Generic unified modelling process for developing semantically rich, dynamic and temporal models

Generic Unified Modelling Process for Developing Semantically Rich, Dynamic and Temporal Models

Hubert Grzybek, Stephen R. Gulliver

Informatics Research Centre (IRC), University of Reading, Reading, UK
** Tel: +44(0)++++++++++, E-mail: ++++++++++*

Abstract
Models play a vital role in supporting a range of activities in numerous domains. We rely on models to support the design, visualisation, analysis and representation of parts of the world around us, and as such significant research effort has been invested into numerous areas of modelling; including support for model semantics, dynamic states and behaviour, temporal data storage and visualisation. Whilst these efforts have increased our capabilities and allowed us to create increasingly powerful software-based models, the process of developing models, supporting tools and /or data structures remains difficult, expensive and error-prone. In this paper we define from literature the key factors in assessing a model's quality and usefulness: semantic richness, support for dynamic states and object behaviour, temporal data storage and visualisation. We also identify a number of shortcomings in both existing modelling standards and model development processes and propose a unified generic process to guide users through the development of semantically rich, dynamic and temporal models.

1. Introduction

Models are commonly defined as a representation of aspects of the real world, developed from a certain perspective, used in specific purpose [models in education paper] and perceived by a defined actor [1]. Models allow us to represent an object or idea of interest and communicate that idea with others. A group may then collaborate to analyse, modify and gain information from the model. Models are therefore used for knowledge representation, design information sharing and analysis, and as a means of recording information that is valuable to a domain. In this light, we argue that information systems in general are models in the sense that they store information that represents important facts about a domain. This information is then used by actors in order to support that domain.

The usefulness of a software-based model is heavily reliant on how accurately it portrays its target domain. Reality alignment, in terms of a software system, demands that: the conceptual data structures being used by the system are based on an accurate representation of the domain; that the recorded data itself is accurate; and that the rules and operations that manage the data reflects the way that domain users would otherwise interact with the information. The resulting system must also be capable of responding to queries specified by a user. For purely visual models, such as CAD and CAM, this is straightforward as the ability to visualise the model data is sufficient to communicate the intended information. However, for models that contain relational semantics, such as those commonly based around RDBMS (Relational Database Management Systems) systems, the ability to answer a range of queries requires significant effort on the part of the system designers. If we wish to specify the behaviour of objects, the dynamic state of objects, or even record the history of the model to support temporal analysis, we face considerable complexities. Fields that are crucial to modelling language and tool development (i.e. ontology and conceptual data modelling, behaviour modelling, visualisation methods and temporal data management and cognitive sciences) have enjoyed a great deal of focussed research. However, the fruits of this research have yet to be aggregated and compiled into a generically applicable modelling process to support the development of semantically and / or visually rich, behavioural and temporal software-based models.

In this paper we will present and justify the crucial factors that have a major impact on the usefulness of models. This perspective will then be used to analyse a number of existing modelling processes and software-based modelling standards, in order to highlight their clear respective deficiencies. Finally we present a generic model development process, which we believe will lead to improvements in the quality and usefulness of software-based models.

2. Key Factors Involved in Assessing Model Usefulness

From consideration of literature we have identified five factors that we believe to be vital when assessing a model's usefulness, regardless of the model's domain and / or intended usage. These factors are: inclusion of semantics, facilitation of dynamic states and behaviour, support for temporal data and analysis, extensibility and a capacity for integrated visualisation. The following sections describes these features in more detail, their importance and our justification for why their selection is critical to provision of a generic modelling process.

