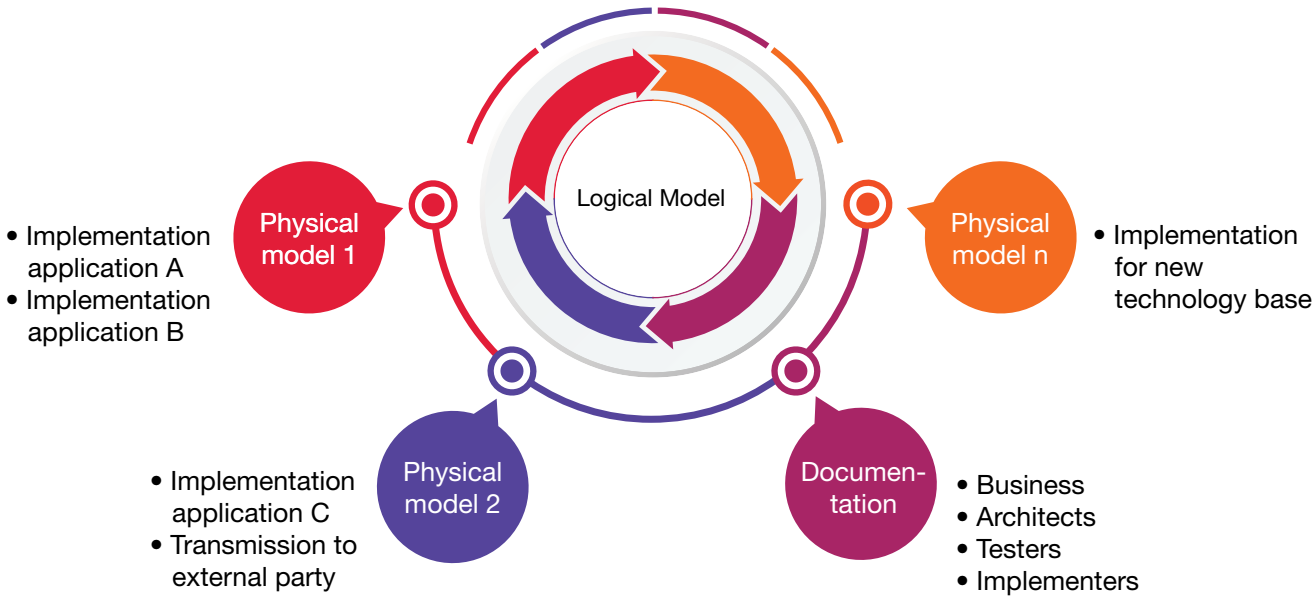
In real life, the logical model is often initially created with great enthusiasm, but due to time and budget constraints any update stemming from defect resolution or change requests only makes it into the implementation, and with luck into the physical models. As business and systems evolve over the years, the “real” model, including the corner cases, can be found in the implementation stage, and the logical model falls out of use.

Furthermore, each handover between stakeholders and each conversion of the model into a different form diminish the available understanding of the business purpose. The logical model often contains explanations of business scenarios, while the physical model describes technical relations and constraints, and only the implementation contains the most detailed design choices. In many cases, therefore, no business background information is readily available to developers working on a change or QA experts verifying a defect fix.



The integrated approach

To ensure that all the various representations of the business model are in line and that the full business understanding is propagated across all stakeholders, one single source must drive the full cycle of models and documentation in a fully automated toolchain. The automation of model and documentation generation is key: It must be easier to update the logical model and regenerate artefacts than to tinker with the implementation alone. The documentation must be available in a form that can be used and fully understood by all stakeholders, as the “single source of truth” from business subject matter experts to architects, implementers, testers and help desk and IT production staff.



