Software and Systems Modeling manuscript No.
(will be inserted by the editor)

Model-Based Tool Support for Tactical Data Links: An Experience Report from the Defence Domain

Suraj Ajit[1], Chris Holmes[2], Julian Johnson[3], Dimitrios S. Kolovos[4], Richard F. Paige[4]

[1] Department of Computer Science, Northampton University, e-mail: suraj.ajit@northampton.ac.uk
[2] CGI IT (UK), e-mail: chris.holmes@cgi.com
[3] BAE Systems, Advanced Technology Centre, e-mail: julian.johnson@baesystems.com
[4] Department of Computer Science, The University of York, e-mail: {dimitris.kolovos, richard.paige}@york.ac.uk

Received: date / Revised version: date

Abstract The Tactical Data Link (TDL) allows the exchange of information between cooperating platforms as part of an integrated Command and Control (C^2) system. Information exchange is facilitated by adherence to a complex, message-based protocol defined by document-centric standards. In this paper we report on a recent body of work investigating migration from a document-centric to a model-centric approach within the context of the TDL domain, motivated by a desire to achieve a positive Return on Investment (RoI). The model-centric approach makes use of the Epsilon technology stack and provides a significant improvement to both the level of abstraction and rigour of the network design. It is checkable by a machine and, by virtue of an MDA-like approach to the separation of domains and model transformation between domains, is open to integration with other models to support more complex workflows, such as by providing the results of interoperability analyses in human-readable domain-specific reports conforming to an accepted standard.

Key words Tactical Data Link, Model-Based Development, Interoperability, Metamodelling, Model Management, Eclipse Modeling Framework, Epsilon.

1 Introduction

The Advanced Technology Centre of BAE Systems has been undertaking research relating to the modelling of Tactical Data Links (TDLs) since mid-2005. Although the primary focus of the work has been the Link 16 TDL (sometimes referred to as TADIL-J) as described by MIL-STD-6016C[1] or NATO Standardization Agreement (STANAG) 5516[2], we believe elements of the research to be applicable to TDLs in general. Basing our work on Link 16 is pragmatic insofar as it has the benefit of being a (largely) general purpose TDL, used across a number of air and sea-based assets developed by BAE Systems. For the purposes of this paper we assume both the US military standard and NATO (STANAG) documents to be equivalent. The reader is referred to Section 2 for an overview of the TDL domain and the general nature of the overarching standards.

When we first began to investigate the TDL domain we were struck by the highly document-centric approach. The main document responsible for the definition of the Link 16 was vast, in excess of 7000 pages, predominantly prose (and growing). It seemed almost impossible to begin to form a view of the correctness of a platform against the standard, and equally challenging to determine the correctness of the standard itself. Indeed, Zeigler refers to Link 16 as 'the nut to crack' [28]; in his presentation Zeigler identifies a number of shortcomings of MIL-STD-6016C [1] leading to labour-intensive and potentially error prone downstream activities:

- Use of natural language,
- Ambiguous semantics,
- Size,
- Potentially incomplete and inconsistent.

Somewhat ironically, the intention of the TDL is to provide brain-to-brain connectivity using heterogeneous equipment over the link. Clearly, given the shortcomings identified by Zeigler (above), we must appeal for a migration from the current document-centric approach to something that is scalable, checkable by machine and understandable to TDL engineers (preferably in a more accessible form than currently available). One possible solution to this problem is our vision of the migration to a model-centric approach as illustrated in Fig.1.

A major consequence of the lack of machine checkable artefacts in the standard, a large subset of which forms

Send offprint requests to:

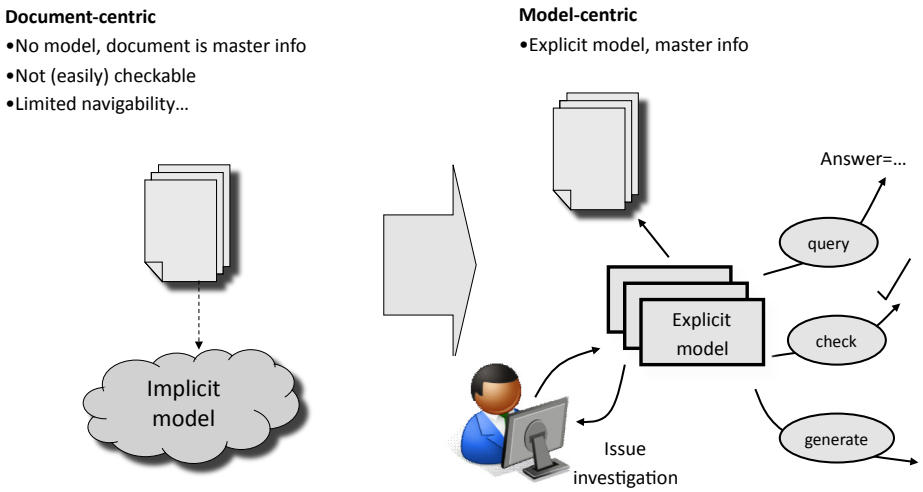


Fig. 1 Migrating from a Document to a Model-Centric Approach

the platform requirements and design documentation, is that consistency and interoperability issues are difficult to identify and may not manifest themselves until late in the life-cycle (when defects are costly to correct). Given Zeigler's reservations (above), adherence to the document-centric standards is no guarantee of success. An absence of views grounded in a rigorous definition of the domain presents a steep learning curve to engineers and the absence of easy to use model-based tools limits the possibility for performing ad-hoc analyses. Furthermore, the absence of a clear set of concepts from which part or all of a standard may be composed serves to militate against consistency and further impacts the learning curve.

In this paper we report on the migration from a document-centric view of platform interoperability to a model-centric view based, primarily, on the use of metamodelling technology provided by the Eclipse Modelling Framework (EMF). The aim of the project was to provide tool support for a labour-intensive and potentially error-prone task requiring significant domain expertise; the complex nature of the problem serves to make the domain somewhat opaque resulting in a long learning curve for engineers. Hence, an implicit objective was the de-skilling of the task by raising the level of abstraction and underpinning the domain with an explicit metamodel. The creation of well-focussed domain models, traceable to the underlying standard also opens-up opportunities for reuse in more profound contexts by allowing the construction of a layered set of interconnected models, potentially spanning the TDL domain from the abstract concepts required to construct the Information Exchange Requirement[1] (IER), to the timeslots in which messages are to be exchanged between cooperating platforms.

