Balancing Simplicity and Expressiveness: Designing Domain-Specific Models for the Reinsurance Industry

Hans Wegener*

1 October 2004

Abstract

Swiss Re has established concepts and tools for specifying and evolving (segmentation) models of its business. Based on a "to be" model, "as is" models are integrated using integrity constraints between them. All these models are founded upon business terminology. We report on two specific practical problems and our solutions to them, namely avoiding branching in change management (through use of temporal selection) and using soft constraints to avoid additional model elements.

1 Introduction

A reinsurance company, or reinsurer, insures insurance companies, also called (primary) insurers. Much like insurers assume risk for a fixed price from their clients, either persons or organizations, reinsurers assume risk from primary insurers. Depending on their needs, primary insurers mainly buy reinsurance to either assume a larger portfolio of similar small risks, and/or to pass on parts of a specific very large risk. For further information on the subject of reinsurance, we refer the reader to [1].

While both insurers and reinsurers assume risk, they are different in that a primary insurer is a retailer with a relatively large customer base and relatively few highly standardized products, while a reinsurer is a wholesaler with a relatively small customer base and relatively large number of products specifically designed for one client. As a result, the volume of data and transactions is not nearly as much of a challenge for a reinsurer as it is for a primary insurer, or a retail bank, for that matter. On the other hand, given product diversity, the size (and therefore

*Swiss Re, Mythenquai 50/60, CH-8022 Zürich, Hans_Wegener@swissre.com

clout) of many clients, and the global reach of the business, standardization is much more difficult on the data side as well.

The nature of reinsurance business, the existence of long-standing legacy systems, and the absence of simplification alternatives made it practically impossible to establish a joint, unifying architecture upon which both existing and future applications may be built. Hence, lacking a viable architecture upon which to *construct* them, an architecture with which to *integrate* systems had to be sought instead. Therefore, our goal was threefold:

1. Establish and manage a consistent model of the business and its financial status to be specified by and communicated to reinsurance and finance domain professionals.
2. Implement applications based on a stepwise refinement of the above business model and support their integration by mapping the (implementation) models onto each other.
3. Decouple applications in a way that their interfaces prove stable under evolution of the above business model.

Domain-specific models are intended to increase the proximity between specification and implementation or, in a way, narrow the communicative gap between engineering and business. In order to do so, business-terminology takes on a more central role in modeling. By doing so, however, modeling takes on more ambiguity for the engineer, due to the less formal use of language in business settings. On the other hand, business users are faced with the unfamiliar rigidity of formal languages. In this article we report about our experiences with a modeling language for reinsurance business and financial accounting. More specifically, we explain our experiences with balancing simplicity and expressiveness to cater for the needs of business non-specialists unaccustomed to the world of formal languages.

2 Specification, Integration, and Evolution

2.1 Architectural Perspectives on the Business

In this article we use the terminology of the Model Driven Architecture (MDA), as defined by the Object Management Group (OMG), to describe and delineate architectural concepts [3]. Generally, we distinguish between three models:

- a computation-independent model (CIM),
- a platform-independent model (PIM), and
- a platform-specific model (PSM).

The CIM is used to represent concepts important to business. These include, for example, the general ledger, risk model figure aggregation rules, carrier structure, or customer segmentation. The PIM is used to represent a refined application perspective on the business. This comprises, for example, business processes, change management, or access control. The PSM is used to represent the technology substrate applications are directly implemented upon. This includes, for example, user interfaces, relational databases, or messaging middleware.

For each of the above (CIM, PIM, and PSM), our company provides standardized artifacts to be used in design and development. Three different corporate teams are responsible for governing architectural practice regarding each model (information and process architecture, application architecture, and technology architecture). Mandatory project milestones are defined to check project artifacts produced for architectural compliance.

2.2 "To Be" and "As Is" Models

In theory one could discard the CIM and just live with PIM and PSM. However, the distinction between CIM and PIM is important to us. To business, the concept of change is a remote one. Anecdotal evidence suggested to us early on that one has to separate between the way business people see current goings-on and their perspective on history. Especially in non-life reinsurance and financial accounting, where complexity is high and—literally—decades of business matter, this distinction is of crucial importance. Furthermore, as any bigger company in the financial services industry, we are plagued by legacy applications, which is a substantial investment not easily done away with.

