Experience in Applying Conceptual Modeling
to Interface with a Real-Life Business Application

Dr. Michel Pilote
University of Toronto
Toronto, Canada

Abstract

Conceptual Modeling techniques have been primarily applied for requirement and design specifications. These concepts can also help to understand the characteristics of mature large scale applications. We have used these ideas to design and implement an Insurance Underwriting Expert System around an existing mainframe program, driven by a sizable old fashioned data base. We describe how Conceptual helped, not only to identify real user needs, but also to untangle many accumulated complications in the existing repository of information. In turn, this case study highlights which Conceptual Modeling mechanisms are most needed to support such tasks.

1. Introduction

The field of Conceptual Modeling (CM) has evolved out of pioneer work published in the late seventies [1, 4, 5, 15]. Although perhaps most acknowledged in the database (DB) community, this new area also attempts to integrate many research results from the fields of Artificial Intelligence (AI) and Programming Languages (PL). This convergence is recognized in a recent International Conference [7]. Significant projects exploring this combination include DAPLEX [14], TAXIS [9], GALILEO [2], GEM [10], POSTGRES [13], SIM [6], and EXODUS [3]. In a Ph.D. thesis [10], we investigated how we could use Conceptual Modeling to define and support linguistic interfaces to a knowledge base constructed with these same tools. This led us to analyze what is left of such a system once you remove all syntactic covers. This research pointed out a few basic mechanisms which appeared to underly the intuitive appeal of Conceptual Modeling techniques. Since then, we have been testing and refining these theoretical ideas against real-life situations [11]. We report some insights derived from applying this approach to extend a traditional mainframe-based application with an Expert System component.

We were asked to help design an Automated Risk Screening system for Personal Lines policies in one of the largest Canadian general insurance companies. This is an application that is attracting a lot of interest in most insurance organizations. Yet previous attempts at applying traditional software methodologies have proven very disappointing and frustrating on many occasions already. Often, "system" people are simply unable to get their users to express what they would need. On a few occasions, a complete system has finally been put together, yet remains unprofitable. For some of these, we hear that the unpredictability and unliability of the data made the program unusable. For the remaining others, users are dutifully feeding this new beast, in addition to handling their previous tasks. It produces copious reports. But nobody does anything with this output. People just feel overwhelmed by the quantity of information. The net effect is more work for no real gain.

This task shows all the characteristics of these Interactive Information Systems (IIS) for which Conceptual Modeling mechanisms are meant to be so beneficial [8]. Locally, all processing is very simple if not trivial. The complexity comes from a very large number of details and exceptions to account for. The reality was even more complex than it appeared at first. When we looked at the operations of the existing insurance system, we found that nobody fully understood all the special cases that were occurring. Many of these were even a surprise for experienced users. Worse, as we observed, the interpretation of some of these phenomena kept changing!

We discuss elsewhere [12] the characteristics of the rule-based Expert System that we developed to successfully handle the user requirements. This paper focuses on the design and implementation of the interface between this AI component and the existing mainframe program and data base. We describe which Conceptual Modeling mechanisms were most useful to achieve this effect. We detail in particular how these techniques helped to classify and understand the many deviations that surfaced amongst incoming data from the mainframe repository. We needed to model, not only the "conceptual" level as seen by users and designers, but also the "physical" level of the data actually stored on their machine. We explain how we used this platform to reconcile these two points of view. We also summarize how we implemented these components in order to leave the company with a system that has already been in "production" for some times.

2. Overview of our Approach

CM techniques greatly helped to produce a working solution, but often quite differently from the way CM is normally applied. Conventional CM ideas primarily contributed to guide and support our project towards effective results. However, we ended up with a much simpler setup in order to leave our users and their technical staff with a fully operational system. We summarize in Figure 1 the relationships between the main components of our solution.