2.1. Semantics

The common factor between any modelling process, in any domain, is that the interpreter, assuming a correct understanding of the model semantics, is able to explain the meaning of the model through a set of natural language statements. For instance, a graph consisting of nodes and edges could be converted to an equivalent text-based form, consisting of statements such as 'node X is connected to node Y through edge A'. Often, graphical models are used in preference of equivalent text-representation as they carry a number of benefits. In the case of graphs, the ability to quickly identify which nodes are connected is a benefit that cannot be matched by text-based representations. As a result, the creation and interpretation of models relies on our ability to interpret from and to natural language descriptions. Based on this relationship, we identify that the semantic richness of a model is reliant on our ability to identify that the statements we can make based on the model's content satisfy the given information requirements. Further, a model's semantic richness and usefulness is enhanced if we collect information from the model based relating to a number of different types of question. Initially we asked who, what, when, where, why and how questions in order to investigate information needs of static models, i.e. typically flat-file based fixed state models. Interestingly this quickly leads to difficulties as the above questions are focussed on analysing a set of events or circumstances during which some actions (for which actors are responsible) took place. With static models, since the model exists outside of the dimension of time, we are only able to question the current state of the model, i.e. 'What exists now?', 'What are the properties of object X now?', 'Which objects is object X related to now?'. Consequently, static models are commonly used for specifying the final design, with the model sometimes being used for product planning.

Whilst such models are undoubtedly extremely useful in supporting design activities, the fact that they are based around static technologies and standards significantly prevents the information from being used for alternative purposes, i.e. in a dynamic, behavioural and temporal context. A good example of this is IFC (Industry Foundation Classes). IFC provides a static, relational and object-based standard focussed on semantically rich 3D architectural designs. The standard is gaining in popularity due to its status as an open standard, which helps in preventing vendor lock-in, as in the case of many associated CAD products. The IFC object model supersedes standard CAD drawings by including full relational object semantics, i.e. rooms are semantically marked up as spaces and are bounded by 'wall' objects with associated properties. The equivalent in CAD is commonly represented by a simple polygon with no object labelling, properties or relationships. Despite the semantic richness of IFC models, which is supplied by the network of entities, attributes and relationships, such models are stored as text using either the STEP format or as XML. As a result, individuals wishing to use information contained within an IFC model for purposes that require the data to be treated as dynamically, run into problems. This is demonstrated well by Spearpoint [2] who, in the context of zone fire simulation software, encountered difficulties when attempting to use the IFC models.

2.2. Dynamic Models and Object Behaviour

Dynamic scenes can be built from the semantics of static scenes, by treating entities as capable of change across the dimension of time; normally as a result of dynamic events. These events may be caused by actions occurring outside the model, i.e. the model being used to represent events taking place in the real world (e.g. purchases from an online store). Alternatively, models may be driven by events taking place within the model itself, or a combination of both. This is best described by Sowa [3], who distinguishes all models in terms of : definitional (emphasising the hierarchical relationships between concept types), assertional (used for representing propositions and conceptual structures) and implicational networks (used for representing patterns of belief, causality and inferences). An additional three categories: executable networks (objects within the model contain algorithms which perform analysis or changes to the model itself), learning networks (example cases are used to tune the model's representation by modifying *weights* associated with nodes or arcs) and hybrid networks (a combination of two or more of the previous approaches) are used by dynamic models as they treat the contained information as subject to change. Executable networks are especially interesting as they require that the system includes algorithms that analyse or change the state of the information, i.e. objects within the model including algorithms which, when triggered, update the state of the model. Twinned with the dimension of time, models supporting behaviour allows us to represent events, including their causes and effects, in order to create a closer representation of a real-world domain. This concept is used in a wide range of systems, such as computer games and simulations, in order to associate the potential for a particular entity or type of entity to cause a certain type of event. For instance, a dynamic model of a building would be capable of automatically creating emergency plans, such as determining safe fire exit paths, based on the state and event of certain fire alarms in the model.

If we now ask who, what, when, where, why and how questions, we find that dynamic models provide us more positive results. Take a building fire example, by adding behaviour and the dimension of time, we are able to ask questions such as "what event is currently taking place?", "what is the current state of the model?", "which objects are currently involved with the event?", "who/what is responsible for initiating the event?" and "Where is the event taking place?". Accordingly, a semantically rich model, supplemented with object behaviour and

dynamic states, allows us to portray the mechanics of a domain in a highly detailed manner. However, aside from a few exceptions, such as temporal GIS, few software-based models are implemented with the capability of recording previous model states. Lack of a temporal dimension limits us only asking about the current of the model.