To achieve a clean separation of concerns, the CIM deals as a carrier of the "to be" world, while the PIM is the place where "as is" abstractions and, most importantly, change are dealt with. The CIM is mapped to the PIM by a transformation mechanism to be detailed in Section 2.3. But there is not one single PIM, there are—literally—dozens. In addition to history and legacy systems, the global nature of our business adds jurisdiction to the equation. From the perspective of the corporate headquarters, this is not a relevant abstraction, either. In fact, Swiss Re operates numerous accounting systems for closing the local books of its legal carriers, plus one for the entire firm that integrates data from all these. Therefore, PIMs abound in that there is plenty of business abstractions in individual PIMs that in theory belong to CIM. Notably, this is less an effect of bad architectural management than of the intrinsic complexity of the business. As a result, the CIM abstractions are moderately complex and not numerous, but the number of PIMs is substantial and thus considerably increases complexity. Architectural artifacts are thus specified at different levels of abstraction, with different models in mind.

In order to manage them concurrently, our integration architecture had to handle three separate goals:

1. Define, structure, and declare constraints between business abstractions in a model. We designed a tool enabling business users to model them on the basis of business terminology. Thus they establish a CIM dubbed the *reference* model. The tool repository is fully historized, providing version-based as well as temporal access to its content.
2. Allow for variations of the business model in the way it is implemented. For different applications, the structure can be manipulated, detail taken away and adorned with technical representations of business abstractions. The resulting PIM is a *context-specific* model of the reference model. This is, again, provided by the mentioned modeling tool.
3. Provide mappings between different models based on their history or other metadata to enable application integration and model evolution. PIMs can be mapped to other PIMs only with reference to the CIM. We designed a second tool for mapping between different PIMs that makes use of the CIM for purposes of disambiguiation (of history).

The division of labor between historization, mapping, and integration was chosen for different reasons. Legacy applications have little choice but to operate on the premises they were designed upon. Therefore, the choice of change adoption has to reside with the application integrating others. Separating between historization and mapping enables us to avoid CIM bloat.

2.3 CIM and PIM Abstractions

Our language for creating business models is the Swiss Re Data Language (SDL). As the name suggests, it is primarily designed to support modeling of content rather than behaviour. The SDL Tool is our repository and modeling environment that supports users in designing and understanding the structure of the business based on the SDL, which has a history of four years. The most recent major release went productive in June 2003.

In the SDL, reinsurance and/or finance concepts such as type of claim, currency, account, industry segment, or investment category, are described by *business terms*. Each business term exhibits a descriptive name (evoking its meaning), and a formal definition (denoting its meaning). Name and definition are used to specify and/or understand the business semantics behind concepts. Depending on their context of use, business terms describe attribute types, attribute values, or entity types. They belong to the CIM.