![Figure 1: Main Components of our Solution](image)

The Conceptual Model (la) results from modeling our understanding of the user requirements. The Physical Model (2a) comes from applying the same CM techniques to define and classify the information extracted from the existing mainframe application (3b). The Initial design (3a) of this system documents in particular the characteristics of the data used by this program. We found these specifications frequently out-of-date, if not simply missing. Experienced programmers would generally work directly with the data used by the existing mainframe system rather than consult these write-ups. Our "implementation" (2b) of the physical model constitutes the interface between the incoming data from the mainframe system and the rule-based Expert System. We expected initially to end up with a corresponding implementation (1b) of the Conceptual model as support for the user rules. It would have been quite feasible to convert the incoming raw data to fit this format. To our surprise, users and programmers alike felt more comfortable with the "physical" model than with the more abstract "conceptual" one. This reaction is understandable when coming from programmers, who already think in terms of what they are used to work directly with. How can we explain this similar reaction from users? It might be that their existing interface with the mainframe system had...
already trained them to think in those very pragmatic terms.

This Conceptual model (1a) was nevertheless essential to support our own efforts at classifying the special cases occurring and reconciling partial views with the full detail of the existing program. This level is also useful to explain these distinctions in this paper. But it is interesting to note that it has been removed from the on-going system now in regular use in that company.

We tried in this work to use as few CM mechanisms, in their most simplified form, as sufficient to represent the necessary distinctions. We ended up with three main mechanisms: instances, properties, and partitions. These three devices are respectively inspired by the three major abstraction mechanisms typically provided by a CM formalism: classification, aggregation, and specialization. Objects, also called entities, are grouped into categories of "similar" objects by the instance relationship. This device allows to "classify" the domain of discourse into a number of "classes".

Concepts are knits into a whole system through "properties". They support the aggregation of parts into higher level concepts. Our "partitions" represent a restricted form of the more general "ISA" mechanism usually found in CM formalisms. They allow the specialization of concepts into subconcepts.

2.1. Notations and Conventions

The CM language that we use in this paper is a simplified adaptation of the constructs generally found in other similar work. We successively present the formats for defining version of each of the three major CM abstraction mechanisms.

Classification: Concepts and Instances Definitions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Conceptual Model</th>
<th>Physical Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;MetaConcept&gt; &lt;ConceptName&gt;</td>
<td>CONCEPT POLICY</td>
<td>RECORD 02</td>
</tr>
<tr>
<td>&lt;ConceptName&gt; &lt;InstanceId&gt;</td>
<td>POLICY 5614523</td>
<td>02 #4442</td>
</tr>
</tbody>
</table>

A new concept is introduced by the name of a "meta-concept" followed by the "name" of this new type of objects. This new concept is considered a new instance of the indicated meta-concept. In the "Conceptual" model, all concepts are instances of the meta-concept CONCEPT. The "Physical" model involves two types of meta-concepts: RECORD and PART. The first of these "Physical" concepts models the familiar basic disk storage structure. The second represents a "group" of these records, which together contains some related information.

An instance description starts with a concept name followed by an identifier for this object. For a "Conceptual" object, this identifier is the value of a selected property of this instance. Our naming of a "Physical" entity deserves a few words of caution. At any point in time, a data file extracted from the mainframe database contains a fixed number of records in a well-defined sequence. This gives us a relative position for each record in a particular data file. We use this existing property of a data record to construct a unique means to identify each record of each type. We write for example "02 #4442" to stand for the 4442

A double headed arrow from A to B indicates that there is a link from A to B. An arrow from concept A to concept B indicates a relationship. This device allows to "classify" the domain of discourse into a number of "classes".

