Executable Model Ontology for Temporal Intelligent Organizations in Network Systems

Orientation by David Aveiro

Universidade da Madeira

Abstract. This doctoral project aims to create a paradigm shift[1] on software development by developing the theory and the methodology that allows developing software with one order of magnitude less effort and with over one order of magnitude of perceived improvements, on the combination of interfaces, execution time and interoperability. The candidate foresees the path for this goal by being able to build a simple executable model, based on a theoretical solid ontology, from which we can generate code automatically into fully working applications. The belief that this can be achieved is based on the professional experience over the last 16 years, where the author acquired experience with the development of code generation tools in its own software company for the development of applications for small enterprises. This doctoral project is in its starting stage.

Keywords: code generation, system ontologies, model driven applications, data object model

1 Introduction

This doctoral project aims to create a paradigm shift [1] on software development by developing the theory and the methodology that allows developing software with one order of magnitude less effort and with over one order of magnitude of perceived improvements, on the combination of interfaces, execution time and interoperability. Therefore this project's main goal is to create a "Silver Bullet", as defined in 1987 by Frederick P. Brooks, in the classic paper "No Silver Bullet: Essence and Accidents of Software Engineering" [2] with the interpretation given in the preface of [2a].

The candidate foresees the path for this goal not by building increasingly complex systems that rely on bigger and more complex libraries, packages or frameworks, but the other way around – being able to build a simple executable model, based on a theoretical solid ontology, from which we can generate code automatically into fully working applications.

The belief that this can be achieved is based on the professional experience over the last 16 years, where the author acquired experience with the development of code generation tools in its own software company for the development of applications for small enterprises. The author has built several versions of a proprietary code generation tool to develop Borland Delphi applications (1997-2002) and web based applications (2000-2013) with HTML, MySQL, PHP, javascript and jQuery.

From that experience emerged the need to conceptualize an ontology that would enlarge the domain of applicability of code generation tools from database oriented

applications into more generic and powerful software applications. The new ontology should handle subjectivity, flexibility and the need for handle change in software. By handling change we mean the ability to fully customize generated code over time with minimal code written, but also get back to the ontological model, change it and regenerate code without losing the previous customizations and with bounded impacts on the application with the purpose of minimizing the development and maintenance costs.

The new ontology has to appropriately manage the structural business logic, including both the fundamental core ontological aspects and the operational aspects, using the consistent principles uphold by Enterprise Engineering.

The grail of executable models would be a system that would be able to comply with all of the following demanding requirements:

- Handle any amount of data, of any type, and either centralized or distributed way over a network.
- Handle any number applications that have private data but also shared data providing integration channels with other applications, sometimes with real time requirements.
- Handle any number of stakeholders interacting with the system in different locations through different devices, some of them simultaneously (handling for example, battery constraints, variations on communication throughput, latency or jitter, less memory, less disk space, less parallel computation power, faster results with less precise results).
- Continuously improve performance by increased knowledge about the application and its users.
- Support the strategy pattern to allow multiple algorithms over the same complex data structures.
- Handle broad programming challenges by supporting multi-paradigm programming, logic programming and probabilistic programming.
- Handle code as normal data. Allow code transfer besides the usual data transfer, allowing code to be executed locally or remotely transparently regarding on the best interest of the system.
- Remember everything (events, facts and values) that happened in the application, unless instructed to forget.
- Perform forward inference over events to provide in time analysis and allow temporal analysis for specified or implicit/typical time ranges.
- Allow to travel in time in the system, that is, perform (data, code and event) as if the system was in a certain date in the past to perform analysis scenarios.
- Allow to replay events up from a certain point in time in the past and check for differences between scenarios.
- Allow periodic code re-generation based on the new versions of used functions and overtime compile to memory for finding optimization in the functions where specific patterns in input parameters justify the optimization effort.

2 Software development still in trouble

After 60 years of software engineering, almost no software version passes the teenage years, independently of the amount of money, expertise or effort that was put into creating them. Any software project sponsor still faces a new project with a high associated risk of success.

In "Peopleware" [3] the authors of this classic book from 1999 state that "about 15% of all projects studied (…) were canceled or aborted or postponed or they delivered products were never used. For bigger projects the odds are even worse. Fully 25% of projects that lasted 25 work-years or more failed to complete."

