19th International Congress on Modelling and Simulation, Perth, Australia, 12–16 December 2011
http://mssanz.org.au/modsim2011

Improving water information management through Information Modelling

R.M. Argent[a], R.A. Maguire[b], W. Slade[c], R. van Luinen[c], and R. Sjogren[c]

[a]*Climate and Water IT Services Branch, Bureau of Meteorology, Melbourne*
[b]*Climate and Water IT Services Branch, Bureau of Meteorology, Canberra*
[c] *SMS Management and Technology, Melbourne*
Email: R.Argent@bom.gov.au

Abstract:

The Bureau of Meteorology has responsibility for collecting, holding, managing, interpreting and disseminating Australia's water information. Given this mandate, the Bureau is developing the Australian Water Resources Information System (AWRIS) to receive and manage water data and related information, and to support the production and dissemination of a variety of water information 'products' and services. Traditionally, as business needs for data and information changed and developed over time, it would be necessary to re-design data models, databases, analysis tools and products to ensure continuity in data and information supply. An 'information modelling' approach that aims to address long term maintenance and evolution is being trialled as part of development of AWRIS. This approach uses Model Driven Architecture/ Model Driven Development (MDA/MDD) within an enterprise architecture framework to maximise reusability, scalability, platform independence, speed-of-delivery, change control and governance. The approach entails a philosophy and method of development, and a tool (Sparx Enterprise Architect) for translating formal platform-independent models to platform dependent scripts, schemas or code. In its most ideal form, MDA/MDD offers:

- A formal and constrained approach to specification of business needs, in the form of models written in UML
- Integrated documentation as the model is loaded and described using Enterprise Architect
- Rapid and agile development, with automated transformation of models into operational schemas, services and/or code
- Increased collaboration between domain and technical experts, through clearer sharing of needs, more common language for communication, faster turnaround on changes, and better feedback

In practice there are a number of challenges to adoption this approach, both in concept, governance and operation. Conceptual challenges have been tackled through use of an architectural framework - the Zachman Framework – that provides a matrix structure for relating project drivers, concepts and 'artefacts' (e.g. documents, code). From top to bottom the framework passes from Scope (Context) at the highest level through Business (Concept), System (Logic), Technology (Physics), Component (Assemblies), to Operational (Instantiation) level. For AWRIS the scope might be defined by the requirements of the Water Act 2007, while the operational level might be a data delivery service. Across each level within the framework (e.g. across the 'Technology' level) lie the fundamental communication elements, covering the 'What' (Inventory), 'How' (Process), 'Where' (Network), 'Who' (Organisation), 'When' (Timing) and 'Why' (Motivation). Governance and operation of the MDD/MDA approach have provided challenges that include variations in application due to unfamiliarity with UML and the Zachman framework, model repository governance and change control as use and user numbers expand, and inconsistencies between pre-defined schema, model and documentation, confounding the effectiveness of the transformation from platform independent to platform dependent artefacts.

Keywords: *Information Modelling, AWRIS, Enterprise Architect, Model Driven Architecture, Model Driven Development*

1. INTRODUCTION

The Bureau of Meteorology (Bureau) has responsibility for 'collecting, holding, managing, interpreting and disseminating Australia's water information' (Government of Australia, 2007). Given this mandate, the Bureau is developing the Australian Water Resources Information System (AWRIS) to receive and manage water data and related information, and to support the production and dissemination of a variety of water information 'products' and services, including basic data products, modelling systems and reports such as water resource assessments, water accounts and streamflow forecasts. The raw data for AWRIS includes standardised hydrological geospatial information, and observations of stream level, groundwater, water storage levels, hydrologically-relevant meteorological data, rural and urban water transfers and use, information on water trading, water restrictions, and water quality.

On the input side of AWRIS it is required that data from over 200 organisations be received in a variety of formats from a range of different systems, for a significantly disparate set of observation sensors, and that these data be ingested into a central system. Even for organisations using 'industry standard' time series data management systems, variations in local business needs and system configuration make it difficult to align like with like in the AWRIS central data store. A standard, XML-based 'Water Data Transfer Format' (WDTF) is being used to increase clarity in the meaning of the information received by AWRIS and also to move to a nationally consistent approach to both publishing water data and transferring water data between organisations.