Aggregation: Definitional and Factual Properties

<table>
<thead>
<tr>
<th>Concept Definition</th>
<th>Instance Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;MetaConcept&gt; &lt;ConceptName&gt;</td>
<td>&lt;ConceptName&gt; &lt;InstanceId&gt;</td>
</tr>
<tr>
<td>(propertyCategory)</td>
<td>(property) = value</td>
</tr>
<tr>
<td>(Definitions)</td>
<td></td>
</tr>
</tbody>
</table>

A property shown as part of a concept definition indicates the range of acceptable values for this property when applied to instances of this concept. Properties on a concept can be grouped into property categories to help distinguish related types of information. We will be using the following property categories:

- id: a property whose value can be used to name an instance
- links: a relationship to another user-defined concept
- parts: a property ranging over one of the built-in definitions
- partitions: a property used to "partition" a concept into many specializations

A concept definition can include many groups of properties, each made of a property category, and a list of property definitions. An instance description is restricted to a list of property-value pairs. Range definitions can be any built-in or user-defined concept. Built-in concepts include the familiar types STRING, INTEGER, CHARACTER and BOOLEAN. These distinctions were useful for implementation efficiency. In addition, in the "Conceptual" model, we also used the ranges DATE and DOLLAR to indicate special display formats. DATE instances would show as MM/DD/YY, while DOLLAR instances would append $ to the appropriate digits, and with a dot before the last two digits. We distinguish multi-valued properties by adding the suffix '.LIST' to their definition.

Note that links and parts are simply two arbitrary labels reflecting the widely used distinction in CM formalisms between entity-valued and data-valued properties respectively. Examples of this nature include data-valued (DVA) vs entity-valued (EVA) attributes in [6], or attributes in the Entity-Relationship model [4] vs binary relationships between classes. The range of all our data-valued properties are built-in concepts, while the range of all other entity-value properties are user-defined concepts. The role of a "partition" property is explained below.

Specialization: Explicit and Partition-induced

<table>
<thead>
<tr>
<th>Definitions</th>
<th>Induced Specialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;MetaConcept&gt; &lt;ConceptName&gt;</td>
<td>&lt;MetaConcept&gt; &lt;ValueN&gt;</td>
</tr>
<tr>
<td>&lt;ConceptName&gt;</td>
<td>partitions</td>
</tr>
<tr>
<td>(property) = &lt;Value1...ValueN&gt;</td>
<td>(&lt;property&gt;) = &lt;ValueN&gt;</td>
</tr>
</tbody>
</table>

Explicit "ISA" definitions can be followed by any property definitions. This general format is used only in our conceptual descriptions and is not part of our implementation. Instead, we heavily applied the "partition" properties to organize concepts instances into useful sub-categories. To use PASCAL terminology, this construct defines a property as a "discriminant" for a record. The right part of the diagram explains how an induced specialization could have been explicitly defined. It highlights how we identify each newly specialized concept with its discriminating value. These partitions are mutually exclusive for single-valued properties, but they can also overlap in the case of a multi-valued property. Other recent CM implementations also found it useful to introduce a similar mechanism. SIM (6), in particular, makes use of a subrole attribute on each class with subclasses, whose value is the "name" of each of these subclasses.

2.2. Graphical Notation

Throughout the paper, we will complement syntactical definitions following the above formats with more graphical depictions of the interrelationships between these concepts, according to the following conventions:

A → B: An arrow from concept A to concept B indicates that there is a link from A to B. A → B: A double headed arrow from A to B indicates that values of this link are lists of instances of B; A ↔ B: Mutual links are combined into bidirectional arrows; A ⇒ B: A double lined arrow depicts that A is a specialization of B.

For example the definitions:

<table>
<thead>
<tr>
<th>Concept POLICY</th>
<th>RECORD 02</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>links</td>
</tr>
<tr>
<td>policy_number: STRING</td>
<td>policy: POLICY</td>
</tr>
<tr>
<td>links</td>
<td>parts</td>
</tr>
<tr>
<td>02: 02_LIST</td>
<td>symbol: STRING</td>
</tr>
</tbody>
</table>
would be depicted as*

\[ \text{CONCEPT POLICY} \leftrightarrow \text{RECORD 02} \]

Their instances would be shown as

```
POLICY 5614523
02 = 46442
```

In all our definitions and examples, the case and style of characters is purely meant as aids to facilitate understanding. Our conventions reflect the role that an object is playing in a construct. It has no relation to the identification of the object. Bold characters are reserved for keywords, property categories and meta-concepts. Concept names are normally shown in capital letters. Values and instance identifiers are generally presented in italic. However, the same name can appear in different formats in related definitions. This will happen in particular with the name of specialized concepts induced by a partition, which are also the values of this property.