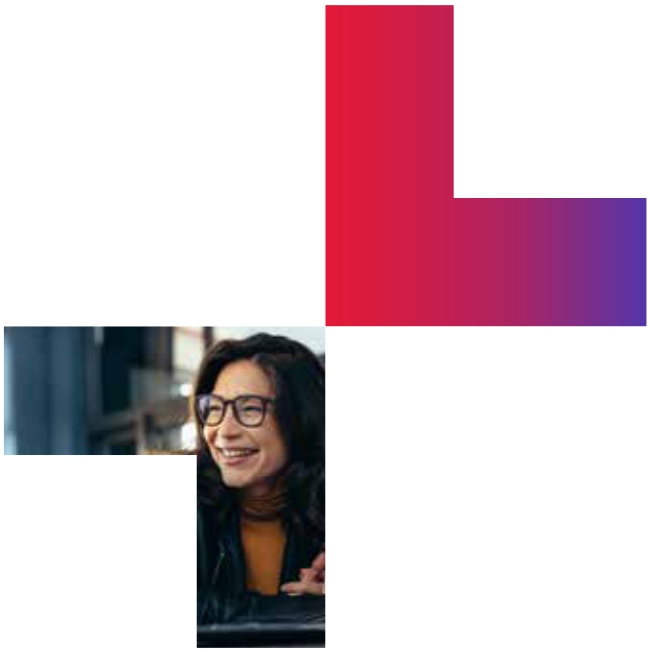
The limits

Various technologies and toolsets have different capabilities, so how can one source model be converted into arbitrary target representations? Is this not bound to be a failure?

Indeed the expressive power of different model technologies such as XML Schema, JSON Schema and DDL for relational databases do not fully overlap. Each technology offers possibilities that are simply not available in the others. Therefore, an approach that attempts to accomplish full equivalence quickly becomes awkward and ultimately fails, as numerous abandoned attempts easily found on the internet prove.

But in many business settings, not all of the expressive power is needed. Instead, a subset that is available in many technologies may well be sufficient. Typically, business data is grouped into business entities, for example orders, which have elements on “header level” that exist once for each entity, and repeating elements on “line level,” such as order lines. Elements should be typed, for example as dates, numbers, or strings of certain formats, they should have names that are easily understandable, they may be mandatory or optional, and both their technical nature and their business meaning should be documented.

The above properties may be expressed in all the relevant model representations that the author has so far encountered.



There are applications of models that cannot be bound by these restrictions. For them, the integrated approach is not suitable. However, the vast majority of real-life business models used in enterprise data exchanges can benefit, as the restrictions do not pose a serious drawback.

A use case

A global enterprise maintains two very large business objects for transport planning and execution. One describes items shipped together in a header-line structure, while the other relates to the transport booking and execution with the carrier. Both objects need to be shared in a landscape of more than 200 different systems, which are a mixture of custom-built legacy applications based on Oracle databases, bought applications from various vendors, and – the newest addition – cloud applications running as “software as a service” (SaaS).

The knowledge about the previous data model was scattered among an enterprise logical model with no further documentation maintained in a tool that no longer existed, and scarcely documented database models for various applications. Changes almost never made it back into the EA model, and the actual use was very limited.

XML schema as a logical and physical model

We have chosen XML Schema (XSD) as the primary model. Here, the logical model is at the same time one of the physical models used for web service (SOA) integrations. While this contradicts the “best practice” of always keeping logical and physical modes apart, it results in a strong drive to keep both models up to date, as they share the actual artefact. The file used to describe and approve the business model is the very same used to implement SOA integrations.

XML Schema is a well-established standard with plenty of tools available, which prevents vendor lock-in. If the provider of the current modeling toolset disappeared, an alternative could be picked without losing any of the investment. Indeed, different departments have decided to use separate tools (JDeveloper, Eclipse, MS Visual Studio to name a few) to work on the same model.