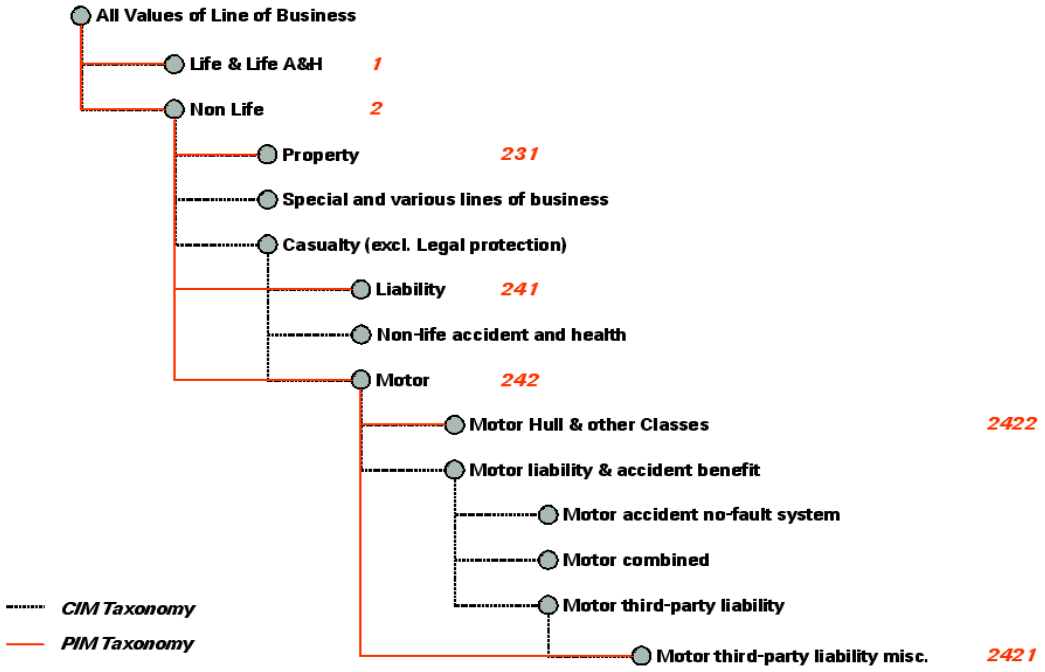

Figure 1: The line of business (extract) as seen from the corporate center perspective ("to be") and from the financial accounting point of view ("as is"). The numbers are the codes as they are used in the latter's context.

Business terms form the terminal symbols in SDL. They are then structured, in our case hierarchically to form *reference trees*. A reference tree captures the segmentation of the business (into specific and generic concepts) as it should be from the corporate center perspective. Reference trees are represented by an attribute type and group together attribute values. They belong to the CIM.

Business terms can also be used to represent entity types. Because it emanated from the financial accounting world of our company, at this moment SDL only supports hierarchical relationships between entities (as in the general ledger). (It is considered to extend this model to more flexible use of relationships as in entity-relationship modeling.) The resulting structure, represented by an attribute type, is referred to as a *custom tree*. The values of the attribute type representing the custom tree act as entity types, which in turn compose a number of attribute types together. The resulting model element forms, again, part of the CIM.

Our carrier for forward-engineering PIMs out of CIMs is the *context*. A context can be anything from an application (which it normally is) to an industry standard

to a mere illustration. It serves two separate purposes. First, it is used to associate business terms with a context-specific representation, the *codes*. Second, it is the platform for modeling context-specific variants of a reference tree, the *filtered tree*. As the name suggests, a filtered tree contains a subset of the values of the reference tree it is based upon. Furthermore, the taxonomic relationships between the values are congruent with that of the reference tree in that a parent of a value in the reference tree must also be its parent in the filtered tree, as long as it also occurs there. As such, context are the place where metadata about structural mappings between CIM and PIMs resides.

PIMs are administratively separate from the CIM, but there is a wide number of interdependencies. In order to minimize the potential harm that can be caused the SDL Tool checks overall repository consistency (CIM and PIMs). *Hard constraints* guard against violations of repository integrity, while *soft constraints* merely provide hints at potential inconsistencies outside the realm of the SDL Tool proper.

3 Simplicity vs. Expressiveness

3.1 Temporal Selection Instead of Branching

The CIM is defined as the model relevant to reflecting changes in the understanding of the business. Therefore, all CIM elements are versioned to the end of managing their lifecycle. Different changes can occur to the structure of a reference tree. Attribute values can be added, removed, moved, promoted, demoted, shifted, merged, or split. Business terms themselves can be changed as well, notably their definition or name. Business term versions are valid starting at some point in time, possibly becoming invalid at another. At any given time, at most one version (and with it the associated structures like reference or custom trees) is valid. Names are not identifying properties of business terms (homonyms do occur), hence we use a unique identification number—the SDL-ID—to unambiguously identify them.

The SDL Tool does not support branching. Instead, temporal selection is the mechanism for choosing repository object versions belonging to a specific configuration. The definition of the *valid version* for a repository object is as follows:

1. its validity started in the past (valid from smaller than now),
2. it has the maximum version number of the above set, and
3. it is still valid (valid until bigger than now).