2.3. Project Organization

![Figure 2: Initial Set up and Information Flow](image)

(1) An individual would fill an application form requesting some Automobile or Homeowner coverage. (2) He would submit this form to his agent, who would in turn feed this information into the mainframe. (3) This new record would be added to a very large data base. (4) Overnight, the machine would process the case, print out a policy, and return it to the agent. (5) It would also verify a number of underwriting guidelines. Any violation of these rules would be reported on exception reports, regularly sent to human underwriters. (6) These specialists would appraise the discrepancy by consulting the computerized record of the policy. (7) If necessary, they would request to review the paper application as well. At this time, they could choose to accept or cancel the policy. (8) They could also request additional clarifications by phone from the agent before reaching their decision.

![Figure 3: Project Configuration](image)

(1) The policy data base would fill about twenty-six tapes. It was organized as one huge sequential file with multiple types of variable length records. A number of accessory indices would give rapid access along necessary dimensions. Two main ways were provided to interact with this data. (2) One would go through the prepared inquiry/data entry screens, one policy at a time, keyed by policy number and a few other qualifiers. (3) The other method involved running some selection utility on the tape copy of the whole data base.

To prototype a better solution, (4) we brought in a SUN 3/260 workstation, linked with the host computer. This Unix-based machine was equipped with 8Mb of main memory and 560Mb of disk storage. (5) The underwriter could use this new terminal to access mainframe records in the same way he was familiar with. (6) We could also transfer data for specific periods or agents, and run the screening module sequentially on each of the selected policies.

3. Conceptual and Physical Models

3.1. Conceptual Model

We summarize the components involved in the description of a policy, first at the conceptual level. A policy will have been "in force" for a number of "terms". Each of these terms represents a valid period of coverage from an effective date to an expiration date. An important distinction is to recognize whether a policy is for Automobile or Homeowner coverage. The screening rules are different for each kind. Many other types of information depend on this initial choice. This distinction is indicated by a symbol property on POLICY. Its possible values are Auto or Home.

![Figure 4: Conceptual View of Automobile Policies](image)

Figure 4 details all the components that we can find in an Automobile Insurance policy on this system. Some of this information is common to all types of policies. It is shown in Figure 4 as the concepts TERM and U/W, directly connected with the concept POLICY. Others are specific to the specialized concept AUTO_POLICy. These include DRIVER, VEHICLE and a specialization of the concept U/W meant for additional underwriting notes specific to an Automobile case. Each of these concepts is further explained below. We first focus on "putting the whole picture together".

The core of the concept POLICY can be defined as shown on the left of the following diagram. The right part describes an instance of POLICY which we will be using as an example throughout the paper.

```sql
CONCEPT POLICY
id policy_number: STRING
links
terms: TERM_LIST
underwriting: U/W
determine
symbol: Auto, Homeowner
rate: Discount, Regular
billing: Monthly, Annual
agency: AGENCY
POLICY 5614523
id = 12/06/84, 12/06/85, 12/06/86
underwriting = AUTO-UN DRIVER-UN
symbol = Auto
rate = Discount
billing = Annual
agency = Agency One
```

*) Generally, the meta-concept CONCEPT will be omitted from each diagram.
Appendix A contains a sample of what we had to cope with. It shows the idiosyncrasies, complications, and exceptions that we encountered when interfacing with the "real" data coming from the mainframe database. Appendix A contains a sample of what we had to cope with. It shows the idiosyncrasies, complications, and exceptions that we encountered when interfacing with the "real" data coming from the mainframe database.

3.2. Physical Model

CM techniques can also help to understand and organize the various idiosyncrasies, complications and exceptions that we encountered when interfacing with the "real" data coming from the mainframe database. Appendix A contains a sample of what we had to cope with. It shows the idiosyncrasies, complications, and exceptions that we encountered when interfacing with the "real" data coming from the mainframe database.