A number of peripheral tools were developed to convert different types of documents and artefacts (discussed in Section 2) into a form compatible with our chosen metamodelling technology without manual intervention. We also implemented a number of algorithms operating on our network design model to provide the TDL engineers with a tool to perform the interoperability analyses in a fraction of the time taken previously. An additional benefit was the ability to present the results using an easily understandable domain-specific graphical notation (something the engineers did not produce when performing the analysis manually). This demonstrates the ability to retain specific document-based views of the domain, however these views are now derived directly from the domain model.

Further prototyping has been undertaken to migrate the metamodel to a Rich Client Platform (RCP) plug-in architecture. Whilst not a feature unique to the RCP architecture, this investigation did illustrate the ease with which the model-based approach could be made *active* in the sense that the user becomes able to interact directly with the model, selecting and browsing features of the model, not restricted to simply selecting the objects that are to be fed into interoperability queries. This allows the user to zoom in and out of details of the model, helping to manage potentially large quantities of data.

2 Background

In this section we provide the reader with an introduction to the TDL and describe the problem to be solved using metamodelling and automated model management techniques. A more detailed introduction to the Link 16 TDL is provided in [4].

[1] The IER describes the kinds of information to be exchanged between specific platforms to meet an operational requirement

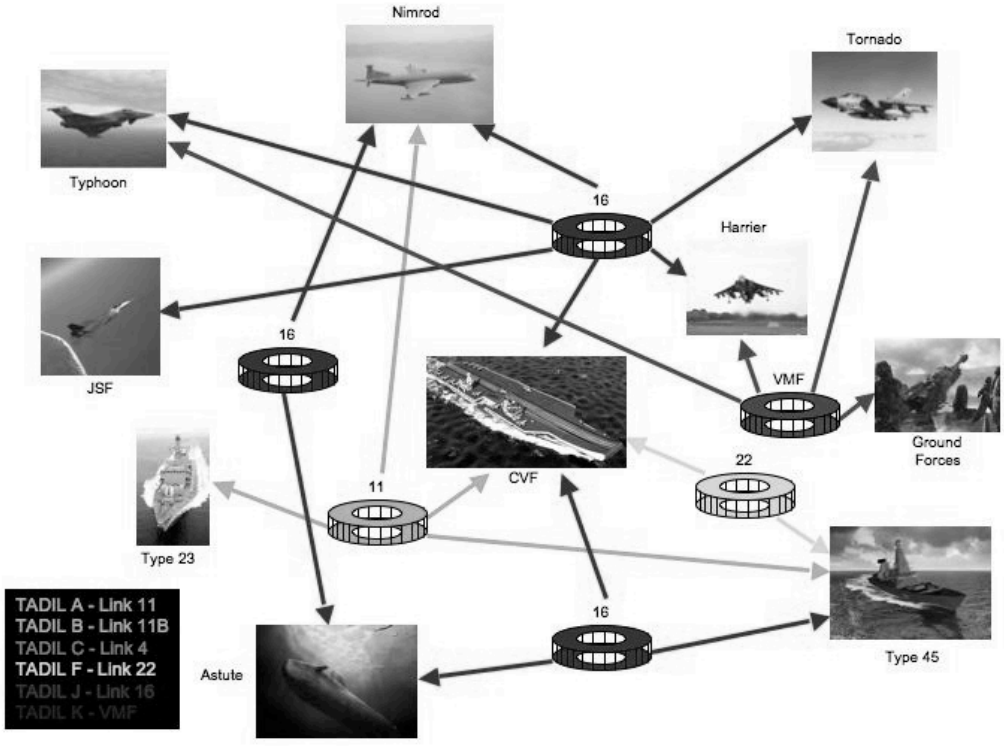


Fig. 2 TDL Overview

2.1 An Introduction to Tactical Data Links

The TDL provides one of the backbone technologies underpinning the defence community's goal of Network Enabled Capability (NEC)/Network Centric Warfare (NCW) across both national and coalition forces by providing the information and infrastructure to afford users with both an integrated picture of the battlefield and also providing tasking orders and responses. A number of TDLs are in service with coalition forces, and are implemented on a variety of assets, such as aircraft, ships, land vehicles, and command stations, an example of which is illustrated in Fig.2. It is possible for TDL platforms to operate on multiple links concurrently (e.g. in the example shown in Fig.2 the (now withdrawn) Nimrod aircraft is shown as communicating on both Link 11 & Link 16) acting as (e.g.) a bridge (known as a Forwarding Unit). Research undertaken by the ATC to date is restricted to single link implementations only.

The Link 16 TDL is a general purpose TDL, in contrast to some others, e.g. Link 4A or Variable Message Format (VMF); a list of data link characteristics is provided elsewhere[3]. Link 16 has evolved over a number of years, stemming from a requirement identified by the US military in the early 1970's for a TDL offering a broad range of functions that would be applicable for use across multiple forces (e.g. Navy, Marines, Air Force, Army, etc.).

2.2 Link 16 Documentation

Much of the Link 16 TDL is described by a large standard in narrative form, combined with many tables and relatively few figures [1,2]. Fig.3 provides an outline view of the TDL standard and its relation to the network design that implements the timeslot map for a domain-specific requirement. We describe each of the core concepts in more detail below.

A Data Dictionary exists at the lowest level of the standard, identifying the set of *types* defined for use on the link, these types are identified by a unique key pair, the Data Field Identifier (DFI) and Data Use Identifier (DUI) pair (referred to as simply the DFI/DUI). Each DFI/DUI may take one of more values as defined via the Data Item (DI); the DI may provide a list of enumeration literals (such as platform types) or a range of values (such as a position in latitude) that may be held by the DI. The set of messages that may be transmitted over the link are defined in the form of a Message Catalogue that, in turn, makes reference to the DFI/DUIs defined by the Data Dictionary, as illustrated in Fig.4.