The reason for this design is an important business concept, *invalidation*. Invalid repository objects are considered unfit for use in business. (More about that in a minute.) If, for example, the version from the set of all object versions valid in the

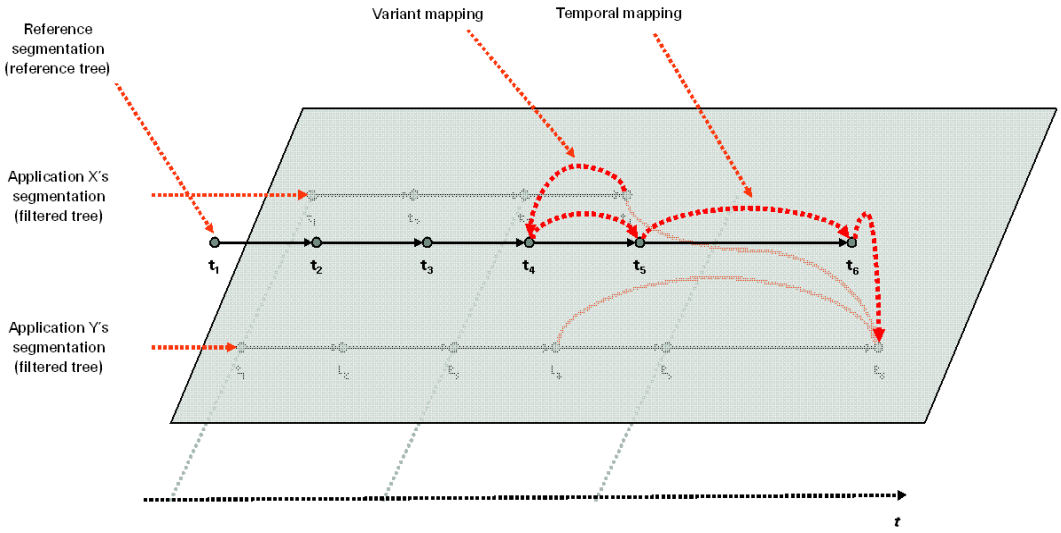

Figure 2: Variant (PIM) integration via the reference (CIM) and across time as seen at Swiss Re. The filtered trees can be easily mapped to the reference tree and back using context-specific metadata. Temporal mappings are based on successor information. If two applications X and Y want to integrate, they first map from the source application's PIM to the CIM, then across time and back to the target application's PIM.

past is no longer valid, then by our above definition the object itself is no longer valid (i.e., there exists no valid version).

As there is at most one valid version, the worst that can happen is that no valid version exists at interface calls (or exchanges). This raises the issue of handling CIM elements that are no longer considered valid. To deal with this, we make a distinction between the lifecycle of business terms and the structures they are part of. Any change to the structure of a reference tree leads to a new version; the reference tree itself may only be valid or invalid. If it is invalid, it is no longer considered part of the CIM. On the other hand, a value may posess a third state. If it is invalid, but part of a valid reference tree, it is considered deprecated (cf. Table 1).

The distinction between business term version history and the term's current status gives way to an (in our opinion) elegant architectural decision we took with respect to how applications integrate and react to evolution of the CIM. Generally, there are three main players in our integration architecture:

1. a number of applications (OLTP, ODS, DWH, or OLAP), designed for their

Value	Reference Tree	Status
Valid	Valid	Valid
Invalid	Valid	Deprecated
Valid	Invalid	Invalid
Invalid	Invalid	Invalid

Table 1: The state of a business term is determined by its own validity, the validity of the tree it is or has been part of, and whether it still is currently part of it. If it is not part of the currently valid reference tree, the value is generally invalid (with respect to that tree).

own PIM,

2. one model repository, containing the CIM and all PIMs, and
3. one mapping repository, with PIM-to-PIM mappings across a PIM's history and between different PIMs.