2.3. Temporal Data Management and Analysis

Since we have already established flat files as not being for dynamic models, we must decide on another option. Databases are arguably the most sophisticated software tools for storing and managing access to structured data. These commonly implement one of two solutions, namely a relational or object database solution. Since relational databases are more mature, and have better support for storing temporal records, we suggest that the implementation of our current modelling process will use a relational database system.

Temporal data management is crucial to many existing information systems where the history of events and states is required to support user activities. Banking is perhaps the best example, where organisations are critically reliant on historical records of transactions. By adding temporal data storage, and temporal analytics, to our software model we are able to answer the "When?". Such functionality would also allow us to re-play an event in order to discover how it occurred and what were the circumstances surrounding its inception.

At present, however, few modelling systems support the storage and retrieval of temporal data. Perhaps the best example is 'temporal GIS', which allows for the management of temporal geographical data. GIS systems, however, are limited to models that focus on spatial and geographical data sets. We wish to develop a process that is not constrained to only spatial models, but rather any type of modelling scenario that requires dynamic states, temporal data management and analysis. This brings us to the concept of extensibility and domain specificity.

2.4. Extensibility, Domain Specificity and Levels of Abstraction

Extensibility and domain specificity are closely related issues that describe how flexible a modelling standard or process is in terms of whether it can easily be adapted to a previously unplanned purpose or to a new domain. These issues are also closely related to semantic richness with semantics suffering if a standard aims to support a broad range of domains. For instance, CAD tools are generic in that they are able to support a broad range of domains, yet are commonly semantically poor as it is up to individuals to infer the meaning of a set of basic geometric shapes. Conversely, models developed using the IFC object model contain a wealth of information including geometric data, detailed object relationships and properties. However, the IFC object model is constrained to supporting the development of models with the domain of building design. Ideally, in such cases where modelling standards exist but do not fit an organisation's requirement, they should be able to develop their own modelling structure to suit domain needs. Although this is a difficult and time-consuming task, our process may contribute in assisting such efforts by identifying design patterns and a set of activities required for designing semantically rich and dynamic models. In cases where existing standards already exist, it may be found that the effort required to extend the existing format to suit an organisation's need is too great and it may be worth re-inventing the wheel in order to develop something that has a closer fit with requirements. This depends, however, on whether the modelling standard was developed to support future extensions. Whilst the IFC object model claims to be extensible through support of user-specified data structures, and the ability to link those structures to IFC models, the fact that this standard focuses on flat files makes it inherently difficult when attempting to extend it to support dynamic scenes. One of the benefits of our proposed process will be to draw designers' attention to the common problems caused by designing modelling structures that only fit the current needs, and to encourage them to develop structures that are capable of being extended and modified in order to take advantage of information contained in the resulting models.

When looking at existing modelling processes, we found on-going discussion about the usefulness of guides, and processes that aim to be generic and all encompassing. Proponents of domain-specific modelling, such as Kelly and Tolvanen, criticised UML for being overtly generic and thus providing little support for specific problem domains [4 pp.56]. Another important criticism of UML is that it attempts to increase the level of abstraction at the wrong side of the problem, i.e. it focuses on the design of the software and not the domain itself. Our approach to modelling focuses firstly on the problem domain, where the model is converted into software, the model concepts are treated as separate from the various tools and supporting code that interact with the model. UML models, such as class diagrams, bind objects that represent domain concepts and supplementary objects, which are added to perform some functionality that is deemed as valuable by designers. We believe that these two issues should be treated separately in order to support stakeholder communication and to allow for a firm boundary to separate the bounds of the model representing the domain from issues specific to systems development.