In "practice", users managed to greatly expand and innovate beyond what it first appeared. Sometimes users had significant changes to apply to a policy. At other times, they might get their entries too messed up. For these and other similar reasons, we observed that they would frequently just cancel the current module and create a new one. Thus the information about a particular term of a policy would end up spread over many modules. We would have to check the surrounding modules with similar effective and expiration dates. Moreover, these dates would not always be entirely reliable. Sometimes a new module would visibly serve to correct a mistake made about them.

Even keeping track of the "current" module was not necessarily trivial. Sometimes, the last entry would be canceled or lapsed and a previous module reactivated! Other times, the most recent module would have been filled automatically, in violation for a future renewal, and it still have to take effect at a later time.

4.2. Policy Rates and Billing

The insurer was offering different sets of rates for different types of risks. "Normal" rates would correspond to the most common cases. For a special category of "safer" situations, the company had introduced a "discount" type of policy with significantly lower fees. In both cases, rates could also vary depending on the territory in which an agent was allowed to accept business. The premium for each policy was computed according to some "rate tables" stored on the mainframe, specific to each agent and type of policy. A further monetary reduction could also be applied for annual pre-payment instead of monthly billing. These distinctions could be expressed as:

A policy is accepted for a "term" lasting six months or one year from its effective date to its expiration date. Each term is supposed to be represented by a different "module" in the data base. The first term would represent a "new" policy. All subsequent ones would stand for "renewals". This was part of the "theory" behind the design of the system.

Policy Rates and Billing

The insurer was offering different sets of rates for different types of risks. "Normal" rates would correspond to the most common cases. For a special category of "safer" situations, the company had introduced a "discount" type of policy with significantly lower fees. In both cases, rates could also vary depending on the territory in which an agent was allowed to accept business. The premium for each policy was computed according to some "rate tables" stored on the mainframe, specific to each agent and type of policy. A further monetary reduction could also be applied for annual pre-payment instead of monthly billing. These distinctions could be expressed as:

A policy is accepted for a "term" lasting six months or one year from its effective date to its expiration date. Each term is supposed to be represented by a different "module" in the data base. The first term would represent a "new" policy. All subsequent ones would stand for "renewals". This was part of the "theory" behind the design of the system.

In "practice", users managed to greatly expand and innovate beyond this foundation. We found the full reality quite more entangled than what it first appeared. Sometimes users had significant changes to apply to a policy. At other times, they might get their entries too messed up. For these and other similar reasons, we observed that they would frequently just cancel the current module and create a new one. Thus the information about a particular term of a policy would end up spread over many modules. We would have to check the surrounding modules with similar effective and expiration dates. Moreover, these dates would not always be entirely reliable. Sometimes a new module would visibly serve to correct a mistake made about them.

Even keeping track of the "current" module was not necessarily trivial. Sometimes, the last entry would be canceled or lapsed and a previous module reactivated! Other times, the most recent module would have been filled automatically, in violation for a future renewal, and it still have to take effect at a later time.

4.2. Policy Rates and Billing

The insurer was offering different sets of rates for different types of risks. "Normal" rates would correspond to the most common cases. For a special category of "safer" situations, the company had introduced a "discount" type of policy with significantly lower fees. In both cases, rates could also vary depending on the territory in which an agent was allowed to accept business. The premium for each policy was computed according to some "rate tables" stored on the mainframe, specific to each agent and type of policy. A further monetary reduction could also be applied for annual pre-payment instead of monthly billing. These distinctions could be expressed as:

A policy is accepted for a "term" lasting six months or one year from its effective date to its expiration date. Each term is supposed to be represented by a different "module" in the data base. The first term would represent a "new" policy. All subsequent ones would stand for "renewals". This was part of the "theory" behind the design of the system.
This was viewed as a secondary and temporary measure. However, when we looked at real cases, we found at times more than half of the incoming policies falling into this category. The notes identifying these occurrences were distributed at various places in the recorded information, in free-form format. We had to search each record for references to the discount offering, filtering out mentions of the same words occurring for other purposes, such as for street or city names in addresses.