On the output side of the system, each of the water information products is tailored to meet a set of user requirements, and therefore each draws on the underlying data to differing degrees and requires different aggregations and analyses to produce relevant values. For example, delivering nationally consistent information on water storages requires calculation of a standard 'daily value' of water level and volume for each storage from observations recorded anywhere between minutes to multi-week intervals.

Traditionally, as needs for data and information changed and developed over time, it would be necessary to re-design data models and databases, as well as analysis tools and products, to ensure continuity in data and information supply. Additionally, as business needs change and new products are developed or current products are updated, then the underlying data structures also often need to change. This flags a need for information systems development and architecture that permits effective system evolution.

This paper presents an 'information modelling' approach based on research largely undertaken by CSIRO, that aims to address some of these issues of long term maintenance and evolution, as well as contributing to effective development processes. The approach aims for a system that is evolvable and readily maintainable, not only in regards to the data structures but also in regards to the expression of business needs and the relationship between business needs and the code, data tables and services meeting these needs.

2. INFORMATION MODELLING

Information Modelling exists within all IT projects and systems in a wide variety of forms and generally with a wide variation in formality. For example it is not possible to create any form of database without performing some form of Information Modelling, however rudimentary that modelling may be. In some organisations this modelling might be performed by a Data Architect using a formal notation such as UML (Booch et al., 1999) within a modelling software tool, elsewhere this might be performed by a lead developer drawing on a whiteboard, which is then transcribed into Data Definition Language (DDL). Thus, it is not possible to divorce Information Modelling from the process of realising an IT system.

It is possible, however, to determine a desired level of formality for the Information Modelling process and to work towards achieving this within an organisation. Most organisations tend to sit somewhere in the "middle of the road" with respect to the formality of Information Modelling that they use on a day-to-day basis. Typically this level of adoption manifests in some or all of the following:

- Requirements: generally captured in point form, with, say, individually numbered requirements in a large 'requirements document', or as a set of Use Cases. Sometimes both are produced, often with one being a reworking of the other.
- Database models: usually in the form of Entity-Relationship Diagrams which are often translated into database creation scripts automatically.
- Ad-hoc modelling by developers of aspects of the system, most often as UML Class Diagrams and, more rarely, including UML Interaction Diagrams. Usually these diagrams are drawn using diagramming or drawing software rather than using a formal modelling software tool.

- XML Schemas: often hand-crafted and usually poorly annotated or documented.

Lack of, or lack of adherence to, protocols exists to a greater or lesser degree in almost all software development efforts, and is probably familiar to many readers. This can contribute to a number of issues that are not commonly considered:

- Traceability: the links from a specified business requirement through to the system elements that implement that requirement are often unclear, if not completely untraceable. This makes it difficult to manage scope on a project, to know with any degree of certainty when the system is "complete", or even to know if what is being developed meets actual requirements.
- Design artefacts: generally, these are not managed nor maintained once they are created, hence becoming "once-off" pieces of work, which are neither useable nor useful for maintenance or for future design of a similar system.
- Documentation: is of varying quality and often not even representative of the implemented solution since the developed solution has "moved on" since the design documentation was produced.

Moving to a formal Information Modelling approach can address these issues. The framework for such an approach that is being tested by the Bureau is Model Driven Architecture / Model Driven Development (MDA/MDD)(Mellor et al., 2004; Miller and Mukerji, 2003).

3. USING MODEL DRIVEN ARCHITECTURE / MODEL DRIVEN DEVELOPMENT

Model Driven Architecture and Model Driven Development revolve around the concept of first creating platform-independent models that, ideally, formally represent or reflect business needs, and then automatically translating these into platform-specific model elements that perform the required operations. For example, a platform independent data model may be transformed into an Oracle (platform dependent) Data Definition Language script or even directly instantiate a schema on an Oracle server. Another example may be to transform a platform independent Class Diagram into Java code (platform dependent) classes.