Messages are tree-structured and functionally-oriented. Once populated, our metamodels may be validated against our defined constraints to confirm well-formedness, e.g. confirming that no bits of a J-Word are undefined. There also exists the possibility of

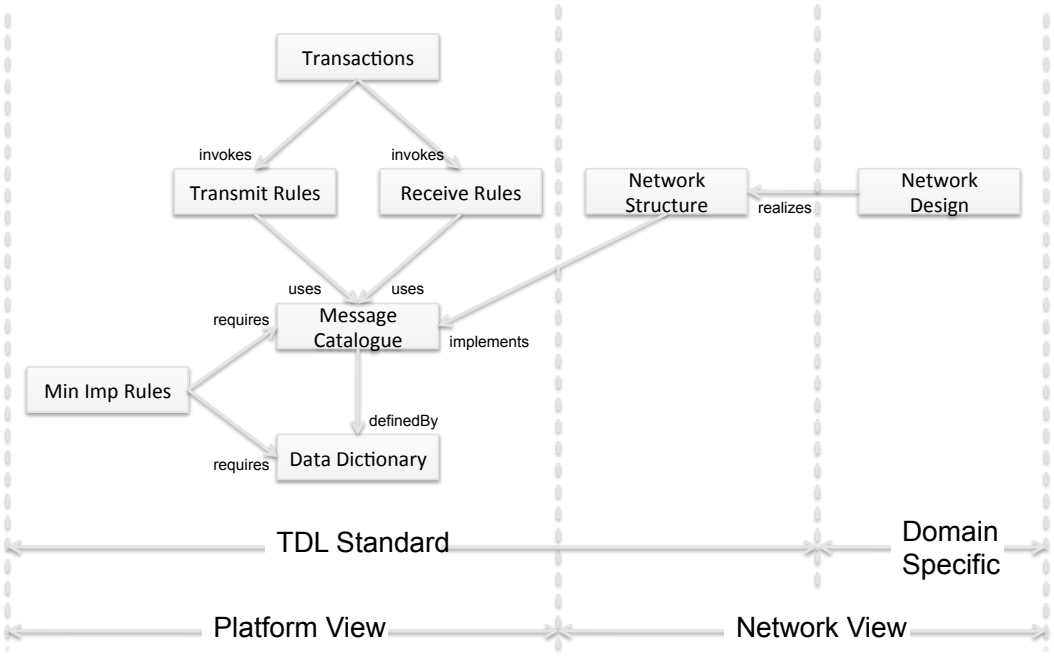


Fig. 3 TDL Standard

transforming the data to other formats, e.g. to allow population of a DOORS database to support requirements capture and traceability. The reader is referred to [16] for a description of our modelling of the Data Dictionary and Message Catalogue.

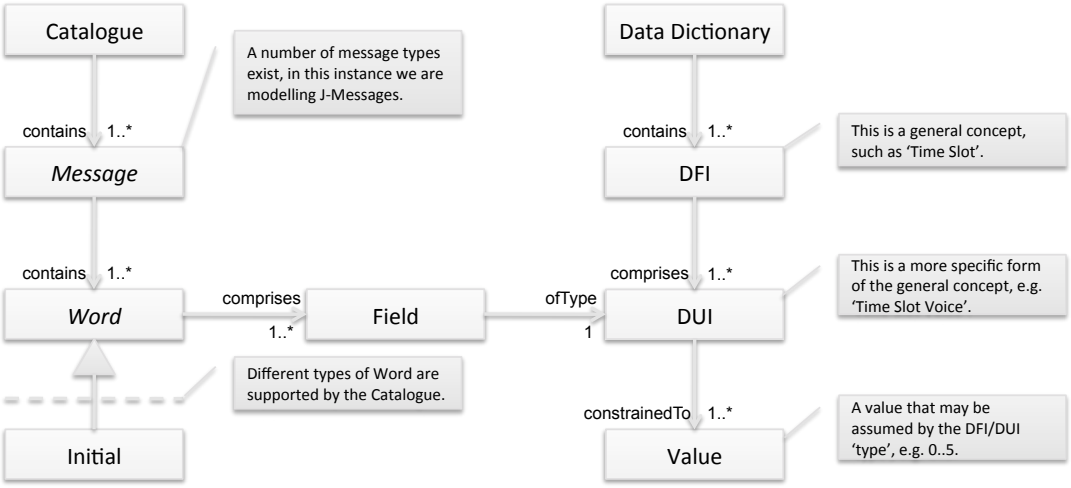
In addition to the message structures, the Standards define the required behaviour associated with the transmission and receipt of a given message. This is described via a combination of natural language and tables via the Transmit and Receive Rules. Once again, we have analysed the source documentation and created a metamodel that is linked to the underlying Message descriptions. Hence, one can see the layered nature of the standard and its explicit representation in our metamodels. The standard also describes the network Time Division Multiple Access (TDMA) architecture and configuration information relating to the timeslots and the Recurrence Rate Number (RRN). This information is provided in narrative form with supporting tables. It is this information that gives meaning to the data provided by the network initialisation (INDE) files (discussed in more detail in Section 2.4) and which we were able to model. Lastly, the standard defines requirements for system participation on the link in the form of minimum implementation (MIN IMP) rules as illustrated by the structures on the left and right hand sides of Fig.5.

2.3 Processing the TDL Standard

It can seen from the introduction provided here that the TDL standard is an extremely large *document*, it is well-structured insofar as each major functional component is described in a dedicated section, and there is quite significant use of tables to try to render data clearly. However, we concur with Zeigler's summary of the general nature of the shortcomings of the TDL standard [28]. As a consequence, we have been selective in our choice of technique when attempting to model and then capture data from the standard, focussing on tabular data in preference to natural language. The TDL standards do not describe an explicit model, hence this must be inferred by analysis and validated through discussion with TDL domain experts. However, there is scope to make progress with a model-based approach by tackling the sheer volume of semi-structured data, such as the Data Dictionary, Message Catalogue, Transmit/Receive Tables, MIN IMP rules, before attempting to tackle the semantics of natural language specifications.

2.4 Network Design

The Link 16 TDL is a nodeless message-based network utilising a Time-Division Multiple Access (TDMA) architecture, hence each platform must know when to transmit and when to receive. There is considerable variability in both message sets and update rates required by platforms, and the network design must accommodate such diversity. The functionally-oriented messages are clustered into a number of Network Participation Groups (NPGs), the network design comprises a number of concurrent TDMA architectures known as stacked nets (see Fig.6) onto which the NPGs are mapped. Hence, a platform must also know which NPG and net to use; platforms will jump from one net to another at run time as they transmit and receive messages in specific NPGs.