In practice, all useful references to the discount companies could be found in records of type 03, 12, and 15. This is expressed by the reverse links to RECORD 02 in Figure 5. Together with the other necessary distinctions, we obtained the "physical" concepts defined as:

```
RECORD 02
partitions
pco: STRING
agency: STRING

with instances like
```

```
RECORD AGENCY_RECORD
partitions
name: STRING
code: STRING
address: STRING_LIST
rate: 11, 70, 12, 72, ...
```

The representation of these combinations shows many more complications. The `pco` field on RECORD 02 indicated both a rate level and the type of billing chosen. Codes 11 and 70 indicate monthly and annual billing respectively for normal cases. Codes 12 and 72 express the corresponding distinction for the lower rated discount policies. Other codes would also stand for other special arrangements. Finally, the agency field on RECORD 02 contains an agency number. Many numbers can be associated with a same agency. Each different number will be associated with a different set of rates.

4.3. Underwriting Information

The mainframe application was extended at some point by the company staff to record additional underwriting information. For users, this enhancement appeared as a new type of data entry/inquiry screen. The one corresponding to our example would look like:

```
POLICY ← U/W ← PREVIOUS_LOSS ← DRIVER_U/W
```

For example, we can find the instances:

```
U/W 5614523
previous_losses = NONE
drivers = Samuel Longman, Mina Longman
license_canceled = N
modified = N
changed_address = N
previous_insurer = PRUDENTIAL
previous_ins_policy_no = PA77012594
year = N
```

Physically, this part of the system might be the most "unusual". Each one of these screens is stored as is, including all headings, field labels and intermediary blank spaces. As shown in Appendix A, each underwriting screen is represented as a list of records of type 15, each containing two lines of the screen.

This information was regularly missing or misplaced. Because it was not essential to perform the rating and issuance of a policy, users in a rush would frequently postpone to fill this part. Of course, they would often forget to get back to it. In addition, when creating a new module, they would often "assume" that the system would copy the previous version of an underwriting screen into the latest module, although they had been told that the program had not been set up to do that. As a result, we would often find the underwriting information missing from the latest entry yet present with some earlier one. A number of times also, we bumped into modules with twenty-two records of type 15 instead of eleven as usual. Observation revealed that the first eleven of these lines would generally contain a "blank" screen, followed by a second screen, filled up this time.

We modeled the physical representation of this information as:

```
RECORD 02 ← PART 15P ← RECORD 15
```

```
POLICY ← U/W ← CURRENT_LOSS ← PREVIOUS_LOSS ← DRIVER_U/W
```

176
We took each group of eleven lines representing an underwriting screen as one single record. The multiple link from RECORD 02 to PART 35P accounts for the possibility of finding more than one group of eleven lines associated with a same policy record. The Implementation section of the paper explains how we handled 'NONE' as value of previous losses.

4.4. Driver Information

Conceptually, a policy involves a number of drivers. Some are principal operators for some vehicles. Others are only part-time operators. Each driver is associated with a number of "attributes" describing his driving experience. These relationships can be expressed as:

```
POLICY <-> DRIVER <-> ATTRIBUTE
```

For example, we have the instances:

```
DRIVER Samuel Longman
attributes = DTL, SST, MVRO, ...
policy = 5614523
age = 25
birth_date = 10/29/40
principal_operator_of = Plymouth Caravell

ATTRIBUTE DTL
id = 02
driver = Samuel Longman
date = 01/01/76

ATTRIBUTE SST
id = 35
driver = Samuel Longman
date = 02/02/86
```

Physically, as shown in Figure 5, there is one record 24 for each driver. Each of these entries contains space for up to sixteen "attributes". An attribute is made of a three- or four-letters code, and a six-digits date. For example, we would regularly see the codes DRA and FCA paired with the same date.

```
VEHICLE Plymouth Caravell
policy = 5614523
coverages = TPL, AB, Comp, Coll...
parts
year = 83
body_code = 4D
premium = 413.00
```

