(Meta) Meta Model Extensions for Manageability of Large Scale Collaborative Modeling

Graham McLeod

Inspired Consulting Training Research and Tools
P O Box 384, Howard Place, 7450, South Africa
mcleod@iafrica.com – http://www.inspired.org

Abstract. There is ample evidence to suggest that collaborative modeling offers significant advantages over modeling carried out by individuals. Collaborative modeling can be achieved by workshops and other interactive techniques. Recently there has been increasing interest in supporting collaborative modeling with web and repository based tools, especially where the desired participants are separated by distance and time zones and potentially language. This paper introduces and formalises some constructs and extensions to meta models (and meta meta models) which have been found useful in enhancing the usefulness of large scale collaborative modeling tools and the manageability of the models employed in support of enterprise architecture management. Issues addressed include: subject domain, ownership, authority, context, time, version, status, multiple opinions, user groups/roles, multiple languages and avoiding information overload.

Keywords: Meta Modeling; Contextual Modeling; Collaborative Modeling; Repository

1. Collaborative Modeling

Modeling is the activity of creating manageable and useful representations of things in the real world. Models offer the ability to understand complex structures, phenomena or behaviours in a safe and cost effective way. They also offer the opportunity to safely and cheaply alter assumptions and input variables and observe the outcomes that these produce. In Information Systems, modeling is widely used to understand business requirements, processes, data and information and technical architectures and solutions. Collaborative modeling has been used extensively and productively for many years in the shape of Joint Application Design (JAD) and other facilitated physical group session techniques.

Information systems are increasingly vital to organizations, indeed strategic in many industries. They often nowadays extend beyond the confines of a department, single business function, business unit or even the enterprise itself. The user and specifier community also frequently spans geographic locations and time zones. These factors have driven researchers and practitioners to pursue technology supported techniques, including groupware, knowledgebases, wikis, electronic

whiteboards and others, with the aim of achieving some of the benefits of collaborative physical sessions in the virtual world (Engelbart, 1992). Additional benefits could potentially be achieved by the ability to involve more parties, reduce costs, achieve collaboration despite distance, time zone and language barriers.

2. Background to Our Work

Our organization has developed and marketed a collaborative web-based enterprise modeling repository and tool environment (Archi/EAWebModeler) since 2000. This is primarily applied in the development of Enterprise Architectures, Programme Management, Methods Management and Strategy Formulation. As our clients have tackled larger and more complex projects, we have been encouraged and pressured to provide a range of services which assist in the management of these efforts and improve the usefulness of the tools. Some of these requirements have already been incorporated into existing product. Some have been prototyped in pre-release versions of the tools or in proof of concept developments. Others have yet to be fully tested and designed. Our experience in dealing with these problems and in finding solutions is brought together in this paper, which intends to summarise the requirements and identify and formalise proposed solutions meeting these.

An example of the application environment is one of the world's best known Information Technology hardware and services vendors which is managing a process of transformation and rationalisation, including global business unit reorganization, business process improvement, ERP implementation on a massive scale and reducing the count of in house applications from over ten thousand down to less than four hundred while simultaneously ensuring continuity. This involves a massive amount of modeling (including business structure, business process, applications, data collections, services and supporting infrastructure and other aspects) by several distributed teams working concurrently. Much of the information is imported from or exported to other tools, captured, navigated or maintained via web interfaces, or via graphical models.

3. Challenges Encountered

In deploying this type of repository and collaborative modeling environment, the project, client and ourselves have encountered a variety of challenges, including:

- *Volumes and performance*. Our tool achieves great flexibility (extension and redefinition of meta models at run time) through the use of a high degree of abstraction in the definition of internal repository structures and persistence layer as well as the run-time parsing of interface and business logic patterns, generating required web interfaces. This abstraction has a cost in terms of amount of processing and input/output activity for persistence. Volumes can grow to dealing with millions of repository objects and several hundred thousand of a single type (e.g. Data Column). When one multiplies this by

the number of concurrent users and the number of objects that may require retrieval for one instance of the type, significant challenges arise in maintaining acceptable response times.