In using this approach for AWRIS, the Bureau hopes to be able to define *a priori* the various data models required by the system and the transformations between these models. Defined in this way, the models can then be used to generate software (e.g. XSD, DDL) or used by an interpreter to execute the transformations. Ideally, this would enable people to describe what has to be done without having to worry about how it is to be executed. This has the *potential* to deliver the following benefits:

- Enabler of business involvement - as the models drive the execution of the system, the water information business representatives see a direct benefit of describing their data and transformations as they can see the direct result of their efforts within a short space of time (usually minutes) rather than having to wait weeks as is usually the case.
- Enforce constraints on the representation of a solution - this enables modellers to concentrate on how to put together a solution rather than spending time determining how to represent it in code.
- Documentation forms part of the modelling process and is generated directly from the model. Therefore, development effort is significantly reduced as the one model creates both the various system artefacts and the documentation describing them.
- Speed of development - as the models are interpreted directly, there is no time spent in transcribing the models into working code. This cuts down substantially on development and testing effort as, once a pattern has been developed and tested, it doesn't have to be developed and tested again.
- Promotion of Agile development practices - schema and transformation changes can be readily incorporated into a build as development and testing is effectively automated. Turnaround from requirements (e.g. the models) to a functioning system can be done quickly so as to enable paired analysis and the ability to trial different solutions in a short space of time to determine which one works best. They can also iteratively and incrementally build up a solution as they solidify their thinking based on results that they can see.

By capturing business requirements and models in a formal modelling environment (ie with both tools and notation), modelling processes can then occur that show the derivation of system-level models while concurrently maintaining traceability back to the business-level models. The business-level models and

requirements can also become the common language employed for exchanges between the domain experts (aka subject matter experts) on one hand, and by the IT Architects and Developers on the other.

Two key elements in the Bureau's adoption of an Information Modelling approach have been the use of a formal framework for linking between business needs and the resulting operational code, and the use of technological enablers to provide required features for control and automation of processes.

3.1. Enterprise Architecture Framework

The development of AWRIS has employed the Zachman Enterprise Architecture framework (Figure 1) (Zachman, 1987). The Zachman framework provides for an ontology of system artefacts from the highest level scope (e.g. the requirements of The Water Act 2007) down to operational level components such as application servers and deployed executable software.

	Why	How	What	Who	Where	When
Contextual	Goal List	Process List	Material List	Organizational Unit & Role List	Geographical Locations List	Event List
Conceptual	Goal Relationship	Process Model	Entity Relationship Model	Organizational Unit & Role Rel. Model	Locations Model	Event Model
Logical	Rules Diagram	Process Diagram	Data Model Diagram	Role relationship Diagram	Locations Diagram	Event Diagram
Physical	Rules Specification	Process Function Specification	Data Entity Specification	Role Specification	Location Specification	Event Specification
Detailed	Rules Details	Process Details	Data Details	Role Details	Location details	Event Details

Figure 1. The Zachman™ Enterprise Architecture framework

The Zachman framework is not a prescriptive methodology – it shows where to put artefacts that are created rather than prescribing how they should be created. Classification of artefacts is done according to a two dimensional matrix:

1. The columns describe the fundamentals of communication: i.e. the 'Why' (Motivation), 'How' (Process), 'What' (Inventory), 'Who' (Organisation), 'Where' (Network) and 'When' (Timing).
2. The rows describe the 'reification' of artefacts, starting with Scope (Contextual) at the highest level and descending through Business (Conceptual), System (Logical), Technology (Physical), to the Operational (Detailed) level.

For a given cell in the framework matrix, each artefact is 'aware' of all of it's neighbours across the row (ie all 'communication' aspects), while within a column it is only aware (derives from or contributes to) its immediate neighbour above and below. The concept of refinement through the reification layers provides for business view alignment and linking to system implementation.

The Zachman framework is undergoing testing as part of the AWRIS development,, using the software application 'Sparx Enterprise Architect' as the collaboration space for modelling. A Zachman framework exists for the AWRIS system as a whole. Each project creates a Zachman framework specific to the project and, ideally, project artefacts are merged into the AWRIS Zachman framework at the conclusion of the project.

3.2. Model Driven Architecture Enablers

Whilst the Zachman framework provides a classification of system and project artefacts, there exists the need for a set of enabling tools for the development and creation of the artefacts themselves. The application of integrated software tooling allows for the eventual generation of actual implementation artefacts such as

XML schemas, database creation scripts, code files, configuration information, and deployment packages. This capability has the potential to dramatically enhance the ability of an IT team to quickly produce reasonably large parts of a system automatically while maintaining traceability and documentation aspects of that system. Development of AWRIS and associated water information products employs, or aims to employ in future, the following enablers:

- Sparx Enterprise Architect – a collaborative modelling workspace using UML notation.
- HollowWorld (Cox and Atkinson, 2009) – an Enterprise Architect plugin developed by CSIRO that can be used to transform models into conformant GML schemas from a set of predefined templates.
- SolidGround (Francis and Atkinson, 2009) – an Enterprise Architect plugin developed by CSIRO that can be used to transform models into database schemas based on a defined GML application schema. It also provides a model registry capability.
- FullMoon (Cox, 2009) - an Enterprise Architect plugin developed by CSIRO that can be used to validate defined GML application schemas and to generate conformant XSDs from them.