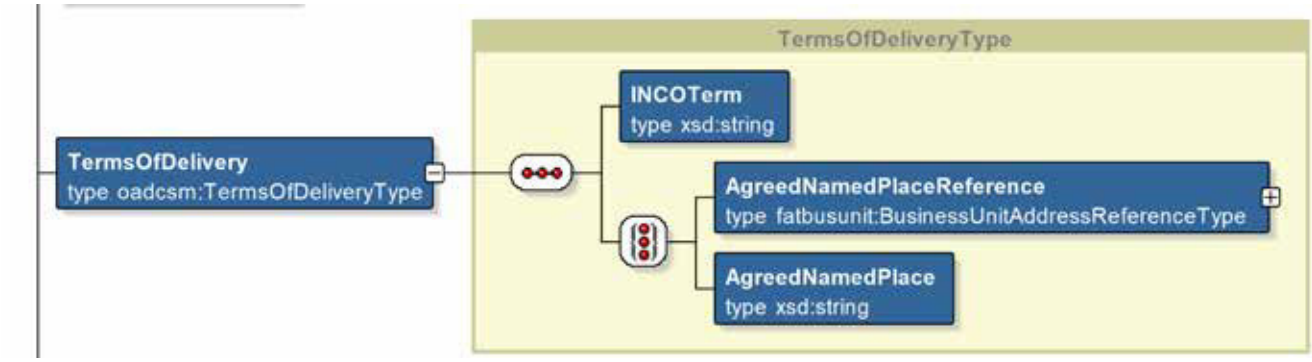
The XML Schema language has many expressive capabilities such as substitution groups which do not have an equivalent in all target modeling languages. Therefore, clear modeling guidelines have been drawn up to ensure that only the common capabilities are used. Although both of the business entities in question are rather complex (more than 1500 lines of XSD code), these restricted modeling guidelines have not posed any serious constraint to expressing the actual business data.

The modeling tools for XML Schema also provide a graphical view of the schema, clearly showing which elements are mandatory, which are optional, and which compound elements belong together. Due to a convention of using plain English with minimal abbreviations for the element names, it is easy for business subject matter experts to immediately understand the model itself without needing additionally prepared (and hence error-prone) presentation material. The enterprise architect and the business stakeholder literally work together on the same model, which boosts understanding and correctness.

While clear and consistent naming is key to understandable models, thorough business level documentation should be provided with regard to the purpose and expected use for each field. In XSD, this is reflected in annotations to each element.

“With the single source of truth approach to data models, we have raised our data quality, reduced implementation errors, reduced costs for solution maintenance and improvements, and achieved a quicker time to market for various functional, technical and legal changes. I whole-heartedly appreciate Sven for designing the robust concept and taking the lead in developing this fully scalable solution.”

Bharath Kolli,
Solutions Architect & Innovation Leader, global retail company

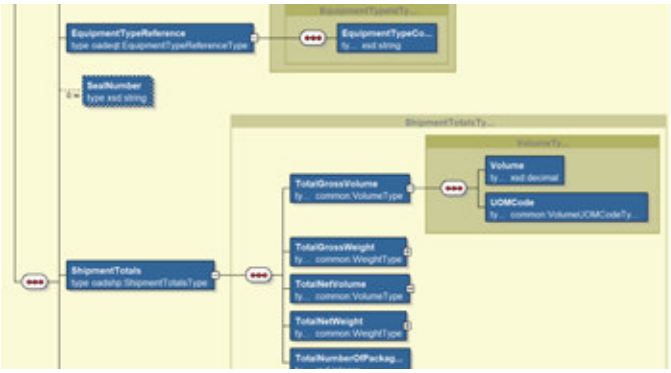


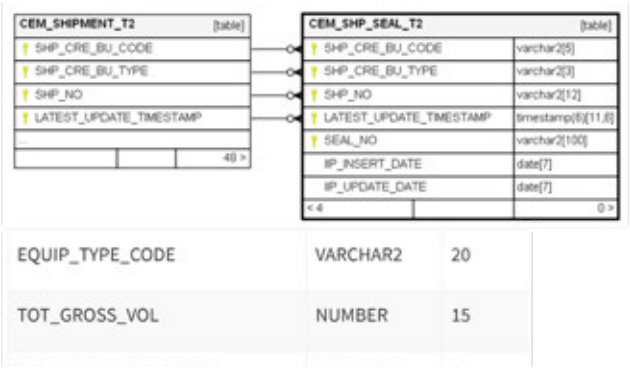
The relational database physical model

A substantial part of the IT landscape is custom-built using relational database technology. These legacy systems expect the data to be replicated into database tables. The solution architects are used to mapping from a common exchange format to their respective target formats, but they need a well-defined and documented standard logical model for this.