Applications use the model repository to store the PIM underlying its own design. The model repository is used to design the CIM, version n, and forward-engineer the PIMs, version n for context version m, from it. If a CIM version $n+1$ occurs, the PIMs version $n+1$ for context version m can be generated based on available metadata. If a local change occurs, a single PIM version n for context version $m+1$ can be created, leaving all other PIMs unaffected. This way it is possible to keep applications in sync with the CIM.

Due to the structural integrity constraints, a PIM-to-CIM mapping is much more simple than mapping between different (often consecutive) versions of a PIM, or even worse, between different PIMs that are structurally only loosely correlated. As for the mapping of different versions of a PIM, ambiguities can occur, for example values can exhibit more than one parent value in the filtered trees they belong to. (It may even happen in reference trees.) Here, manual intervention is required. The metadata provided for mapping a PIM between different versions is stored in the mapping repository and used by applications. The model repository is uninterested in it. The same holds for mapping between PIMs, where ambiguities can occur as well, for example values existing in one PIM's filtered tree but not the other's. (Note that both PIMs are understood to refer to the same CIM version, for which reason there no history-related ambiguities occur.)

As far as the lifecycle of attribute values is concerned, mappings are provided for values that are truly invalid and have a known *successor*. Applications are required to map them, especially at inbound interfaces. If no mapping is available, the value

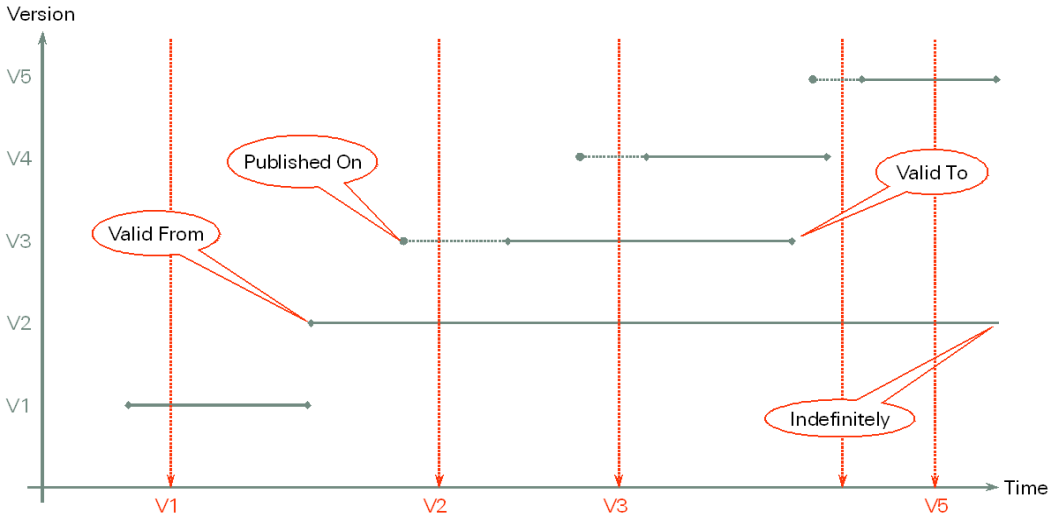

Figure 3: Versions and time in SDL. The horizontal axis show the versions valid as of a particular point in time.

must be rejected. As for deprecated values, the question when to make a deprecated value truly invalid is a business decision. Hence, mappings may be provided or not, and applications may use them or not. More specifically, some applications (notably OLTP systems with strong auditability requirements) may decide to never change the original values associated with some stored fact. However, the assumption is that for most systems, the deprecation period is not indefinite, especially in management reporting.

The welcome side-effect of such an architecture is that it allows for the automatic handling of PIM-to-CIM mappings (and back to a PIM again). Given that we are driving towards this goal, this is important to us. At the same time, there is a wide range of tools at the discretion of applications to integrate with other applications one the one side, and to extend or shorten their own adoption of the latest CIM version on the other side. If, however, they decide to always stick with the current CIM version, the complexity they are exposed to is tangibly reduced.

Temporal selection has possible unwelcome side-effects. For example, the idea that any repository object exhibits at most one valid version may be inappropriate for some settings. However, we have as of yet not encountered any such situation, because by the very definition of it, the CIM is *the* "to be" model. If anyone requires his/her own model, the PIM can be used to combine a context with the reference to create a PIM. Since contexts exhibit validites as well and filtered trees are combined

from reference trees and contexts, the additional degree of freedom is obtained.