The aspirational goal for the use of MDA in development of AWRIS is to implement fully functional systems or system changes directly via transformation of the models.

4. TESTING THE APPROACH FOR AWRIS

A central model repository has been setup for all members of the water information development teams. This central repository allows members to collaborate and share modelling artefacts in a managed manner. During an earlier phase of AWRIS development, only the data architect had access to a local repository and as a consequence only a small number of elements of the AWRIS architecture were formally modelled. Within two months of the central repository being setup over twenty team members were contributing requirements, architecture and design across four different projects. Broader adoption of a formal representation of knowledge will ultimately provide the Bureau with a valuable asset that will inform new and existing members for many years to come.

The adoption of the Zachman framework has directed members to separate their thoughts and ideas into conceptual layers of Scope, Business, System, Technology, Component, and Operations. By being required to give thought to what the business *does* as opposed to what the system *should do*, better requirements are now being drawn from the solution and this provides a true justification of why or why not a system may be required to implement these requirements. Using formal tools to define the requirements is also increasing the traceability and structure of the requirements.

Conceptual models provide an understanding of the architecture and design without polluting the design with implementation details. WDTF was previously only defined as an XSD schema. This component has now been reverse engineered to produce a technology model and has been registered in the central repository. As a technology model the relationships between the elements of WDTF can now be visually inspected. However there are many relationships that are not formally described in the XSD schema, hence it is not possible to see these relationships and their existence is only understood by user guides in the WDTF documentation package. To resolve this issue and to formally define the relationships a conceptual model has been reverse engineered. This model for the first time describes the relationships between the various WDTF elements, although inconsistencies between schema, model and documentation have confounded this approach.

4.1. Pilots

Model driven architecture was piloted in an early phase of AWRIS development with technology models defined for both database and transfer models. From both these models, physical DDL and XSD schemas were produced using forward engineering tools in Enterprise Architect. Use of forward engineering tools ensured that the logical model and the physical model were synchronised and reduced the workload for architecture and development teams. In the past the data architect would model the database and provide a print out to the development team who would then hand-write the database scripts. This process was time consuming and error-prone. Due to long turn-around time, modifications were often performed on the scripts directly and the changes were not reflected in the logical model, with obvious consequences for authoritativeness. Now, as the Data Architect changes the model, the changes are automatically produced to the scripts.

Some of the challenges encountered with testing the MDA/MDD approach in AWRIS have been:

- Varying levels of skill and experience with both UML and the Zachman framework, resulting in variation in the quality and completeness of work undertaken.
- Complex software service design, arising from interpretation of the requirements of MDA in application to loading WDTF-formatted data into a database, potentially converting a low level ETL coding or configuration task into a high level formal UML modelling task.
- Inconsistencies between the documented WDTF standard, the physical WDTF XML schema, and the WDTF model that has been reverse engineered from the schema.
- Static forward-engineering, where, for example, a database schema was generated directly from a model, and the execution of the DDL and subsequent tuning of database parameters was performed by Database Administrators external to the modelling workspace.
- Governance of the model repository, models and framework has not been fully articulated, which in turn leads to implicit knowledge that is not explicitly captured in all circumstances.
- Change control within the repository. This has been attempted using the Subversion tool but has not been completely successful.

5. CONTINUING WITH THE MDA/MDD APPROACH

The initial foray into Model Driven Architecture has determined that this approach has some merit and is worth exploring further.