The current reference source of success analysis of software projects is the CHAOS Manifest [4] from the Standish Group. The 2011 reports states that, although it was their best result ever, only 37% of software projects could be considered a success; 21% were clear failures and the remaining 42% were challenged. The fundamental problem didn't change – building software is still hard and challenging.

As time passes by we are getting better at building software, but we are still far from acceptable project success numbers when compared to other areas of engineering. Some argue that the methodologies, paradigms and tools we use today in software development are fine for small projects, but we still lack the knowledge about how to scale up to huge projects, because of specific problems that arise in such highly complex assembles. This is a topic that deserves more research in order to overcome those specific difficulties.

3 Building software

The ultimate goal of building software is to help users to be better by deciding, producing, consuming and remembering better. Being humans we have flaws in our judgments. Science has studied many cases of our biased decisions [5].

Most of our daily decisions are based on simple rules that are really just simple models. Probably the most common model of them all is to copy what others are choosing as decision criteria – there is safety in big numbers. But in the future, people will rely more on software tools to better model the world, be able to make informed and rational decisions (unbiased) and, in time, become better than our nature.

Building software, big or small, is about creating models of reality. Every model is a simplification, so a lot is lost in the modeling process. A software model is useful for analyzing the current state of the world by querying or viewing synthetic reports, for predicting the future or retrodicting (predicting the past).

Models represent worlds that can be continuous or discrete. Actions performed can deterministic or stochastic, that is, unpredictable – especially if the model does not take account with all possible actions that are happening in the world. Information about the world might not even be available (or with the required precision). The roles of the several agents in the world can differ and change over time from collaborative mode to adversarial or even irrelevant interaction.

4 The essential problems of software

According to Frederick Brooks [2], "the essence of a software entity is a construct of interlocking concepts: datasets, relations among data items, algorithms, and invoca-

tions of functions." This author also states that the essential attributes of software that make them so challenging are: complexity, conformity, changeability and invisibility.

3.1 Complexity

Allen B. Downey [6] states that "classical models of science tend to be law-based, expressed in the form of equations and solved by mathematical derivation. Models that fall under the umbrella of complexity are often rule-based, expressed as computations, and simulated rather than analyzed."

According to that author, with the usage of computational modeling there has been a shift along the following axis from:

- continuous to discrete
- linear to non-linear
- determinism to stochastic (random, non-deterministic)
- abstract to detailed
- one, two (few) elements to many elements
- homogeneous to composite (heterogeneous)

This new kind of model is often appropriate for different purposes and interpretations of the world.

- From predictive to explanatory

Many social phenomena's can be explained by simple models like the Schelling's model of segregation, the standing ovation model by Miller and Page's or the threshold models of collective behavior by Mark Granovetter.

- From realism to instrumentalism

As George Box said it: "All models are wrong, but some models are useful."

- From reductionism to holism

Reductionism approach is based on the premise that you can only understand the system when you understand each of its parts. Holistic is the view that some phenomena only appear at the system level and do not exist or appear at the components level. This is also related to the philosophical approach of considering a system in a teleological perspective, that is, considering their use, end and purpose, contrasting with the ontological perspective that focus on the elements that constitute the system, their interconnections, relations, groups, similarities and differences.

According to Downey, this changes may lead to a new kind of engineering and organization of social systems from:

- centralized to decentralized
- isolation to interaction
- one-to-many to many-to-many
- top-down to bottom-up
- analysis to computation
- design to search

And finally, to a new kind of thinking, from:

- Aristotelean logic (only true or false) to multivalent, fuzzy logic
- frequentist probability to Bayesianism

- objective (objectivists – only a single truth exists) to subjective (subjectivists – each person can hold a different truth - or constructivists – truths are established trough social interaction)
- physical law to theory to model
- determinism to indeterminism (breaking the cause-effect link through randomness, probabilistic causation and fundamental uncertainty)

A complex model is not necessarily a better one; actually it is usually the opposite. Albert Einstein said: “Things should be as simple as possible, but not simpler.”

3.2 Conformity

The problem of conformity arises from the need to interact with other existing systems, and usually not very organized ones.