3.2 Soft Constraints Instead of Additional Model Elements

Our use of soft constraints was caused by the need to accomodate conflicting needs in the design of the CIM (and, as an effect, the PIMs). Many of the constraints are not generic to the entire domain. We had three choices:

1. express this in the form of additional CIM elements, thus adding to model complexity,
2. express this in only a few PIM elements, thus abandoning parts of our CIM-centric model,
3. not express this in any model, thus allowing for inconsistencies in some PIMs or the CIM.

We opted for choice three, because the other options were expressly rejected by business users as "too complex." Thus we ended up with a CIM and PIMs that were not expressive enough to model all the concepts existing in the real world. SDL Tool administrators are merely warned of (not prohibited from causing) inconsistencies. However, we did not yet run into any serious problems. Little (if any at all) confusion was caused by the missing model elements. It turned out that in the specific context of use there was so much redundant (i.e., tacit) knowledge available that the lack of expressed constraints did not really make much of a difference for model consistency. Expressing specific needs in the CIM can have a detrimental effect on other parties, while not adding critical capabilities for the other. This makes the CIM much less usable for many while not even really helping the remaining few. One can exploit organizational correlations between IT and business by leaving out model elements (i.e., non-generic business concepts), thus increasing the *suitability for the task* of model(s) for business users. For example, the homonym rate in model repository was thought to be a strong indicator of glossary intelligibility. However, in financial accounting we found that our request to eliminate homonyms was rejected because it was not percieved as useful.

A consequence of this kind of modeling are consistency issues. If the modeling environment is no longer capable of checking consistency, the model is incomplete with regards to the problem at hand. We have used different techniques to deal with the resulting design space (cf. [4]), for example by having interfaces deal with additional repository object states in change management (see below).

Along with the different version selection mechanisms of the SDL Tool (temporal or version-based), we have begun to make application interfaces first-class abstractions in the PIMs and use mechanisms similar to profiles in the Unified Modeling

Language (UML) to further automate handling inconsistencies (and, besides, mappings). Applications will specify their change adoption model (e.g., real-time, delayed, none), involved PIMs, and one is capable of (semi-automatically) providing PIM-to-PIM mappings, even across different referred-to CIM versions.

4 Position Statement

Based on our experience, we claim that soft constraints (i.e., warnings instead of errors) are indispensible and should become an intrinsic part of domain-specific modeling languages. Compiler builders have long and successfully employed this technique. This will at the same time reduce complexity for the language user, while not overly compromising consistency.

Secondly, in our experience validity and jurisdiction (not covered here) are indispensible abstractions to be used in a globalized financial services company. So far—in our taste—especially validities play an underrepresented role. The fact that some configuration-management problems can be (halfway) solved with time as a first-class repository abstraction speaks for it.

Third, in any larger environment dealing with legacy, inconsistency is an unwelcome, but crucial issue to deal with. Outside pure forward-engineering settings, consistency management must become part of engineering practice. This affects modeling languages (see above), tools, and the development process.

Acknowledgements

The two introductory paragraphs on reinsurance business are an (almost) verbatim copy from [2]; used with permission.

References

[1] R.L. Carter. *Reinsurance: the Definitive Industry Textbook*. Euromoney Institutional Investor PLC, 2000.

[2] Robert Marti. Information integration in a global enterprise – some experiences from a financial services company. In Gerhard Weikum, Harald Schöning, and Erhard Rahm, editors, *BTW 2003, Datenbanksysteme für Business, Technologie und Web*, volume 26 of *Lecture Notes in Informatics (LNI)*, pages 558–567. Springer, February 2003.

[3] Joaquin Miller and Jishnu Mukerji. *MDA Guide*. Object Management Group, June 2003.

[4] Hans Wegener. Generative programming and incompleteness. In Krzysztof Czarnecki, editor, *OOPSLA Workshop on Generative Programming*, October 2001.