5.1. Immediate Consequences of Adoption of Information Modelling

As with any other technology or methodology, adoption of a particular approach has many implications. The most relevant ones for this approach are:

- All staff must be trained in how to model and how to use the chosen modelling tool(s) effectively. The level of training required by a staff member varies with their roles.
- Cultural changes within the organisation must be understood and effected. For a Model Driven approach to work effectively, everyone involved, and all stakeholders and sponsors must be "on-board" with the approach and must abide by the relevant processes and rules. Once such rule is: "if it is not in the model then it does not exist and it won't be built".
- Development and project management approaches must take into consideration the methodology. Deriving large parts of a system automatically from models obviously requires that the models exist. This has a tendency to raise the importance of and increase the focus on initial design activities since developers cannot start until sufficient modelling work has occurred. This is not to say that the models must be completely accurate and fully designed before work can start – this would lead to a modelling form of analysis paralysis – but it does mean that certain well-defined parts of the model need to be in place.
- A new role for a "toolsmith" appears necessary. Current MDA / MDD tools and approaches provide varying amounts of support for implementing model transformations and generation capabilities. Enterprise Architect provides reasonable support for a limited style of template-based model transformation and code generation. Much more sophisticated transformations are possible but require the development of EA-specific add-ins. Examples of this approach include the SolidGround MDA tools. To properly leverage the capabilities of an MDA / MDD approach, one or more people in the role of toolsmith are needed to create and maintain templates and tools.
- Pattern identification and development might also be required. Often all of the possible patterns amenable to model driven automation approaches cannot be determined prior to actually commencing system development.. Some patterns only emerge after multiple systems are under development. Such emergent patterns should be identified as early as possible, modelled in the modelling tool, stored as a pattern and model driven tooling put in place. This enables reuse at levels higher than just code or component level reuse.

5.2. Future Advantages and Consequences of this Approach

Choosing a Model Driven approach to systems development has many advantages, and of course consequences. As noted previously, consequences include training, cultural change, and changes to the approach taken to develop systems, as well as overheads in tool maintenance.

An important and often overlooked consequence is that of governance. With an MDA / MDD approach, changing models has direct impacts on existing systems. An example of this is the Model Registry that is being used by two current AWRIS projects and soon to be used by a third project. Changes to models published to the Model Registry can, if appropriate governance is not in place, cause unintended changes to operational systems. Governance processes are required to ensure systems are not inappropriately or unintentionally modified.

However, these consequences are generally balanced by the advantages of the MDA / MDD approach, provided that it is well applied to appropriate components of development. Maintenance and maintainability of systems is enhanced in many areas, primarily due to the automated generation capabilities of the approach. New services required by a system can be modelled and have the framework implementation and schemas generated quickly via transformation and code generation templates (in EA). The framework can be "filled out" by a developer and the new service can be made ready for use quite quickly. The service is already documented because it is modelled.

Development becomes less focussed on implementing large amounts of boiler-plate code and smaller amounts of business-critical code and more focussed on just the business-critical aspects. Provided issues of model propagation and management can be ironed out, fewer developers are required to code the same amount of business-critical functionality. Defect rates also drop since certain types of copy-and-paste and transcription errors are eliminated by code generation.

Overall, the information modelling approach is seen to offer significant advantages for product generation once initial adoption and familiarisation hurdles are overcome.

ACKNOWLEDGMENTS

The authors acknowledge the significant assistance of members of the CSIRO 'sustainable water information systems' research team in adoption of MDA and the Enterprise Architect toolsets.

REFERENCES

Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *The unified modeling language user guide*. Reading: Addison-Wesley.

Cox, S. J. D. (2009). FullMoon XML Processing Framework Retrieved 10/08/2011, 2011, from http://projects.arcs.org.au/trac/fullmoon/wiki/FullMoon

Cox, S. J. D., and Atkinson, R. (2009, 27/08/2009). Hollow World: a GML application schema template Retrieved 10/08/2011, 2011, from https://www.seegrid.csiro.au/wiki/AppSchemas/HollowWorld

Francis, W., and Atkinson, R. (2009, 22/12/2009). Solid Ground Retrieved 10/08/2011, 2011, from https://www.seegrid.csiro.au/wiki/AppSchemas/SolidGround

Government of Australia (2007). *Water Act 2007*.

Mellor, S. J., Scott, K., Uhl, A., and Weise, D. (2004). *MDA distilled: principles of model-driven architecture*: Addison-Wesley Professional.

Miller, J., and Mukerji, J. (Eds.). (2003). *MDA Guide Version 1.0.1*: OMG.

Zachman, J. A. (1987). A framework for information systems architecture. *IBM Systems Journal, 26*(3), 276-292.