From our perspective, the approach taken by UML is also unhelpful in developing modelling languages, due to its focus on developing software-related artefacts rather than models describing problem domains using a complex notation. In fact the majority of popular sources surrounding UML are focussed around the meaning of the notation rather than its usage when applied to a particular problem. Researchers such as Guizzardi have shown that not only are the semantics of UML unclear [5] but in many cases are ontologically incorrect based on currently accepted theories in cognitive science, philosophy and ontology design [6]. LINK TO VISUALISATION ?

2.5. Visualisation, User Interface and Interaction

In fields that value spatial and structural models, particularly those relating to product design, the ability to visualise model information is crucial. As discussed, typical CAD products are capable of little more than supporting the drawing and visualisation of designs. In such cases it is common for modelling standards, technologies and tools to avoid semantic mark-up and focus purely on the ability to create and view objects in the virtual space. An important example concerning failure of current methods to support semantics is the world wide web. Since its inception and growth in the mid '90s, only very recently has business realised the potential benefit of adding semantic mark-up to web-based content. In its current form, the world wide web is only human-readable. As a result machines are unable to distinguish between different types of content. This has resulted in overtly complex systems for web searching, including the use of keyword and hyperlink analysis. Technologies currently being developed relating to the semantic web will reduce search complexity methods, make search results more accurate, and specifically support enhanced context-aware functionality.

We have defined visualisation as the last and least important of our five factors. However, whilst we believe that there is more value in relational semantic mark-up, we cannot ignore the fact that models must be perceived and understood by people. As a result, it is important to design systems with appropriate user interfaces, interactive capabilities and visualisation support the communication and usage of information contained within software-based models. The particular choice of user interface and visualisation depends very much on the intention of the model. For instance, models that record the occurrence of events within a domain where spatial information is irrelevant may be presented through simple text-based means, whilst product designs will continue to depend on rich 2D and 3D geometric models.

3. Proposed Unified Model Development Process

Our modelling process combines a number of theories into a practical method for developing semantically rich and dynamic models. We intend for it to be used as a sequence of phases that users can follow to progressively build and expand a software-based model. With each phase, the model should become more useful in terms of being capable of supporting more information and respond to more types of queries, as based on our previous discussion of the who, what, when, where, why and how questions. Before we discuss the process, however, we must first provide a theoretical foundation for the principles that have guided the development of our process. In the following sections we describe the background theories and practical software-related implementation techniques that we believe are optimal for supporting model development, and provision of a justification for why we have avoided alternative approaches.

3.1. Driving Principles

Based on current practice and literature we found that the following modelling paradigms lead to high quality, semantic and extensible modelling solutions:

- A domain's static semantics may be fully expressed using entities, attributes and relationships as commonly used in ontology design and conceptual data modelling.
- Dynamic models may be described through the use of object-oriented systems development methods whereby objects represent domain entities, their properties are represented by object members and relationships are described through objects representing the relevant relationships.
- Through an analysis of current modelling standards, processes and technologies, we may identify the methods or technologies that should be sought / avoided in order to ensure that future standards are extensible in terms of support for rich semantics, dynamic behaviour, time and visualisation.
- Rich visualisations are based on underlying relational model semantics. As a result, relational semantics should be identified and specified first rather than developing a visually focussed modelling language and attempting to add relational semantics later (as common in current CAD systems).

In the following sections we will go into further detail in explaining the process of fulfilling the requirements for each of our five phases of model development and justify our choice of driving principles.