The main purpose of an Automobile insurance policy is to cover a number of vehicles against various possible risks. Instances of these concepts can be defined for examples as:

```
VEHICLE Plymouth Caravell
policy = 5614523
coverages = TPL, AB, Comp, Coll...
parts
year = 83
body_code = 4D
premium = 413.00
```

Physically, this information was encoded as two records of type 35 for each vehicle description. The first of these lines, identified by a 2 in position 6, would include many fields describing the vehicle. The second line, marked by a 3 instead, would include up to twenty "coverage" areas.

Each coverage group would include in particular space for a two-character code and a five-digit dollar amount. These codes would represent various types of coversages and deductibles. Again, we took each pair of consecutive 35 records as a single entry, which we modeled as the concept PART 35P. This information was thus defined as:

```
PART 35P
links
02 = 84442
35 = 13405, #13406
parts
description = Plymouth Caravell
body_code = 4D
premium = 41300
```

5. Implementation

Each set of policy records extracted from the mainframe data base would come as a large sequential file, typically containing between one and ten million bytes. Each series of records of various types representing a policy module would be buried somewhere in this file. To fit our definition of the physical information, we first needed to gain "direct access" to each module separately. This means being able to retrieve from a policy number and module, such as 5614523 and 03, the corresponding set of records. From each of these definitions, we also needed to be able to follow links defined with other components.

To this effect, we broke each original data file into a set of separate indexed files, one for each type of records. Each entry in the 02 file would represent one module of a policy. It would be linked to its corresponding entries in any of the files representing the other record types. Each of them in turn would point back to their parent record. "Indexed" in this case means that we could ask, say for the 4442nd entry in the 02 file, and obtain the 03 record for that 4442nd module. Its 03 link would point for example to entries 1580 and 1581 in the 03 file. In turn, each of these 03 entries would have 4442 as value of their 02 link. In this way, when matching the word "discount" in the 1580th line of the 03 file, we could infer that the 4442nd module was meant to be a discount policy, as we discussed previously in the example used throughout the paper. The following tables illustrate this indexing scheme.

```
vehicle = Plymouth Caravell
deductible = 50.00
code = 08
```

We implemented partitions in a particularly straightforward way. For any partition defined on a concept, such as in

```
record 02
```

```
record 03
```

177
We identified these instances with the corresponding values of the field property on AAA.

We used this device a lot in our implementation of the physical model. Any time we would partition RECORD 02 according to one of its fields, say pco, we would find in the "Physical" model a corresponding concept, which we might call PCO, whose instances were the corresponding values of the field pco on RECORD 02. One of these instances could then be described as

PCO 12 13 02
description = Discount Company, Monthly Billing
size = 32

Our program made heavy use of being able to attach additional attributes to these instances, like description and size in this case.

In particular, the previous description of the concept AGENCY_RECORD was handled in this way. It was first set up as one of those meta-concepts partitioning RECORD 02, according to its agency code field. We then added to this concept field for the full concept of the agency, its address, territory, and company codes (pco on RECORD 02) controlling its allowed rates.

We referred to this section earlier when encountering the previous losses attribute on the U/W concept. We needed to distinguish between no answer provided for this item, and an explicit indication of 'NONE'. In the first case, it could be because the agent had neglected to fill in this portion. The second was a clear indication of no previous loss. In the latter case, our program would pick up the entry containing the word 'NONE' instead of a loss date or code. We kept such an occurrence as a special token that would mark the appropriate distinction. It is not an instance of PREVIOUS_LOSS as such, conceptually, the nature of this object is unclear. Practically, however, it ended up simply as a "special" value for which we could test for.