Message Catalogue: Message J0, Word I (J0.I)

DFI	DUI	Name	Field	Size	Description
441	032	Timeslot Number, Voice	Bits X..Y	Z bits	…
…					

Fig. 4 TDL Message Catalogue

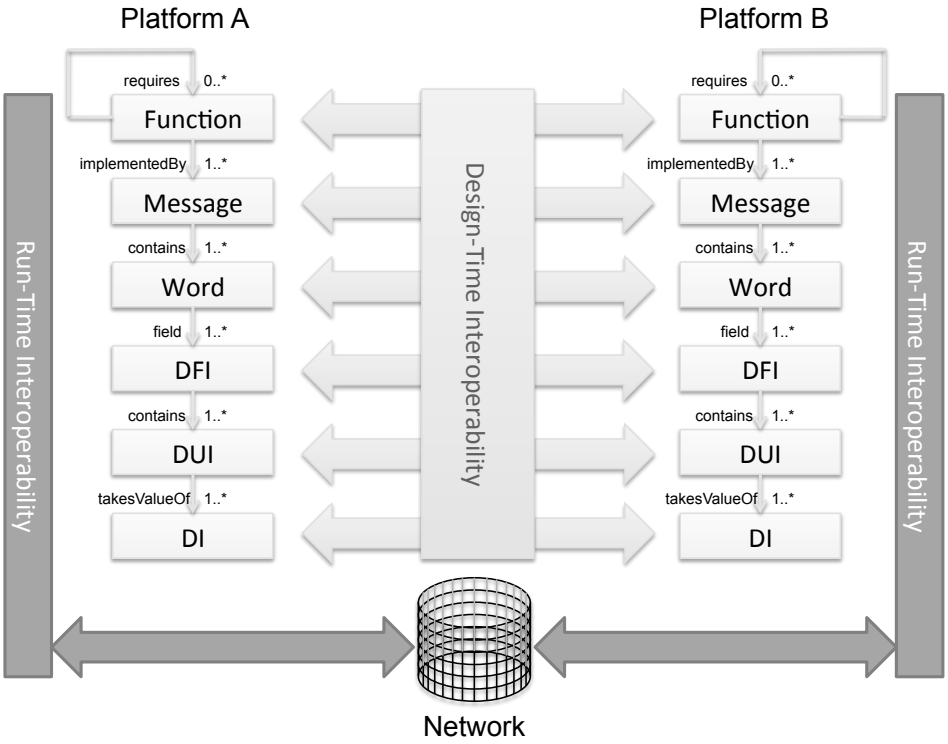


Fig. 5 Basic Hierarchy

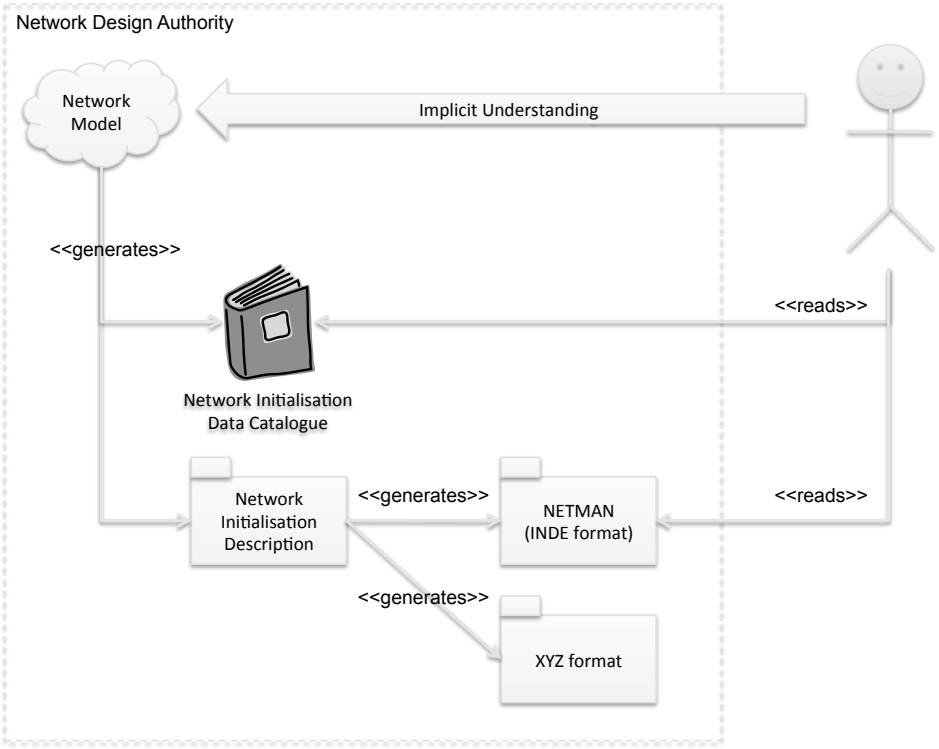


Fig. 7 TDL Network Design and Publication

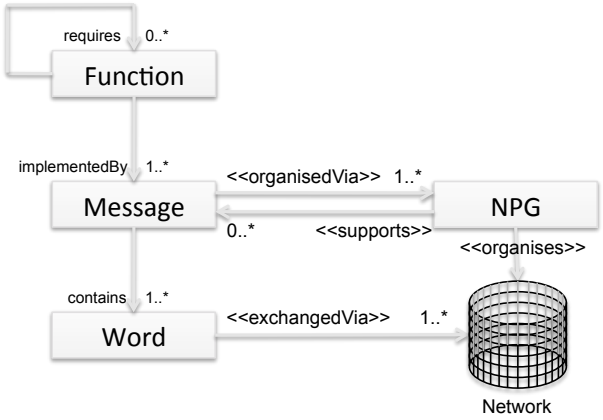


Fig. 6 Functionally Oriented Message Structure

The template for this information is provided via the TDL standard, e.g. [1]. The TDL network design activity for the UK is performed by the Joint Data Links Management Organisation (JDLMO) and published in the form of a human-readable Network Initialisation Data Catalogue (released in PDF); the network design catalogue may be in excess of several hundred pages [5]. In addition to the Network Initialisation Data Catalogue a number of machine-readable formats are also published appropriate to the platform implementation technology; of relevance to this activity is the NETMAN format [6] required for use by a particular air platform to program the TDL terminal with the network design (see Fig.7). This is referred to as the INDE file. Hence, the INDE file provides an instantiation of the network design, and the network design conforms with the description of the network structure provided by the TDL standard [1], e.g. allocating messages to the relevant NPG, using the defined number of timeslots per epoch, etc. One can view the network design as providing the run-time platform interoperability supporting they design-time interoperability required by the IERs and realised by specific platform implementations as illustrated in Fig.5.

The INDE file is a terse, ASCII file describing (amongst other aspects) the timeslots allocated to each platform for transmission and reception. The network design comprises a number of *communities*, each containing a set of *participants* - a *participant* is an object such as a type of aircraft that may communicate over the link. Each participant is allocated a number of *initialisation data sets*, each containing a set of *timeslot blocks*. Each timeslot block belongs to one of a number of sets and identifies the NPG, timeslot and, hence, the bandwidth allocated by the network design. In order for one platform to exchange information with another, the receiving platform must be programmed to receive within the timeslot in which the transmitter is transmitting, the receiver must be receiving within the NPG containing the transmitted information, and the receiver must be receiving on the same net number as the transmitter[2]. Whilst this provides a flexible and efficient (in terms of

[2] in some circumstances a platform may be able to receive on multiple NPGs concurrently

bandwidth) approach it adds complexity to both network design and analysis as platforms jump between NPGs and nets at run-time.