3.2. Supporting Rich Model Semantics

Our first port of call when developing semantically rich software-based models is to discuss model semantics in a static context. The process of defining a domain's semantics typically begins with the process of understanding the domain to be modelled. One of the results of this process is often referred to as a domain ontology or conceptual domain model. Such models consist of a graph of entities and relationships, which in more detailed views include domain attributes. Domain concepts relate to the types of thing that are of interest within a particular domain. For instance, within the domain of architecture, walls, windows and doors would be identified as concepts of interest to be included in the domain ontology. Relationships refer to the types of connections or links that may exist between concept types. For instance, doors may be related to walls in order to signify that a door is within a piece of wall. Attributes relate to the types of properties that are of interest for particular entities or relationships. Attributes in terms of building concepts may include an object's cost, material, weight or supplier. These three concepts form the heart of modern relational database systems and provide us with a very powerful method for modelling domain semantics. The most popular type of conceptual data model to take is Chen's Entity-Relationship model [7]. This model allows for the description of a conceptual data model for a domain that may then be implemented within a relational database. Consequently we advocate the use of the standard ontology development process that can then be converted into an entity-relationship diagram for implementation within a relational database.

Within this paradigm we are able to treat the types of 'thing' that may be said to exist and be of interest to a particular domain to fit within the meta-ontology (an ontology that uses a higher level of abstraction in order to describe a wide range of domains) that contains entities, attributes and relationships. This fits well with existing modelling standards that focus on rich semantic, such as the IFC object model, which defined a set of objects with associated attributes and relationships.

Whilst we agree with current ontology and conceptual data model design guidelines, we find them to be insufficiently detailed, yet overly complex, to be successfully applied by novices. Basic guidelines, such as those provided by Noy and McGuinness [8], describes the ontology development process in terms of analysing the language used in the domain and picking out the important concepts, the characteristics that are commonly associated with the concepts and the relationships that may exist between different concept types. The relationships may either be hierarchical, i.e specifying that a particular concept is either a specialisation (if it inherits the general concept's properties and adds further properties or restrictions of its own) or a generalisation of another concept or the identification of relationships between concepts that exist within the domain. Such guidelines, whilst useful, are insufficiently detailed in supporting amateur model designers in the making of important design decisions. As shown by researchers, such as Guizzardi [thesis], there are numerous important issues, patterns and guidelines that should be taken into account when designing conceptual domain models. One simple example involves the modelling of the distinction between entities based on genuine concepts, the instances of which may be individually counted and uniquely identified (i.e. sortal universals) and the roles that they may play in particular concepts. Novice designers commonly mistake genuine sortals for role types, by creating a 'customer' and an 'employee' entity for an online retailer whereby both entities bind the properties that would normally be associated with the sortal 'person'. Whilst this initially appears to be a minor error, the problem become apparent when an individual becomes both an employee of the company and a customer. Such errors would force the introduction of false data into the system, i.e. the creation of two separate records, signifying two separate individuals, which in reality referred to the same person. As a result of these findings, we believe that the aggregation of design patterns, expert advice and common mistakes that should be avoided should be aggregated into our process in order to support novice designers in developing high quality conceptual data models of their target domains.

3.3. Adding Dynamic States and Behaviour

The conversion of an implemented database based on the previously defined conceptual model should be supplemented by the development of an object-oriented system that mirrors the domain concepts, entities and relationships in the form of a network of objects. This phase should be followed with the addition of temporal data features as discussed in the following two sections.

Implementing the Model as a Network of Related Software Objects

To increase the usefulness and semantic richness of our model we must facilitate both object states that can to change over time and object behaviour. This phase involves the development of a system based on the object-oriented paradigm and associated technologies (such as Java or Microsoft .NET). In order to support dynamic states we must first create objects based around the domain concepts recognised in the previous phase. This should consist of a simple mapping so that domain concepts, including entities and relationships, are represented as objects, and the attributes of these objects are represented by object members. Unfortunately, object-oriented languages do not have an explicit concept of a 'relationship' and as such relationships are only implied through references to other objects, e.g. an 'arraylist' containing a set of strings. This unfortunate oversight requires that

we specify concepts represented by relationships as a special object type. The work of Wren [9] is particularly useful to overcoming this problem as he identified a number of design patterns for creating objects that represent inter-object relationships.