- *Managing ownership of objects and rights* regarding who can create, edit, delete, retrieve and otherwise manipulate them. These need to be managed for groups, roles and individual users as well as for composite objects (such as models and documents) and at the levels of abstraction relevant to the meta model, the instances of types and the values of defaults and templates.
- *Avoiding information overload in the user interface.* With the vast amount of information available (more than 300 types; up to several hundred thousand instances of a given type; the richness of a single item's information (given all the contexts in which it might be used) running into large numbers of properties (30+) and extensive relationships (30+) for a single item) it becomes a real challenge to only show users what is relevant and interesting and in a way that is responsive and useful. This requires selecting just the types required, the instances required as well as the view of that information required (subset of properties and relationships as well as preferred representation)
- *Managing local and global views* – which may involve promotion of items based upon state. e.g. I may want something to appear only in my personal view while I am toying with an idea; to move to a group visibility when we work on a more defined version as a team; and then to be promoted to global visibility when it is ratified as a corporate policy or approved item
- *Presenting information in the best way for different user groups.* Managers may respond best to a "rich picture" view allowing drill down from a dashboard; Technical personnel and analysts may want a report or matrix; analysts and modelers may prefer a graphical model while we may need an XML representation for integration with another tool or environment
- *Supporting different "versions of the truth".* These may represent the evolving picture over time (previous version, current version, future version), *scenarios* given different assumptions or goals or even *opinions* of various stakeholders about a single concept at a single time
- *Dealing with the same semantics in different languages.* Where the modeling community is large, a variety of languages may be used natively by the participants. It is desirable to allow users to work in their own language but still produce meaningful views of the information or models for other language users
- *Variable information quality or status.* When one looks at a community product (e.g. Wikipedia), some content items will be more authoritative than others (Anderson, 2006). In a collaborative modeling environment, items will often evolve through a variety of states, e.g. from the idea of one individual to a group draft for critique, to an official sanctioned view. We

may want to see only some of these, or at least be informed of the status of what we are viewing.

4. Potential Solution Concepts

We formulated (through literature research, our own individual work and collaborative modeling) a set of concepts for addressing the challenges introduced above. Some of these are already deployed in the production product, but others were the subject of prerelease development or proof of concept implementations. These include:

- *Caching and inherent hierarchies* have been pursued to preload instance collections and provide rapid presentation and navigation (via expansion) of own type relationships between items of a single type which support hierarchies and network relationships. It is possible to support multiple hierarchical relationships per type. E.g. Business Unit objects might be arranged in a hierarchy of geography as well as reporting structure
- *Domains*, defining a collection of related types, usually by subject area, were introduced to provide a convenient way of administering subsets of the meta model from a security and rights perspective, as well as a way of rapidly filtering repository content to reduce information overload for users who have rights to large tracts of the repository, but currently wish to work within a restricted focus area (e.g. Process Architecture)
- *Context* was conceived as a way to provide a selection and filtering mechanism for instance data. Examples of context are: Business Unit; Project; Geographic Location. Note that these can be instances of types used for other modeling purposes as well
- *Filters* are a very important mechanism in the current implementation and will be enhanced further in future designs. These live functionally between the persistence layer which retrieves data from the repository and the business logic layer which processes it and passes it to the view layer for presentation, import or export. Filters are attached to an active user session and influence what that user will see from the universe allowed by security permissions. Filters exist to allow:
  - Filtering *domains* of interest
  - Filtering *items* based upon *attribute values* (including identity; date/time of update; user modifying)
  - Filtering based upon *relatedness* of items to other items (either types of items or specific instances)
  - Filtering based upon *hierarchy* (level within own type hierarchy; membership of the tree below a given parent; relatedness to an item within a designated membership tree within the same or another type)

It is possible to combine filters with logical conditions (AND/OR), to sequence them, and to force them to be active upon user login, thereby extending the security system. Filters can be named and saved for ease of reuse. Plans exist to extend the filter capabilities to also allow:

- Filtering via *context*
- Filtering on *status*
- Filters which accept execution time *user inputs* as arguments for comparison
- Filters which allow *sampling* instances (e.g. Return every *n*th item..) useful for analysis, testing etc.