Timeslots are arranged in the form of a carousel comprising a fixed number of slots (one carousel for each of the sets of timeslots), different participants may have differing numbers of timeslots allocated however these are distributed in accordance with specific rules. The bandwidth (number of timeslots) allocated to a specific participant is identified via the Recurrence Rate Number (RRN). Timeslot allocations are presented graphically in the form of a timeslot map, see Fig.8. The timeslot map is a form of Domain Specific Language (DSL) for the TDL network design, note that the ordering of timeslots in the timeslot map has the effect of aligning related timeslot blocks into sequential blocks.

The NETMAN format INDE file suffers from a number of drawbacks that serve to militate against its accessibility to the TDL engineer:

1. It conforms to a terse text-based format (see Fig.9).
2. It contains implicit information:
   (a) Timeslots are identified explicitly as either transmit or receive, however implicit rules allow platforms to receive via the default net and also via a number of special case NPGs;
   (b) Participants may be collected into groups of arbitrary size; however the INDE file does not identify the notion of groups and also does not identify individual members of each group explicitly;
3. It requires skill to read and understand;
4. It cannot be fully understood without recourse to the Network Initialisation Data Catalogue, which may be a sizeable document comprising several hundred pages.

2.5 Motivation

Discussions with the TDL engineers of the customer project identified that they are required to analyse the network design information to ascertain interoperability between platforms at the timeslot level on a fairly regular basis. Such analyses are required to confirm the fitness for purpose of the network design for specific operations, such as platform data link testing. Unexpected interoperability issues can result in wasted test flying hours, at considerable cost. Initial discussions identified two candidate queries that were performed manually to identify timeslot interoperability issues between any two platforms A and B:

1. Identify all timeslot blocks transmitted by Platform A that **may be received** by Platform B.
2. Identify all timeslot blocks transmitted by Platform A that **may not be received** by Platform B.

Fig. 9 Network Initialisation File

Whilst the network initialisation file identifies all timeslot blocks transmitted and received explicitly by each platform there are additional implicit rules of which the TDL engineer must be aware, such as the receiver can *see* all timeslots transmitted in its default net.

It has been demonstrated above that the network design information used is somewhat opaque, and there is anecdotal evidence to suggest that a significant learning curve is faced by engineers performing timeslot interoperability investigations. Hence, tool support to execute commonly encountered queries over the network initialisation data set will both reduce the time taken and improve the accuracy when performing typical timeslot-based interoperability assessments, and this serves to reduce the likelihood of wasting valuable flight tests. Raising the level of abstraction of the source material as may be achieved by migrating to a model-based view will make the domain more accessible to engineers and reduce the learning curve; an additional benefit perceived is the possibility of integrating the network initialisation model with existing TDL models developed by the ATC over the past few years.

3 Metamodelling Technology

In this section we provide a brief introduction to the model-driven technologies used in the context of this work: EMF for (meta-)modelling and Epsilon for automated model management. In principle, other model management languages such as OCL, ATL or QVT could have been used towards the same end.

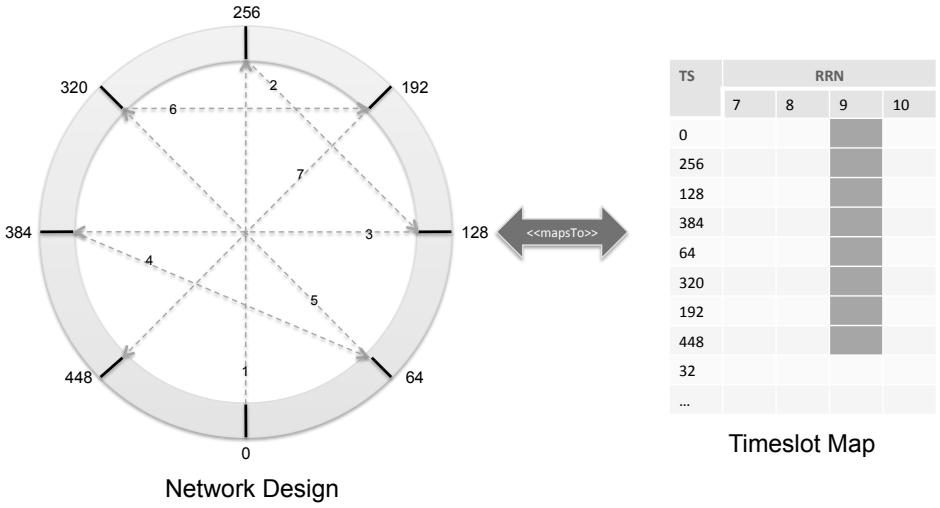


Fig. 8 Timeslot Map

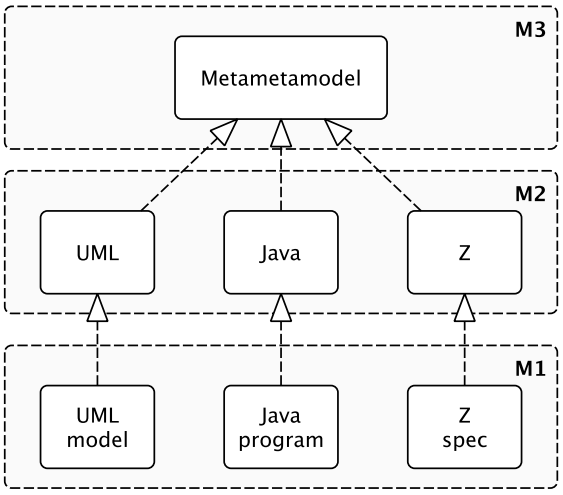


Fig. 10 A Typical Metamodeling Architecture

3.1 The Eclipse Modeling Framework

A metamodelling architecture enables engineers to define the abstract syntax of modelling languages and models that conform to these languages. To achieve this, a metamodelling framework typically provides a three-layered (M1-M3) hierarchical architecture as seen in Fig.10. M3 contains the core metamodelling language which is used to define modelling languages, M2 contains modelling languages defined using the metamodelling language of M3, and M1 contains models conforming to languages contained in the M2.

The Eclipse Modelling Framework (EMF) [20] is a modelling framework built atop Eclipse in an effort to provide a practical approach to modelling and model management for Java developers. EMF provides an integrated graphical editor for specifying metamodels, using its Ecore M3 language, and a framework (EMF.Edit) for generating graphical editors for new modelling languages from their Ecore metamodel. Moreover, EMF provides tools for extracting models from Java annotated source files and XML Schema documents. While EMF started as an independent modelling framework, in its latest versions it has been aligned with the MOF 2.0 featuring model validation with OCL and storage capabilities using XMI.