Assuming a hierarchy of concepts that describes the sub-class / super-class relationships has been defined, we should also use object-oriented language features to implement the hierarchical class relationships. It is important to note at this stage that we are focussing on the specification of objects from the perspective of the domain and that the developed objects should be based on a close resemblance to the previously specified domain conceptualisation. The creation of objects prepares us for the addition of object behaviour through the specification of events that are of interest within the domain, including their causes, effects and any associated rules.

Adding Events, Causes, Effects and Rules

To support dynamic modelling, we focus on the identification of events or actions that take place in the domain, and their associated causes and effects. By causes we refer to types of pre-conditions, or prior events, that cause a certain event to occur. These causes differ between different domains and there are important distinctions to be made based on the purpose of the model. For instance, in models relating to the social world, developed in the context of organisational semiotics, all events must be attributed to an actor, typically a person or an organisation, which takes responsibility for the event. In physical models, however, the motion of physical forces or masses may cause an event, e.g. gravity causing objects to fall that change the states of other objects. In either case, we must be able to trace the cause of events to be able to answer 'who' or 'what' was responsible for the situation taking place. Equally, we are also interested in mapping the effects of events. For instance, the application of a specific force in a certain direction to an object of a certain weight and shape may result in movement of a certain type, which we can be represented as a change in state in the object's associated location. This kind of modelling/mapping fits well with UML's behavioural modelling diagrams and associated notations, such as use case models, activity diagrams, state machine diagrams and sequence diagrams [].

3.4. Adding Temporal Data Management and Analysis Support

As previously discussed there are few modelling systems that cater for the storage or analysis of temporal data due to the lack of standards, and the complexity of implementing such features. However, the complexity of implementing temporal data support varies widely depending on the specific requirements. For instance, in models that are driven by reliable, real-time sources, such as sensors within a building, it is fairly simple to develop a solution that is capable of recording sensor state changes as they occur and present this information as feedback. In less reliable cases we require that model users are able to retrospectively modify historical data. This adds a significant level of complexity and may reduce overall system performance as a number of tests and queries must be executed in order to ensure that the user's request leaves the model in a consistent state with respect of time e.g. the time line of the object's history must remain continuous and with respect to other constraints placed on the model. For instance, how should the system cope with a retrospective update that causes a change in the flow of events that caused later events to take place?

Although progress in terms of standards and relational database product development is slow, there are numerous pieces of research that serve as excellent sources of practical information for supporting the implementation of time support within database systems. Two of the best information sources and guides, which describe the complexities of creating and managing database structures, to support temporal data is the work of Jensen [10] and Snodgrass [11]. Our decision of segregating the phases of conceptual database design from the addition of temporal data may appear counterintuitive, since most software development projects design and implement the database in its entirety before moving on to code development. However, this method is based on the advice of Snodgrass [11 pp. 344] who argues that the two phases should be kept entirely separate, and that the addition of temporal data support should be added only after the non-temporal conceptual model has been fully designed.

The reason we have included this phase after developing the dynamic object-oriented part of the model, is that it is common practice in such systems to develop an object to act as the database API and forward calls to the database. After making the necessary changes to support time support to the database, only the API needs to be modified in order to make the necessary tests and updates to the database in order to support temporal data storage. This method also allows for the fine-tuning of object and database structures before the temporal aspects are applied in order to reduce later modifications, which are more complex due to the presence of structure and algorithms relating to time-varying data.

3.5. Model Extensibility

As previously discussed, it is important to develop modelling systems so that information contained by model instances may be used for alternative purposes. By opting for a relational temporal database, with an object-oriented solution, we enhance extensibility of the developed system in a number of ways. Firstly, by treating the model data as dynamic, we are able to make use of the real-time model in a dynamic context if needed, e.g. for simulation modelling, yet can still take a 'snapshot' of the model in order to make use of the information in a static context. Secondly, by adding temporal support we are able to analyse the timeline of events and state specifically what took place during model's execution. By using object-oriented development methods we are able to support the modification and extension of modelling artefacts by defining publicly available components, and the inclusion of interfaces and APIs allowing third-party applications to interact with the model. Storing the data in a database also allows for simple and fast querying, multiple simultaneous user support, conversion to other formats (as required), and integration with other tools that make use of database content.