- *Automated chunking* of large result sets has been implemented to:
  - (For types with an inherent hierarchy): present an expanding tree view (not new but enhanced with caching and Ajax to improve performance)
  - Examine the pattern of identity properties for a set of instances and achieve a meaningful pseudo hierarchical index which presents a reasonably small result set and allows easy navigation to individual items. This can provide an alphabetically organized tree while ensuring that levels do not contain too few or too many entries
- *Opinions* are to be supported by allowing values per property per user (but only within the content layer of the schema)
- *Representation* can include a variety of views of the data, including:
  - Text
  - HTML plus CSS (for browser presentation)
  - XML (for import, export, API or tool exchange)
  - CSV (for exchange with spreadsheets and other tools)
  - Matrix (cross reference between instances of two types)
  - Worksheet (similar to spreadsheet or relational table with rows representing items, columns representing properties)
  - Graphical. This includes models and embedded vector graphics or bitmaps representing repository instances and relationships. Meta models and model types are also supported.
- *Model Type* is a structure for defining the combination of types, relationship types and representation required for a given viewpoint or model goal. Examples would be UML Use Case; Domain Business Object Model; BPMN Process Model. Arbitrary model types can be user defined at run time and use any concepts present in the meta model. Representation symbols can be user defined

- *Model* is a collection of repository items and associated relationships which are visually represented in the same model. Models are consistent with a single model type
- *Project* – a collection of related items or models (themselves items) from the repository, of interest to a given community. Could also be seen as a "namespace" within which identities would be unique
- *Documents* provide a way to define a meta structure (start point and navigation path over types and relationships as well as specifying sequence and required properties) for retrieving matching item data and generating a composite document
- *Versions* – The ability to hold previous, current and future values and states of content for repository items
- *Scenarios* – Larger scale collections of repository content associated with a scenario and having a relationship describing the association (e.g. An item might appear in multiple scenarios and be described in one as: "Retain, but improve performance" while in another it may be described as: "Replace with ERP solution".
- *Packages* were introduced as a way to specify export sets, including domains, types, item instances. Primary purpose is support of migration, distribution of meta-sets (meta models + model types + instance data), and synchronisation between repositories
- *Journaling* – with many participants potentially updating shared content from many locations and 24/7 operation, it becomes vital to log all updates and record who performed them. This is necessary at both instance and meta model level. This is achieved by non destructive updating (by cloning and modifying a new instance of the item modified) with automatic chaining between previous and current versions

5. Prototype

While the current production implementation is powerful and reliable, it has grown over time and various concepts have been added for pragmatic purposes, not always in a comprehensive or consistent way. It was realised that it may be possible to unify, simplify and improve the efficiency and efficacy of the design by analysing the current facilities and merging these with purer versions of the concepts discussed above. We were also influenced by the work of Martin Fowler in Analysis Patterns 1997; Pieter Wisse in Metapattern 2001; Lukas Renggli in the Magritte meta described system 2006 and our long experience with the Smalltalk language and the Squeak implementation thereof.

We undertook prototyping activity and proof of concept work in Squeak with a view to trying out different implementations of some of the above concepts. Useful insights were obtained in this way, including:

- The need to include a clean separation of a *three layer object architecture* (not for the user interface, business logic and user interface i.e. MVC, but for the meta meta, meta and instance levels of representation of model defintion and semantics). This needs to be achieved at a logical level even though the implementation may flatten the higher two layers to Smalltalk classes
- The idea of *context* is very powerful and can potentially be used for many of the solution requirements, including:
  - User concepts such as Business Unit or Geographical Location
  - Separating local (User), shared (Team or Group) and Global contexts where a user's default view might include items from his/her own local context, his/her team and his/her organization's global context
  - Distinguishing items by state or "officialness"
  - Scenarios
  - Models
  - Packages
  - Projects

  In short, anywhere where we need a selected subset of items. Similar concepts can be exploited at the *meta level* for concepts such as domains and model types
- It was found to be useful to introduce a *meta meta classification of relationship types* as well to allow some relationships to be treated in special ways. These will include concepts such as "parent/child"; "requires/required by"; "instances/instance of"; "contains/part of"; "precedes/follows" and also spatial relationships such as "above/below"
- A goal identified earlier, viz simplifying definition of models, reports and documents, is supported by *unifying the concepts under the abstraction of model type* categorised into a variety of types including: document; report; graphical model. An important addition to this list is *User Interface*, which it was found, with the new flexibility in specifying representation, could be described just as another kind of model/document
- *Filters* at the type/relationship type/domain level can be merged with the structure of model type, but with different behaviours (exclusion rather than assembly). Filters for instances can be merged with the concept of model in a similar way, since the pattern required is similar
- The concepts of tracking changes over *time, version and baselines* can be unified allowing easy identification of current, historical and future value

- The concept of retrieving items via *state* can be achieved with a property filter

6. Formalising the Models

The resultant model is shown below. The notation shows concepts (conceptual classes) in boxes. Inheritance is shown via a directed arrow (as in UML). Example instances are shown as bullet points below the concept box. Containment is shown via a UML aggregation symbol (diamond on the container end). Associations are shown as lines. Cardinality of relationships is indicated simply via an asterisk indicating a "many" possibility on that end of the relationship. An own relationship (such as a hierarchy between items of the same type) is shown as a containment relationship to the same concept – a square on the corner. Pure is the name of the proof of concept project.