Systems are used throughout science extensively and by definition systems interact with each other. Systems are normally conceived as something that has an input, an internal structure and an output. According to the ontological approach for systems, as defined by Mario Bunge and Jan Dietz [7], a system has:

- Environment – a set of elements that live on the border of the system communicating with the exterior (either input or output).
- Composition – a set of elements that live inside the system and only communicate between themselves and the elements in the ambient.
- Structure – links of mutual influence between elements of the ambient and the composition.
- Production – things that are produced by the elements of the composition and delivered to the outside through the elements of the ambient. The production of a system can be a product (material), a result (immaterial) or a service (a mix of material and immaterial goods provided instantly or over time).

Conformity is a major issue in software development. Software developers need to find ways that systematically address the problems of interfaces between systems. Having a well defined but rigid API is not a good enough solution. The most likely solutions is to address this issue through patterns, like the facade pattern, or even better, through pattern languages that combine typical usages of known patterns to address typical concerns with typical results in terms of non functional characteristics.

3.3 Changeability

As time passes by, people wish to reflect change in their software, but that is challenging and leads to maintenance problems. This aging effect on software was addressed by the Lehman’s Laws of Program Evolution Dynamics [8].

Implementing an architecture that is able to handle change is a huge effort that only pays off in the long run, but on the other hand, having software with a short lifespan decreases its economical viability. We need to find a way to build faster, better and cheaper software that can handle change.

A possible way to tackle the changeability problem is building software bottom-up using normalized systems [9]. This theory states that software handles only two technology independent, primitive entities: data and action.

- Data entities only hold data, without any outside clue on the format in which it is stored and without any associated methods (like in object oriented classes), except for the basic getters and setters methods. A data entity can contain structured in-

formation like in a structure or record, and can also point to other data entities. All references to data entities have a clear reference to the applicable version.

- Action entities can only contain a single task in normalized systems, that is, it only does one simple thing, although the programmer can decide on the granularity. By separating tasks into different actions we separate concerns. Actions can be hierarchical and include calls to other actions. They use data entities as input and produce data entities as output. All action entities have a reference to the applicable version.

3.4 Invisibility

The challenge of invisibility comes from the fact that software is an abstraction that is not tangible. Information visualization of code execution is quite challenging because it is not easy to map data sets to visual encoding. This process of data encoding consists of classifying data types and choosing the most appropriate visual attributes to represent them.

The first problem is data types. Computer's hardware and software typically represent data as booleans, integers (signed or unsigned) or floating points (with a specific precision and range set). This is quite different from the way mathematicians usually classify data (Natural, Integer, Rational, Real, Complex, etc).

On the other hand, statisticians and data scientists typically use only three basic data types [10]: nominal (categorical), ordinal (when we can establish an order between categories) and quantitative. Quantitative can be split in interval and ratio

According to the French cartographer Jacques Bertin (1967) [11] there are 7 visual attributes that can be used for visualizing data, for 0, 1 or 2 dimensions: position, size, value (or lightness), texture, color, orientation and shape. Later, in 1999, this attribute list was extended to 3 dimensions by Card, Mackinlay & Shneiderman [12].

When a programmer writes code he does not have a clear relation to any physical objects. Software could be mapped to a visual representation of code in order to make it more understandable and therefore with fewer bugs.

5 Ubiquitous Organizations

Organizations are ubiquitous and complex social constructions, that can have many forms. We were used to think on organizations as big and well organized structures like businesses, hospitals and factories. Organization theory [13] has evolved to broaden the concept of organizations to include "groups (social structures) whose members coordinate their behavior in order to accomplish shared goals or to put out a product", service or value. With this definition, a group of young people organizing an event through a social network is an organization. Richard Scott [13] argues that organizations can be: rational systems (with collective goals and formalized structure); natural systems (with individual goals but interested in the perpetuation of the organization to achieve them) or open systems (no structure or roles, but flows of interdependent activities with "shifting coalitions of participants").

For many years software engineering has followed the path of building software as a well organized structure. Due to its complexity and stakeholders pressure to change, software development has been shifting toward agile methods that favor gradual growth instead of the "big design up front" approach. Organizations that are natural systems and open systems still don't have the software applications that allow them to

better handle their natural ambiguity and in time gradually become better organizations. On the other hand, structured organizations that are based on rational systems, have difficulties in rapidly adapting to change, because their information systems are hard to adapt and evolve.