3.6. Visualisation

We have left the issue of visualisation as the last area of interest since existing standards clearly focus too heavily on rich visualisations, whilst sacrificing relational semantics. Nonetheless, models representing physical object are of little use unless they can be perceived and correctly interpreted by people. By storing information through a database that is initially set up to include rich relational semantics, we are able to provide equally rich visualisation tools that allow the user to augment their view of the model, hide certain types of information and discern between different types of objects, relationships and states through colour-coded diagrams. Whilst converting model data to a graph-based depiction is a fairly simple task, as open languages and tools exist for these purpose, such as the excellent .dot graph description language [12], the integration of relational model semantics with CAD data is more complex due in part to the proprietary complexity of CAD formats, but mostly because there is no easy means of mapping CAD documents with a semantically rich relational information.

4. Conclusion

In this paper we have discussed and justified five key factors, which should be considered when designing and deploying software systems to support modelling. Based on these factors we have developed a modelling process that focuses on closely reflecting the semantics and dynamic behaviour of the target domain, rather than becoming immediately mired in the complexity of software design as is common with many existing methods. We have provided and discussed the founding theories and principles that are crucial in developing semantically rich models and have provided examples of how ignoring these factors can lead to sub-optimal or non-extensible solutions. Our approach has assessed the problem of extensibility and has shown that, using the prosed technologies and processes (see figure 1), we will be able to avoid many problems currently being experienced as a result of the current lack of relational semantics, dynamic behaviour and support for temporal data storage as found in many current modelling standards and languages.

Our future work will expand a detailed specification of the process, and evaluation through application to a number of case studies. In addition, we intend to consider the semantic linking of information from various domain models, which would be extremely valuable for analysis and visualisations, as currently shown in GIS systems which bind information sources based on location. The potential of this research domain is considerable, and we hope that this work will help stimulate further interest in the development and adoption of semantically rich, dynamic and temporal models.