We have decided to derive the DB model from XML Schema in an algorithmic manner. As described above, attempts to achieve the fully automatic mapping of all possible expressions do not lead to a useable and understandable model. The design goal was to create a DB model that has a clear relation to the logical model and is easy to understand. This design goal was considered more important than achieving the most concise database model possible.

The algorithm is based on the prerequisite that XML Schema describes a single hierarchical business entity. The root element becomes the root table in a table hierarchy that adds elements as columns, unless an element repeats. In this case, it becomes a sub-table. This process is repeated until all elements of the schema have been processed.





EQUIP_TYPE_CODE	VARCHAR2	20
TOT_GROSS_VOL	NUMBER	15

To avoid manual work, the above algorithm has been implemented in a custom Java program that reads the XML Schema file and outputs the database table definitions as DDL. These are then used to generate the schema in a database.

The column and table named are also derived in an algorithmic manner, but are helped by custom attributes in the XML Schema file. Custom attributes also relieve the program of the burden of "guessing" the SQL data types. In this way, the architects still have sufficient control over the physical database model, but everything is kept together in the same source.

In addition, mapping sheets in Excel format are produced to ease the work of the solution architects who map this to their target system format.

The JSON schema physical model

With the XML Schema and database physical models in place for several years, it has become necessary to also provide the same in JSON format for integration to cloud applications. JSON Schema is significantly different in structure from XML Schema, and various existing tools for creating JSON Schema from XML Schema have delivered less than ideal results: The produced JSON Schemas were cluttered with unneeded IDs, and were often too lenient in validation.

So again a custom tool was developed that algorithmically translates the XML Schema into JSON Schema. Since the input schema is known to be constrained by the modeling guidelines, a straightforward algorithm could be developed:

- Standard XSD types are translated into standard JSON Schema types
- Custom XSD types with validations are translated into JSON Schema definitions with the same validations
- Single elements become JSON object properties
- Repeatable elements become arrays of JSON objects
- The business annotation becomes the description

```
            "INCOTerm": {
                "type": "string",
                "description": "Terms of delivery are the international
trade rules that clarify the responsibilities concerning cost and risk
between a buyer and a seller. INCO terms are a subset of these regulations."
```

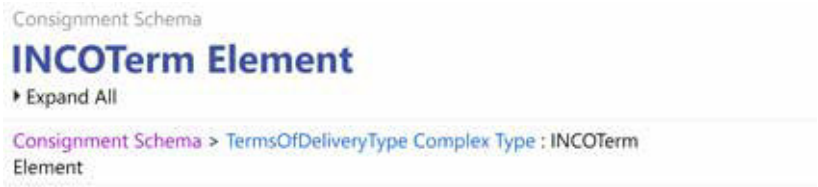
As it is best practice for JSON Schemas to include examples, various toolsets such as Apiary.io are able to generate integration tests automatically. To support this, a custom attribute has been added to the logical model XML Schema so that meaningful examples can be provided by the business subject matter experts in the one place they are already familiar working with.

Future physical models

As the JSON Schema example has shown, the approach supports the addition of new physical models when the need arises, even on a model that has been in production for years. This ensures that the investment in the logical model is protected. Apache Avro is expected to play a role in future integrations. When it is time to provide the business data in that format, the program will generate the describing schema from the same source, which is now the fourth option.