6 TOPO – Global Vision

In order to initiate this project the author initiated the development of a platform called TOPO, which stands for stands for Transparent Open Platform for Ontologies. At this stage the author is developing the ontology for TOPO.

TOPO aims at being transparent in the sense that it promotes a white-box [7] modeling engineering approach to the development of information systems, that is, a construction combining elements into more complex structures, taking measures [9] to control the combinatorial explosion and the consequent increase of entropy that can arise from that construction as it has to be changed over time [14].

TOPO has the goal of being open in the sense that all elements of its structure are open to be used and improved by the community.

TOPO will be a platform because it aims to provide a set of shared components and a wide range of non-functional requirements to allow the easy construction of a family of applications. Applications generated by TOPO will support social interactions of actors as described in DEMO theory [7].

In TOPO, users and applications will keep the power of initiative to communicate with others whenever they want. TOPO is not a framework since it does not implement the typical inversion of control present in frameworks, by relying on callbacks, and using the Hollywood paradigm [7]: don't call us, we will call you.

Since TOPO aims to being implemented as an information system artifact, some premises must be stated about the environmental context in which it would operate:

- TOPO is the core of a layered architecture composed by applications, agents, interfaces and persons assembled as can be seen in figure 1.
- Persons can use several interfaces simultaneously to interact with several TOPO applications through an agent.
- Each person has a corresponding agent that represents the person in the system, although agents only perform actions in behalf of persons when previously authorized. The responsibility of actions is always of a person.
- TOPO implementation will be user interface independent and all communications with the interfaces will happen through message exchange.
- TOPO implementation will be in the cloud, although interfaces may keep some cached data.
- TOPO will work over the Internet with multiple devices as view/controller, either browser or mobile device interface (e.g. Android).
- All actors will use message exchange as communication paradigm.
- TOPO will have a maximum response time for each API, otherwise promise a response for later.
- TOPO applications will share common data, but can also have private data for each user/application.

- There will be an integration API for sharing data with external systems. The integration will be performed by a specialized agent.

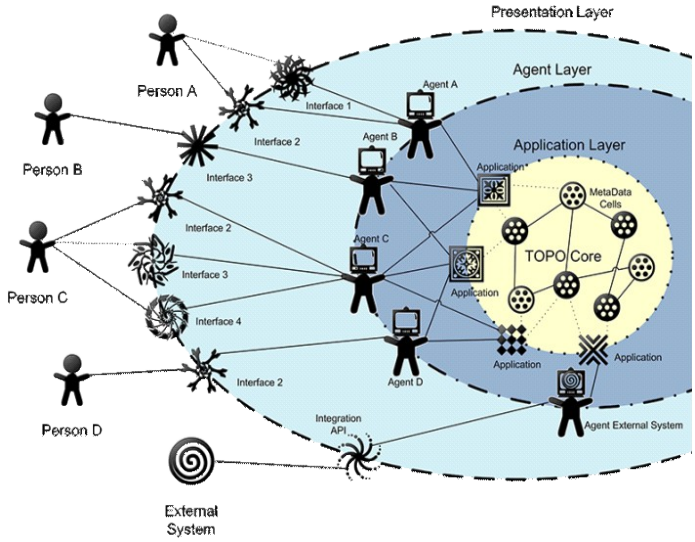

Figure 1 – TOPO layered approach

7 Ontological aspects of TOPO

7.1 General Description

DEMO methodology[7] describes the world using transactions. Each transaction can be initiated by a set of roles, but is executed by a specific role. A ontological transaction follows a universal pattern with the sequence of coordination acts request (by requester), promise (by executor), state (by executor) and accept (by requester), complemented with cancellation of previously taken acts in the sequence. The executor of each transaction also performs a production act (execute) between the coordination acts of promise and state. Transactions at certain acts can initiate other transactions and have dependencies on another transactions. These dependencies are specified in action rules, one for each act in each transaction.

In DEMO methodology there is also a State Model where concepts are connected to other concepts through relationships, and properties are associated to each concept.

7.2 Automata for representation of the Action Model

TOPO will represent a DEMO ontology using a system – that gets some input, performs some actions (either automatically or performed by humans) and produces some data outputs. Notice that the final result of the system is an ontological system. For each act (coordination or production) there is a corresponding action rule. Each action rule is defined as a finite automata where each node corresponds to a task.