3.2 Epsilon

Epsilon[3] is a mature family of interoperable languages for model management. Languages in Epsilon can be used to manage models of diverse metamodels and technologies (detailed below). The core of Epsilon is the Epsilon Object Language (EOL) [21], an OCL-based imperative language that provides additional features including model modification, multiple model access, conventional programming constructs (variables, loops, branches etc.), user interaction, profiling, and support for transactions. EOL can and has been used as a general-purpose model management language (e.g. for operational model transformation). It is primarily intended to be reused in task-specific model management languages. A number of task-specific languages have been implemented atop EOL, including: model transformation (ETL), model comparison (ECL), model merging (EML), model validation (EVL), model refactoring (EWL) and model-to-text transformation (EGL), see Fig.11. These languages reuse EOL in different ways, e.g. by acting as a preprocessor, or by using EOL to define behaviour of rules.

Epsilon is designed to be technology agnostic - that is, the same Epsilon program can be used to manage models from different technologies: the concepts and tasks of model management are independent of how

[3] `http://www.eclipse.org/epsilon`

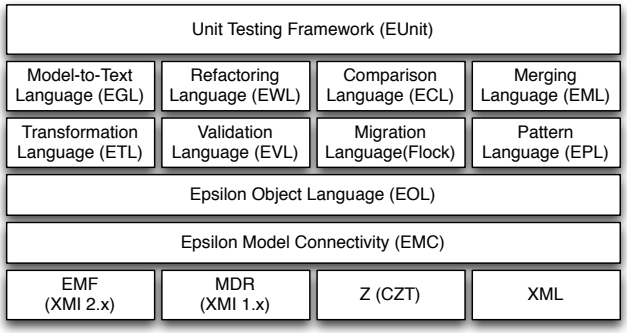


Fig. 11 Overview of the architecture of Epsilon

models are represented and stored. To support this, Epsilon provides the Epsilon Model Connectivity (EMC) layer, which offers a uniform interface for interacting with models of different modelling technologies. New technologies are supported by adding a *driver* to EMC. Currently, EMC drivers have been implemented to support EMF (XMI 2.x), MDR (XMI 1.x), pure XML, and Z specifications in LaTeX using CZT. Also, to enable users to compose complex workflows that involve a number of individual model management tasks, Epsilon provides ANT[4] tasks and an inter-task communication framework discussed in detail in [22].

4 Technical Approach

Source material is provided by an external agency in the form of documents, some items of which are intended for consumption by software systems. This material may be seen as instantiating a specific instance of certain elements of the informal model defined by the TDL standards [1,2]. A DSL (the timeslot map) is associated with this particular aspect of the TDL. Analysis of the source data informs the design of an overarching metamodel (the domain model), whilst the DSL provides an example of the format required by domain experts for presentation of reports generated from queries executed over instances of the metamodel (the report model). As source material is provided in multiple documents parsers must be provided to allow the population of models from source material in its native format. Data contained in the populated models needs to be both consistent and reconciled into a single, coherent model. Hence, there is a need for model transformations and, potentially, model merging. There is also a requirement for validation of the populated models resulting from this process against well-formedness rules. Finally, there must be a workflow to orchestrate the various strands of work, resulting in a system that may be used by the domain experts. This is illustrated in Fig.12.

[4] `http://ant.apache.org`

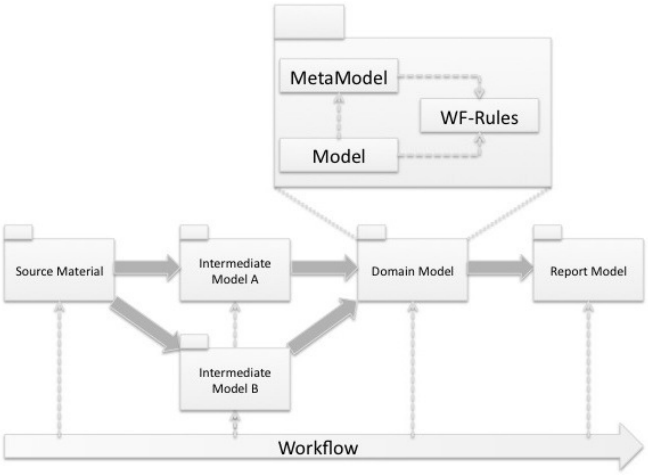


Fig. 12 Technology Independent View

4.1 The Role of Models

In the context of the work described in this paper, models serve the dual purpose of (a) increasing rigour by making explicit the concepts, relationships and well-formedness rules inherent in the network design source information, and (b) raising the level of abstraction of this material such that it is more accessible to the TDL engineers. However, when we talk about *models* in this context we really mean *processable models*; i.e. having described and populated an instance of the Network Initialisation Data Catalogue model we can perform operations on the model instance to achieve some goal, such as:

- Transforming it into another form (vertical or horizontal transformation).
- Establishing its validity against an explicitly modelled set of well-formedness rules.
- Query over the model structure to provide some *result*.
- Render the *result* in a domain-specific format.

The problem was broken down into a number of smaller steps.

1. Identification and prototyping of a model of the TDL network design domain, based on an analysis of the source information, such that we could confirm user requirements and provide confidence to both ourselves and the users that a model-based approach would provide the capability required in order to achieve the positive Return on Investment goal.
2. Selection of appropriate metamodelling and model management technologies.
3. Transformation of source material (which was provided in different formats) into the defined model-based structures.
4. Definition of well-formedness rules to confirm that the model has been instantiated correctly from source material against the metamodel.

5. Assembly of a workflow that automates steps 3 and 4 for extracting and validating models from source material.
6. Prototyping of the queries that had been described to us by the TDL domain experts over a well-formed model; at this point we could compare the output of our model-based approach to expected results.
7. Implementation of a prototype a tool that would be usable by the TDL community.

Each of the steps is described in the following sections.

4.2 Defining the Model

As stated above and illustrated in Fig.7, the network design is provided by an external agency (JDLMO) in a variety of formats, of which the NETMAN format is used by our customer. The general rules governing the network design are provided by the underlying TDL standard, e.g. [1]. This describes core concepts, such as the Time Division Multiple Access (TDMA) architecture, Network Participation Group (NPG), and stacked net structure. Therefore, the network design can be seen to exist within the context of the TDL standard.