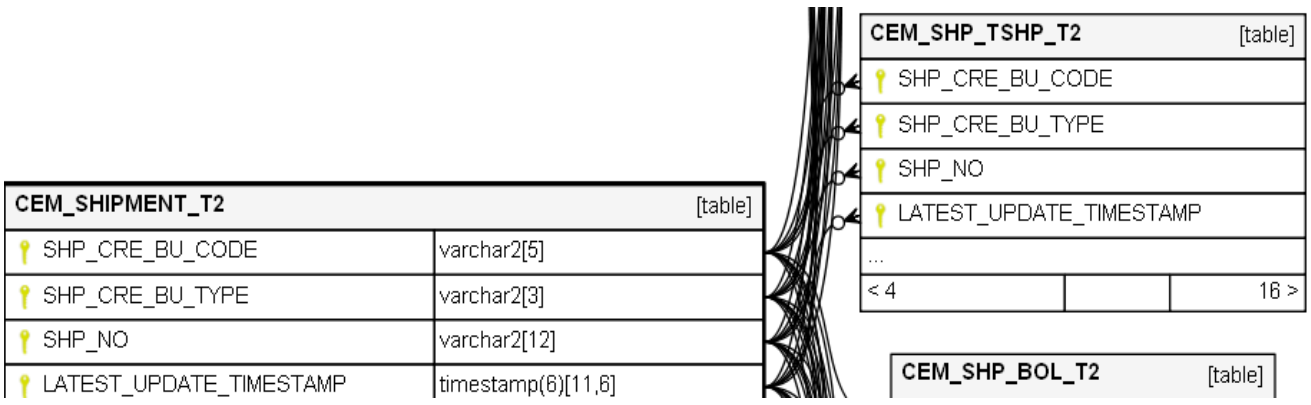
The documentation website

The graphical representation of the logical model is presented together with the business description from the annotations on an intranet page that can be freely accessed by all stakeholders within the enterprise. As well as being a resource that is actively used by the expected audience of architects, implementers and testers, it is also frequently accessed by the IT production team when investigating incidents of the wrong data being received, and is even used as a glossary of specific business terms when training application end users.



We have chosen Innovasys DocumentX! to read the XML Schema and generate the documentation in a fully automated manner. As XML Schema is a standard format, alternative tools could be used if the need arises.

In addition to the XML Schema logical model, which is also one of the physical models, the intranet site contains a generated documentation of the database physical model. This is obtained by creating a database schema based on the generated DDL scripts, and then documenting this schema using the SchemaSpy open source component. Again, as the source is a database, other tools can be used as an alternative.



Documenting the actual database schema instead of inspecting the DDL files ensures that any errors in executing them are captured early.

The JSON schemata are currently only available as plain schema files for download and use, as their structure resembles the XML Schema structure which is already documented including a graphical representation.

Tying it all together

As stated above, an automated approach is key to ensure that the model is kept up to date, even when faced with time and budget constraints.

A continuous integration approach that is used for software development can also be adopted for model generation.

The XML Schema files for the logical model are committed to a version management tool (git), in branches for current production and for future development.

A build server (Jenkins) picks up the changes and runs the following tasks:

- XML Schema documentation as website (Document! X)
- Generation of database model DDL scripts (custom Java program)

- Execution of database model DDL scripts (SQL Plus)
- Documentation of database schema as website (SchemaSpy)
- Generation of Excel mapping sheets for database model (custom Java program)
- Generation of JSON Schema (custom Java program)
- Posting of all of the above to an intranet page
- Notification of changes to registered stakeholders

Conclusion

The challenge of defining, documenting and maintaining diverse models for enterprise data exchange can be substantially eased by enabling the logical model to drive the various physical models in an automated fashion.

Key benefits are the faster time to market, a better understanding across all stakeholders and functions, and less room for error.

The integrated modeling approach might not work for all conceivable specialties of business data, but is viable for a large proportion of real-life scenarios if accompanied by practical guidelines and solid toolset implementation.

If you are interested in learning more and discussing your organization's path to integrated data modeling, please contact us at info@cgi.com.

About CGI

Insights you can act on

Founded in 1976, CGI is among the largest IT and business consulting services firms in the world.

We are insights-driven and outcomes-based to help accelerate returns on your investments. Across hundreds of locations worldwide, we provide comprehensive, scalable and sustainable IT and business consulting services that are informed globally and delivered locally.

cgi.com