Tasks can be of several types (infological or datalogical), that can be either automatic or require human intervention. For example, tasks can include predefined querying some data from the database, flexible querying on the database based on user options, showing some data to the user, requesting some input data from the user, validating input data, storing data, requesting data from web service, etc. All these acts are per-

formed in order for the user to be able to perform the ontological act with the all the available information.

Based on the automatic categorization from similar situations that TOPO should be able to manage, a set of likely data requests should be made available for the case the user might use them. Actual usage should be recorded, as well as new queries to be used as reference for future cases.

7.3 Interface aspect of Tasks

For each task that interacts with the user there is a corresponding interface fragment that can be automatically generated based on the required input data, but can also be fully customized to adapt to specific visualization or data input requirements.

Several interface fragments can be combined in a single user interface, depending of the options and constraints of the device being used.

A task and the corresponding interface fragment can be shared by several actions.

From the authors experience with the code generation tool, a group of 125 types of user interface fields have been identified with the corresponding view, edit and search alternatives. In future research they will be synthesized and patterned.

The arrows in the finite automata correspond to possible actions either automatic (like in the case of if, while, foreach control structures) or requiring user decision when explicit decision is required.

7.4 Agents with mandates

In TOPO, besides actor roles, there are also users, agents and functions. As presented in section 6, agents are the singleton representatives for users, which can perform action on their behalf when properly authorized to do so (mandate).

Agents perform coordination acts by sending messages to each other through the specified roles. For each role there are queues (implemented with sequences) and those authorized to perform a certain actor role can view and/or consume the messages in the queue.

7.5 Organizational functions

Functions, correspond to organizational roles. TOPO will allow a flexible authorization structure specifying which persons are associated to which functions and which of these can fulfill which actor roles.

Functions also perform a crucial role in the delegation, as persons can delegate within the organizational hierarchical functions to perform either production acts, coordination acts or both (with or without reporting back to the delegate on performed acts).

Delegation can also be performed by elements that are not in the same hierarchical chain, or even in the same organization. Additional thought should be invested in this area to understand if and how control mechanisms can be implemented for these special cases of delegation.

8 Complementary elements

8.1 Neuron Model

Concepts can be connected with a neuron like structure. In a simple model, a neuron can be represented with a central body called soma, a tree of dendrites that act as in-

put channels and axon as output channel. If we consider each neuron to be a fact, then dendrites could connect to a arbitrary number of who's, what's, where's, etc. Branches in the dendrite tree can have connections that act as suppressors. This analogy is particularly useful because it can model any boolean expression made up of and's, or's and not's, as it has been shown in neural science.

Neuron model is particularly useful for data retrieval within contexts – the small worlds in networks. If we have a set of fully energized neurons as our current context, it could be possible to energize the structural elements connected to those neurons (who, what, why, when, where, how). Then it would be possible to compute how correlated are other neurons by the level of energy they are getting in their dendrites, and realizing not only the similarities, but also the differences between those new neurons with the one in the current context. In turns it would be possible to join some of those new neurons to the current context and repeat the process, as long as desired. This model of retrieval of information is, in simple terms, the way Antonio Damásio [15] proposes as the way memory works in human minds.

8.2 Zachman Framework dimensions

Zachman Framework dimensions are the elements (who, what, why, when, where, how). These elements are very commonly known as they are the basis for the construction of news.

In TOPO, a “who” can be a “person”, an “organizational unit” or a “role”. A “role” is a generalization of a “person” to allow it to be performed by more that one person over time. “Organizational units” can be arranged hierarchically with great flexibility using the “part-of” structural construction. A person with authority to do so, can link a “person” to a “organization unit” (“part-of”) or a “role” (“instance-of”) with certain mandatory start instant and eventually a end instant. It is also possible to link an “organization unit” to a “role” with the operation “instance-of”.

Elements in “what” are the most complex and versatile element in an ontology definition. Here, artifacts are created and structural relationships are established in order to construct systems, data structures, languages and whatever is required to model the intended application.