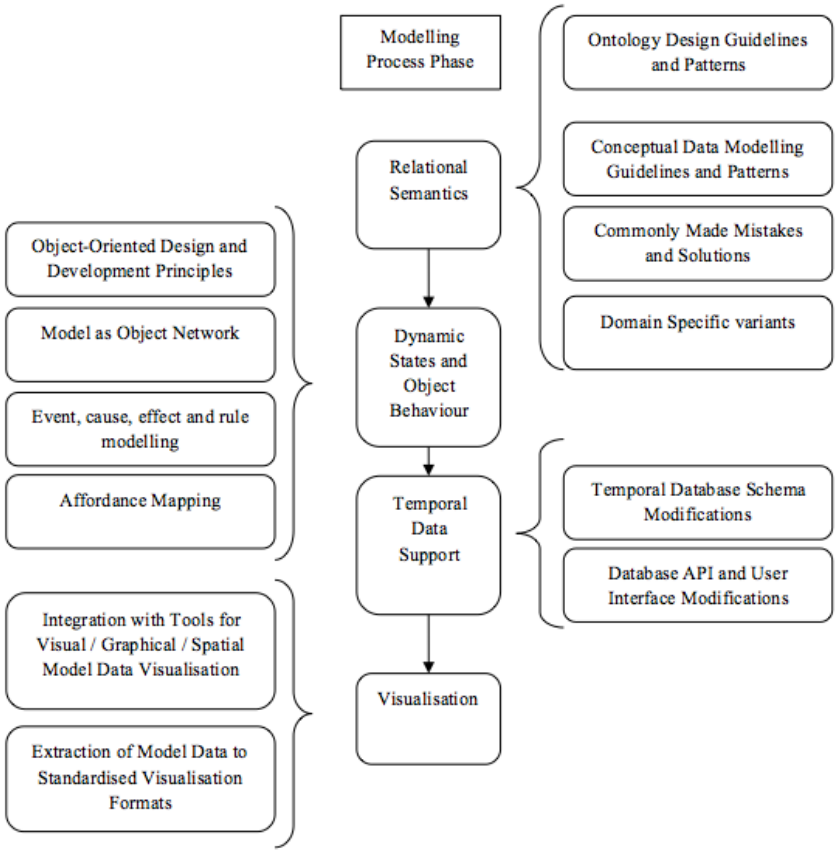

Figure 1: Abstract Modelling Process Phase

References

[1] Krogstie J, Sindre G, Jorgense H, "Process models representing knowledge for action: a revised quality framework", European Journal of Information Systems (2006), pp 91-102

[2] Spearpoint, M.J. (2003) Integrating the IFC building product model with fire zone models. QUT, Gardens Point Campus, Brisbane, Australia: International Conference on Building Fire Safety, 20-21 Nov 2003. 56-66.[Online] Available: http://ir.canterbury.ac.nz/bitstream/10092/486/1/12589008_Integrating%20the%20IFC%20building%20product%20model%20with%20zone%20fire%20simulation%20software.pdf (Accessed 9.2.2011) Sowa J, "Semantic Networks" 1992 [Online] Available: http://www.jfsowa.com/pubs/semnet.htm

[3] Kelly S., Tolvanen J.P., "Domain-Specific Modeling: Enabling Full Code Generation"

[4] Guizzardi, G., Herre, H., Wagner G.: Towards Ontological Foundations for UML Conceptual Models. 1st International Conference on Ontologies, Databases and Application of Semantics (ODBASE'02), Lecture Notes in Computer Science, Vol. 2519, Springer-Verlag, Berlin (2002) 1100-1117

[5] Guizzardi G., "Ontological Foundations for Structural Conceptual Models", 2005

[6] Chen, P. P. 1976. The entity-relationship model-Toward a unified view of data. ACM Trans.Database Syst. 1, 1 (Mar.), 9-36.

[7] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report SMI-2001-0880, Stanford Medical Informatics, 2001.

[8] Wren A., "Relationships for object-oriented programming languages, 2007, [Online] Available: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-702.pdf (Accessed: 8.2.2011)

[9] Jensen C. J., "Temporal Database Management", Defended April 2000, [Online] Available: http://www.cs.aau.dk/~csj/Thesis/

[10] R. T. Snodgrass. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, 2000.

[11] Graphviz Graph Visualisation Software: The DOT Language Documentation [Online] Available: http://www.graphviz.org/doc/info/lang.html

[2] Spearpoint, M.J. (2003) Integrating the IFC building product model with fire zone models. QUT, Gardens Point Campus, Brisbane, Australia: International Conference on Building Fire Safety, 20-21 Nov 2003. 56-66.[Online] Available: http://ir.canterbury.ac.nz/bitstream/10092/486/1/12589008_Integrating%20the%20IFC%20building%20product%20model%20with%20zone%20fire%20simulation%20software.pdf (Accessed 9.2.2011)
[3] Sowa J, "Semantic Networks" 1992 [Online] Available: http://www.jfsowa.com/pubs/semnet.htm
[5] Guizzardi, G., Herre, H., Wagner G.: Towards Ontological Foundations for UML Conceptual Models. 1st International Conference on Ontologies, Databases and Application of Semantics (ODBASE'02), Lecture Notes in Computer Science, Vol. 2519, Springer-Verlag, Berlin (2002) 1100-1117
[6] Guizzardi G., "Ontological Foundations for Structural Conceptual Models", 2005
[7] Chen, P. P. 1976. The entity-relationship model-Toward a unified view of data. ACM Trans.Database Syst. 1, 1 (Mar.), 9-36.
[8] N. F. Noy and D. L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report SMI-2001-0880, Stanford Medical Informatics, 2001.