We were provided with an example network design and the NETMAN Interface Control Document (ICD) [6] as source material. The NETMAN ICD provided both the grammar for the textual concrete syntax in BNF and also defined the ranges of legal values expected for all fields. We used the major terms in the BNF grammar to inform the structure of the underlying abstract syntax model whilst the specified ranges of legal values provided the initial constraints over the model, although a model-based approach allows for more profound constraints than simply checking values of individual attributes. However, the resulting model was not simply an object model of the BNF grammar as it was also necessary to accommodate information identified from the standard. The initial model development was undertaken guided by the source material mentioned above: TDL standard [1], Network Initialisation Data Catalogue [5], and NETMAN ICD [6]. Fragments of the evolving model were validated by populating with exemplar data via the abstract syntax and via discussion with the TDL domain experts, although it became apparent that there were a number of aspects of the network design with which the domain experts were not familiar. As a general approach we validate the structure of our metamodels through simple examples populated via the abstract syntax, using (e.g.) the Human-Usable Textual Notation (HUTN) [7] before investing effort in the development of parsers and model transformations. Development of an INDE file parser was found to inform the design of the metamodel by suggesting refactoring opportunities. Having completed the prototype of the model and associated parser we were able to populate the model and execute the constraint checks derived from the NETMAN ICD. An outline of the process followed is illustrated in Fig.13. Discussions with project TDL engineers regarding the two queries the prototype was to provide over the populated model revealed that the model was incomplete; thus, although the model captured the semantics of the INDE file faithfully it had exposed the need for additional concepts relating to the structure of the network: Communities, Participants, and Groups of Participants (see Fig.14). Whilst the INDE file contained sufficient information to allow a TDL terminal to be programmed with transmit and receive network information, further information was required to establish interoperability between multiple platforms within a given network design. The source for the missing information was the Network Initialisation Data Catalogue [5]. The necessary information was provided in a fairly compact tabular form within two subsections of the document, and we were able to extend the metamodel design and demonstrate a complete example using the process illustrated in Fig.13. Demonstration of the proposed tool via a simple command-line interface and textual output confirmed the validity of the metamodel and prototyped queries, however it also revealed an additional customer requirement for a third query over the model.

TDL timeslot information is rendered via a domain-specific format known as the timeslot map, hence the users added a further requirement that the results of the queries to be executed over the model should be presented in the form of a timeslot map. At this point we had a clear understanding of the customer requirements and had the task of maturing the prototype to a point that it could be deployed to the user community.

4.3 Selecting Supporting Technologies

We chose to describe our metamodels using the Emfatic concrete (textual) syntax [10] for Ecore as we found it to be quite compact and easily accessible to the engineers involved. Metamodels expressed using Emfatic may be used to generate Ecore metamodels (the reverse generation is also supported), and these may be rendered graphically in the form of a class diagram. Epsilon, which was discussed in Section 3, was selected to support the model management activities we required (object language, validation and transformation). The motivation for selecting the Epsilon platform was based primarily upon the availability of technical expertise – other languages that provide similar capabilities (e.g. OCL for querying and validation and QVT for transformation) could have been used instead.

At the time of the migration Epsilon did not natively support parsing of schema-less XML files. Hence, it was necessary to convert the INDE file into XML and underpin this with an XSD. We undertook this task using

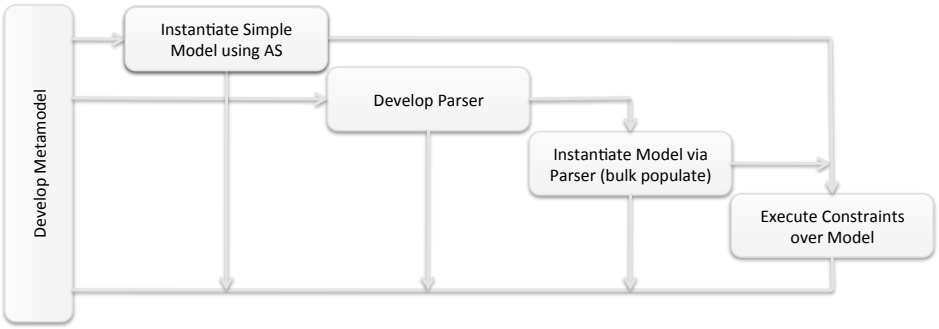


Fig. 13 Outline Development Process

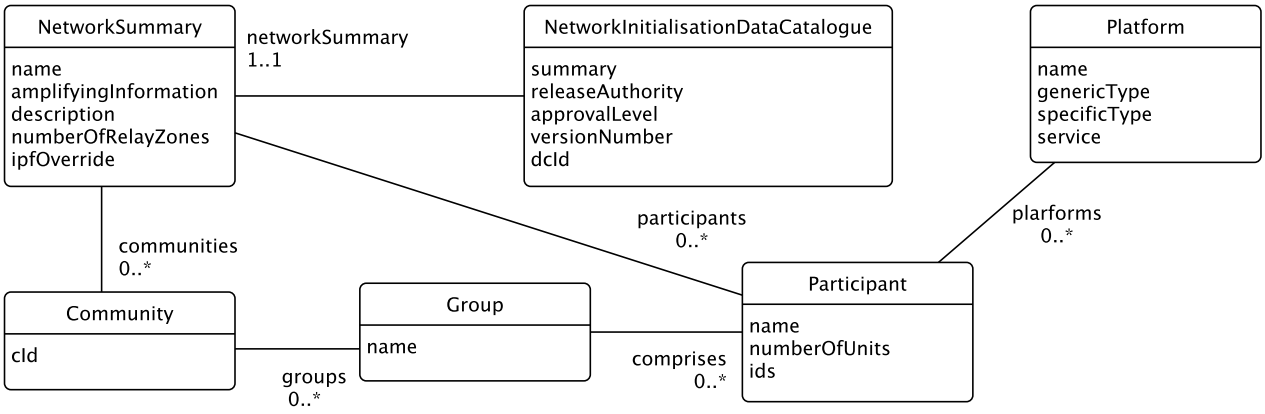


Fig. 14 TDL Network Structure

ANTLR [9] which was found to be a relatively simple task with the added benefit of allowing us to check the source document against the schema before attempting to instantiate our metamodels.

4.4 Transforming the Models

An Ecore metamodel was auto-generated from the XSD underpinning the XML version of the INDE file, however the structure of the metamodel generated from the INDE (XML) file differed somewhat from the target domain metamodel, e.g. it lacked the concepts relating to the structure of the network mentioned above, and it did not make use of class hierarchies. Hence, we were left with the task of transforming the source model into the domain model.