A “why” is a tree of “reasons”. For each reason we repeat the question “Why?” until nothing else can be said other that a fundamental value that does not need further justification. Therefore, the “why” can be represented as a three where the branches have “purposes” in free text, and each leaf is a “value”.

In the modeled processes of an organization, the values are built in on the construction of the systems, and do not need to be expressed while in operation. However, when unexpected things happen and decisions have to be taken, values can help to choose the right track among different courses of action, according to what the company has set as their core values and strategy, and also according to value conditions or restrictions that influenced process design at the respective organizational change context. [16]

The dimension “when” handles time references (moments and recurring periods) and time ranges. Time is one of the most problematic topics because society does not use a good model to handle time, since we have multidimensional layers with great level of ambiguity and inconsistencies.

The dimension “where” is a controlled vocabulary that can reference to many spaces, either in absolute terms of in relative terms. Unlike time, there are very well established systems for coordinates (Cartesian, polar) and available tools to use these systems, even in mobile equipments. A location can be set using a global coordinate system (absolute) or a local one using objects or earth magnetic field to establish coordinates.

The dimension “how” can have multiple interpretations, either using action rules previously described; or in a functional perspective of social decomposition of activities in textual form.

8.3 Metadata

Metadata is usualy described as data about data, but what is metadata for some might be raw data on which others would like to build upon. In TOPO all data is metadata.

There are 3 types of metadata: **descriptive** (that tells features about a “thing”); **structural** (that says how “things” are connected to other “things”) and **administrative** (that tells who has rights about each piece of data and how that data was formed (provenance) and how it should be kept – for how long, how and by whom.

Descriptive metadata will be added to any concept in TOPO. Every descriptive metadata is a stored as a fact that links: a) the concept being described; b) the value (that is also a concept); c) a predicate (not mandatory) stating the type of description being made – also a concept; d) a unit of measure that is only required for interval values (not mandatory); e) the error associated with the measurement (not mandatory), only for ratio types of data and by default in the middle of one order or magnitude below the measurement.

The predicates to be used should, as far as possible, be one of the 15 basic types of Dublin Core [17] or one of its extensions, in order to facilitate future semantic integration between ontologies.

The structural metadata is used to establish fundamental relationships between data. A lot of work has been done in identifying the properties of structural relationships between concepts in foundational ontologies, namely in [18] and [19]. Therefore we will just mention some of its elements and rely further analysis to the references and to further work. In general, structural relations are established between individuals, collections and the universal set of values. Some examples of structural relations are: individual part-of individual; individual instance-of universal; individual member-of collection; universal is-a universal (taxonomic inclusion); universal partonomic-inclusion-of universal; collection extension-of universal; collection partonomic-inclusion-of collection; collection partition-of individual.

Unlike descriptive and structural metadata, administrative metadata on TOPO will be mostly automatically collected by the system using default options and current context, although the user can change it later on. Administrative metadata will be stored using neuron model and the Zachmann Framework dimensions and using, as much as possible the semantics of Dublin Core and its extensions, as well as the advancements in provenance of digital documents from information science [20], [21].

TOPO will use the paradigm of no raw data deletion, that is, all raw data is stored with timestamps (administrative metadata) and future changes to data are stored as new facts with newer timestamps. Those facts are never deleted to allow time travel,

scenario testing and replay. There are always some time references that create forward inferences and data snapshots of the state of the world, for example, in the start of each day of current and last week, start of each weeks of current and last month, etc.

9 Incremental formalization – putting it all together

The purpose of this section is to show how all concepts previously shown can be joined together in order to provide user with a way to incrementally formalize the information's system for its organization. This formalization can happen in a sequence of simple steps.

Following the general idea that software for organizations should grow in an incremental way instead of being build upfront, lets image that there is a client and a provider selling lemonade at the door step with delivery to the clients location.

1. No formalization

In the beginning there is no formalization. The client send messages (personally, or throw email, sms, social network, etc.) to the provider and the provider handles each request (delivering the lemonade himself or with the help of the assistant) without any formalized process. Requests, promises, execution, statement and acceptance are done informally. The provider has only one single transaction to execute: manage messages. Each request is handled differently depending of the request being made. No records are kept.