6. Discussion

6.1. Conceptual gains

Conceptual Modeling can help, we might say, both in practice and in theory. There are many aspects of CM which we used and which helped us a lot, even if we only implemented a portion of these mechanisms. In the same way, there are features that we apply in this paper that are purely theoretical and which we did not need to program. In the conceptual world, its simplicity and are strong and convincing arguments. We can imagine. The main difference is that most of these CM formalisms can also be expressed in terms of other CM formalisms. After all, by definition, these are made to be able to express any conceptual nuance that we can imagine. The main difference is that most of these CM formalisms handle meta-information in rather cumbersome and elaborate ways. Here, we have a major mechanism at that meta-level, whose physical implementation is almost trivial.

Another example of effective collaboration occurs with the distinction between unknown values and the explicit representation of no value for previous losses of the concept U/W. Much has been written about various ways to handle such shades of missing values. We have not conceptually resolved this issue yet either. Still, in the meantime, we can enjoy a very simple and effective solution that works quite well. Furthermore, this performance is probably a strong indicator of where our theoretical investigations should focus.

6.2. Practical Gains

We also wanted in this paper to give other researchers a "feet" for the amount of complications that can be encountered when interfacing with real cases that have been in operation for a long time. The scenario that we described is likely typical of most other large-scale insurance applications, and probably could be generalized to many other financial applications. This program presumably started as a fairly well defined, organized and regular system. It is the necessity of adaptations and enhancements done later, without the guidance of the original team, that broke this initial relative uniformity and simplicity. In other words, this story is an example of the effect of entropy on a computerized application. You start with a "simple" system, but the necessities of adapting in more and more complicated ways lead to such complications in a large computer application over time.

Many of the resulting "features" came as unpleasant surprises for many members of that company. Yet they welcomed the occasion to better control what was "really" happening. Some of the "adaptations" were made by maintainers (such as records 15) while others originated from users (such as multiple modules representing a single term of a policy). At times, the documented field layouts would not even correspond to the information we would observe when looking directly at the data file. The documentation we were given was dated as two years old. As well, many times, codes for some fields had changed. In these cases, we could generally get from users the most recent version that they were using. This is typical of how it is not sufficient to trust specifications of what existing systems and data "should be". Instead we have to observe what is really there, uncover all additional conventions that users and maintenance have added, and adapt to them.

On a final note, the strengths and contributions of CM are quite different from those of Expert Systems. We did not discuss here any aspects of the rules needed for screening each policy record. The problems that we described did not involve so much "deep" expertise. The complexity is of a different nature. It comes from the large number of cases, exceptions and complications that keep recurring.

7. Conclusion

The most remarkable aspect of this work is that, in effect, we have established a mutually beneficial link between two extreme representatives of theory and practice. On the practical side, we had an extreme in terms of real-life application. We had to interface directly with the internal representation of data in a very traditional mainframe database. On the theoretical side, we brought to bear on this situation the "state-of-the-art" combination of AI, DB, and PL. This resulting field of Conceptual Modeling (CM) is still an active topic of research in academic circles.

The main practical benefits of this undertaking have been to incorporate these ideas into a system which is now in production and to suggest how CM ideas and techniques could benefit practitioners struggling with problems of similar complexity. The result of our work has been conceptually integrated with the existing application and the operations of the technical staff in this company. The "theory" world also benefits from this confrontation, out of observing which ideas were needed, which ones were most useful, and which ones could be left behind. It is particularly interesting to ponder the implications of the directions suggested by the practical solutions developed for cases where the theory is still unclear.

The partition mechanism emerges as the focus of all these considerations. It is an application of the very leading edge of CM theory, in its use of meta-level information. It even pushes the current limits of recognized ideas in this area, when we identify specialized concepts with partition pro-
property instances. At the same time, we ended up with a surprisingly simple implementation of this idea, out of working under the practical pressures of efficiency and speedy development. This experience is a suggestive example of how theoreticians and practitioners can help each others to break new ground.

Bibliography


Appendix A - Sample Extract from the Original Mainframe Database

0249 561452390000050661206871260172672800567456986127220000000000 (CAPM NADAGAXONG A0) 8612000176171 ... 0300 561452390000050661206871260172672800567456986127220000000000 (CAPM NADAGAXONG A0) 8612000176171 ...