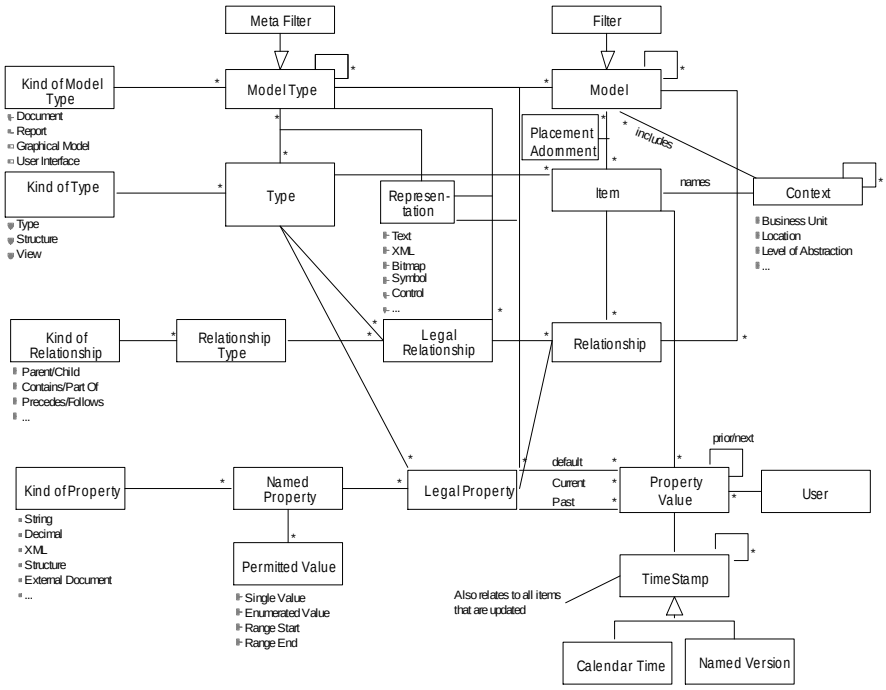

Fig 1. Conceptual Model Including Enhancements.

Left hand column represents meta-meta constructs: next two columns hold the meta model; and those on the right model instance content. Note that the above model does not include the handling of users, groups and roles, mainly due to space constraints.

7. Remarks and Future Work

The model has shown high flexibility, but has also been challenging to implement at the design level with the requirement for high performance. We are trying to integrate all structure definition (including of user defined types and kinds of properties) with the Smalltalk class system for ease of writing business logic, efficiency and persistence. We have used collections heavily in the implementation and have found these to be remarkably effective and efficient, provided the type of collection and navigation strategy are carefully chosen. We are investigating the use of set operations to implement many filtering, selection and presentation operations previously coded as application logic in prior implementations. We have to date not tested massive volumes, but early results are encouraging.

Further work is required in the areas of:

- defining business logic patterns and
- using the representation model to drive automatically generated user interfaces for complex property types (to date we have confined efforts to simple data types)

While we have only exploited the concept to a certain degree in this work, we believe that the concept of context as espoused by Wisse has great merit and would suggest that language designers (particularly Smalltalk and Ruby) consider implementing these constructs in a direct and accessible way.

References

Anderson, Chris, 2006: The Long Tail – How endless choice is creating unlimited demand, Random House Business Books

Englebart, Douglas, 1992: Toward High-Performance Organizations: A Strategic Role for Groupware, Bootstrap Institute

Fowler, Martin, 1997: Analysis Patterns: Reusable Object Models, Addison Wesley

Inspired, 1999-2007: Internal Archi design documents.

McLeod, Graham, 2001: PAMELA: A Proto-pattern for Rapidly Delivered, Runtime Extensible Systems, Proceedings EMMSAD 2001, Interlaken, Switzerland

Renggli, Lukas, 2006: Magritte - Meta-Described Web Application Development, Masters Thesis, University of Bern, Institute for Information Systems and Applied Mathematics

Wisse, Pieter, 2001: MetaPattern: Context and Time in Information Models, Addison-Wesley