2. Joining dimensions

Assuming that the provider is using TOPO, he can start using the Zachman Framework dimensions, described in section 8.2, to take notice of simple facts in each message he receives from clients, and mark small portions of it as pieces of information just stating the dimensions: who, what, where or when. With this information's , records can be kept. There can be many instances of the same dimension in the same message. It's as if the client was using colorful markers to mark certain parts of the message.

The provider does not have to provide any additional information, although some notes may become handy in each association.

3. Neuron Model usefulness

Each relation between the dimensions of Zachman framework an a message (or part of it) is stored using the neuron model described in section 8.1.

With this primitive data system within TOPO, the provider can perform simple search based on a time line (when), on a map (where), on a person or organization (who) and on assets names (what). We are assuming that TOPO is smart enough to: understand references to dates, knowing the present date and understanding the text; use controlled vocabularies for locations differentiating geographical locations from logical ones (within the organization); group names by similarity and parts of names. Names of assets are more difficult to handle at this stage, but they can be handled as tags.

Even with this very primitive information system, automatic data mining is already possible since when is a interval type of data and all the others are categorical data. Therefore it is possible to count how many times a clients performs orders, from where and detailed analysis on when.

4. Creating attributes

As business flourishes, the provider may feel the need to set roles for specific dimensions, namely create specific tags for each product being sold, since now the provider sells a large set of natural juices. He may need to record the request time, delivery time, request location, delivery location, etc. He may even need to identify which client is the payer, the chooser, the receiver, the beneficiary, the influencing persons to buy, etc.

5. Creating concepts

At some point in time, organizing all information around the order message is no longer the most efficient and logic way to proceed since there is too much repetition of data. There is the need to add concepts: client, delivery location, resources (fruit, water, sugar), equipments, etc.

However, TOPO still allows the user to freely associate as many Zachman dimensions as needed to each record of each concept. Keeping this flexibility allows to still have the benefits described in steps 2-4, even for the new concepts.

Some typical concepts in organizations should be used, for example for handling organization internal structures (for example: delivery boys; financial manager).

6. Setting data types and constraints

In order to prevent data records with invalid data, some field may require to set a list of possible values (for example: no juice deliveries after 10pm). Some, might even be associated with controlled vocabularies defined by the user. Some attributes may be marked as mandatory (items in order and amount), others as recommended. These constrains of possible values can be formalized as new concepts called types.

In TOPO concept types are described using the descriptive metadata structure described in section 8.3. They allow to name an attribute, constrain possible values (either categorical, ordinal, interval and ratio), describe it, associate a unit of measure (for automatic conversion) and an associated error in measurements (for error propagation calculus).

Numeric attributes are the usual victims of constrains since they are more prone to errors and more difficult to detect and have more variants on format and usage.

With concepts, attributes and types we have the usual building blocks normally used in developing software. Notice that with these new information a wider range of data mining can now be performed automatically. Many patterns should be provided by TOPO in order to facilitate creation and transformation of structural metadata.

7. Creating structural relationships between concepts

With the new concepts and attributes comes the need of structural metadata, described in section 8.3. Structural meta-data allows to model in TOPO UML or ER or State Model from DEMO. For example: a Client can have several typical delivery locations.

By supporting all types of structural relationships according to some foundational ontologies, the authors expect to provide a more powerful solution than the common ones. This shall be addressed in future work.

It should be possible to add, remove and transform concepts, attributes and their structural relationships over time with bounded effects, as prescribed by normalized systems. This shall be addressed in future work.

8. Formalizing and structuring transactions

DEMO methodology introduces a pattern of coordination acts and the ability to set dependencies between transactions on certain coordination acts, as described in section 7.1. For example, creating a payment transaction, an outsourced delivery transaction, etc.

TOPO will not commit to a unique possible pattern, as some difficult issues still exist like the need for the client to be the initiator of the transaction, the difficulties in the integration of infologic and datalogic transactions in the ontological pattern, the handling of delegation and the handling of discussion states after cancellations.

9. Formalizing tasks in the action model

Organizations have to adapt to circumstances in a rapid changing environment. Therefore users with the appropriate level of responsibility should be able to modify the tasks of a specific action model. This should allow formalizing what should be done typically, what should be checked for a set of criteria. But it should also be override by a responsible person that is able to evaluate that a different procedure is better suited for that situation.

The neuron model can by used to setup boolean expressions or even serve as basis for neural networks to classify clients and business opportunities based on passed experiences.

10. Automatizing steps

The final steps consists on persons delegating some responsibilities on software agents (more or less intelligent) that act on the persons behalf.

Analysis

The usual way software is developed today is jump directly to steps 5 to 7 – restraining fields and data types, and concepts to hold them. Most of the times software development doesn't support transactions at its full extent (steps 8-12), but also blocks the possibility to store non structured information, as in steps is defined in steps 2-4.

In the authors opinion this lack of flexibility creates more pressure to change software as all small (even if infrequent) differences from the expected path creates a barrier for the users because they normally do not allow any walk around solution.

10 Conclusion

This project is in its initial steps. The author have chosen an incremental and bottom up approach, so that implementation can take place with a realistic scope and, at first, simple ontologies could be generated into working prototypes.

Certainly many constraints and insights will come up from the future steps of specifying concepts and implementation aspects taking in account all the dimensions, concepts and innovative ideas presented in this document, that will lead to a creation of a theory and a methodology for developing software.

References

[1] Kuhn, T. S. (1996). The structure of scientific revolutions . University of Chicago press.

[2] Brooks Jr, F. P. (1987). No silver bullet-essence and accidents of software engineering. IEEE computer, 20(4), 10-19.

[2a] Brooks Jr, F. P. (1995). *T*he Mythical Man-Month, Anniversary Edition: Essays on Software Engineering. Pearson Education.

[3] DeMarco, Tom, and Timothy R Lister. 1999. Peopleware: productive projects and teams. New York, NY: Dorset House Pub.

[4] http://www.standishgroup.com/chaos_news/newsletter.php?id=54

[5] http://en.wikipedia.org/wiki/List_of_biases_in_judgment_and_decision_making
http://www.ted.com/talks/tali_sharot_the_optimism_bias.html
http://www.ted.com/talks/dan_ariely_on_our_buggy_moral_code.html

[6] Downey, A. B. (2012). Think Complexity: Complexity Science and Computational Modeling. O'Reilly.

[7] Dietz, J. L. G. (2006) Enterprise Ontology-Theory and Methodology.

[8] Belady, L. A., & Lehman, M. M. (1976). A model of large program development.IBM Systems Journal, 15(3), 225-252.

[9] Mannaert, H., & Verelst, J. (2009). Normalized systems: re-creating information technology based on laws for software evolvability.

[10] Course Introduction to Data Science – Cecilia Aragon - University of Washington (May 2013) https://www.coursera.org/course/datasci

[11] Bertin, J. (1983). Semiology of Graphics, 1967.

[12] Card, S. K., Mackinlay, J. D., & Shneiderman, B. (1999). Readings in information visualization: using vision to think. San Francisco, Calif.: Morgan Kaufmann Publishers.

[13] Scott, W. R. (1981). Rational, natural, and open systems.

[14] Lehman, M. M. (1996). Laws of software evolution revisited. In Software process technology (pp. 108-124). Springer Berlin Heidelberg.

[15] Damasio, A. (2012). *Self comes to mind: constructing the conscious brain*. Random House Digital, Inc..

[16] Pombinho, J., Aveiro, D., & Tribolet, J. (2012). Towards Objective Business Modeling in Enterprise Engineering–Defining Function, Value and Purpose. In*Advances in Enterprise Engineering VI* (pp. 93-107). Springer Berlin Heidelberg.

[17] http://dublincore.org/

[18] Bittner, T., Donnelly, M., & Smith, B. (2004, November). Individuals, universals, collections: On the foundational relations of ontology. In *Proceedings of the Third Conference on Formal Ontology in Information Systems* (pp. 37-48).

[19] Guizzardi, G. (2005). *Ontological foundations for structural conceptual models*. CTIT, Centre for Telematics and Information Technology.

[20] Simmhan, Y. L., Plale, B., & Gannon, D. (2005). A survey of data provenance in e-science. *ACM Sigmod Record*, *34*(3), 31-36.

[21] Moreau, L., Freire, J., Futrelle, J., McGrath, R. E., Myers, J., & Paulson, P. (2008). The open provenance model: An overview. In *Provenance and Annotation of Data and Processes* (pp. 323-326). Springer Berlin Heidelberg.