We achieved this using the Epsilon Transformation Language (ETL). As the file (a large PDF document) provided the additional network structure concepts in a compact and simple tabular form we chose to parse this information directly into XML via a simple bespoke parser written in Java; once again this XML document was underpinned with an XSD and an Ecore metamodel was auto-generated from the schema. The relationship between metamodels and transformations is illustrated in Fig.15. It should be noted that Epsilon has subsequently evolved to allow parsing of schema-less XML documents directly, hence the transformations shown as IndeFile2NetInit and NetDsgn2NetInit could be eliminated, although at the cost of a potentially complex parser - i.e. the complexity of the transformation must go somewhere, either in the parser or in an explicit model transformation.

The model transformations IndeFile2NetInit and NetDesgn2NetInit each handle a separate aspect of the target metamodel, the result of running both transformations is a completely instantiated metamodel of the Network Initialisation domain model. Once instantiated the domain model may be checked for well-formedness (via rules expressed in EVL), and queried for some result (via operations expressed in EOL).

4.5 Validating the Models

The introduction of the intermediate translation of source material into XML underpinned by XML schema was primarily a requirement driven by the adoption of Epsilon as the most pragmatic route to populating models from source data. It also provided a point at which the source data could be checked by a validating XML parser, providing confidence of the validity of the source data prior to transformation into an instance

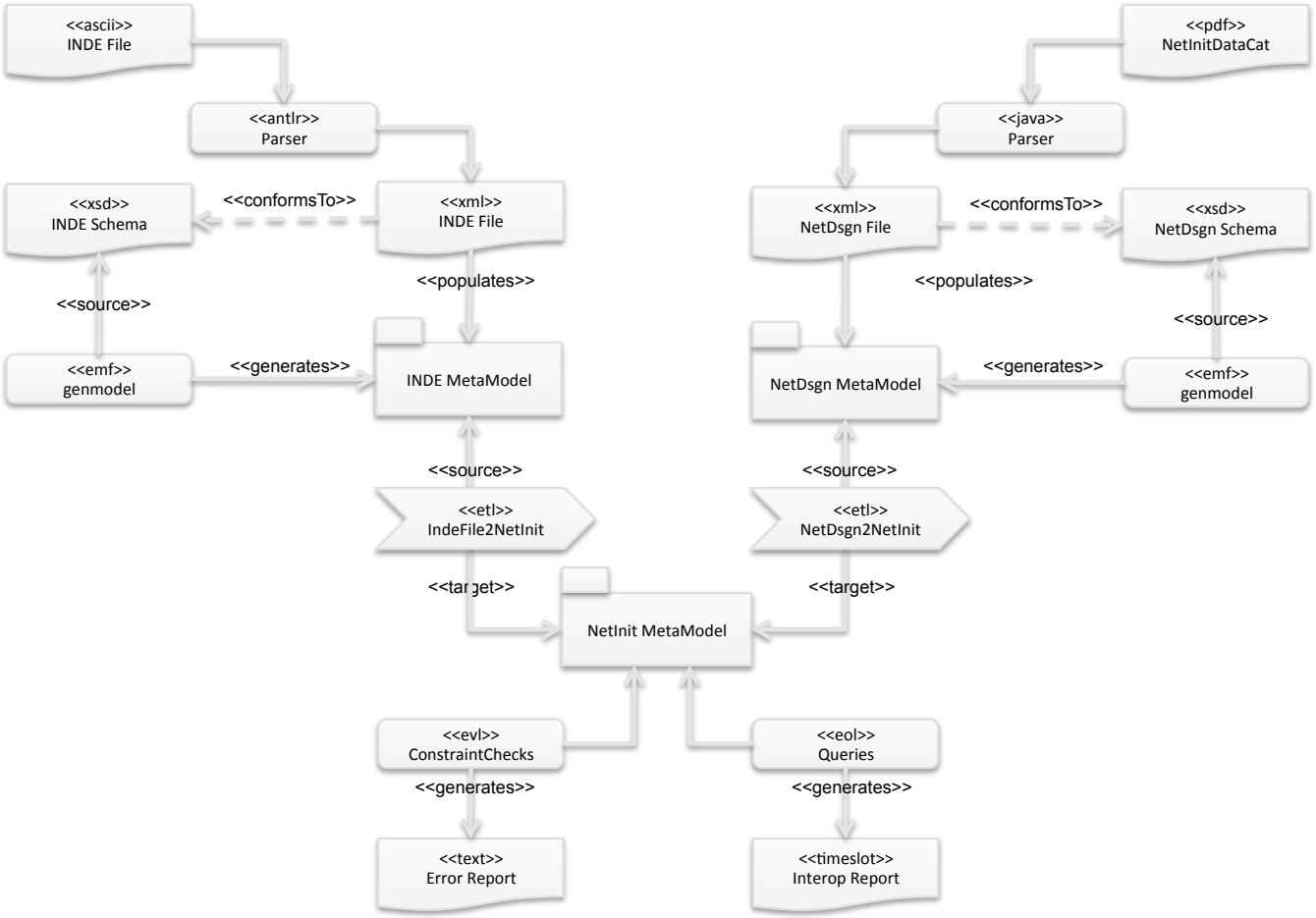


Fig. 15 Metamodels and Transformations

of the domain metamodel. The next step was to validate instances of the domain metamodel against a set of domain-specific constraints/invariants.

This was achieved using Epsilon's OCL-based constraint language, EVL, with constraints being drawn primarily from the NETMAN ICD [6]. An example constraint to confirm that the *blockId* slot value of an *IndexedBlock* object is within the range 1..64 is illustrated below.

```
1  import "../Operations/IndexedBlock.eol";
2  context IndexedBlock {
3
4    constraint BlockIdInRange {
5      check {
6        var lower : Integer := 1;
7        var upper : Integer := 64;
8        var id  : Integer := self.blockId;
9        return id >= lower and id <= upper;
10     }
11     message:
12       self.asString() +
13       ": must have a Block Id in the range " +
14       lower + ".." + upper +
15       " but has " + id + "."
16   }
17 }
```

Whilst the initial set of constraints over the model are relatively simple, discussion with the users and increasing experience in the domain leads us to believe that there is scope for more profound constraints over the model, such as reconciling the Recurrence Rate Number (RRN) against the actual number of timeslots allocated, confirming that all timeslot sets are distributed correctly within the carousel, etc. Furthermore, complex processing may be deferred to operations written in EOL and invoked via EVL. The example above provides a simple demonstration of the approach via the *import* statement (line 1) which provides the constraint with access to the *asString()* operation on the class *IndexedBlock* (line 12). In our opinion, the use of EVL offers some advantages over the XML schema; firstly constraints expressed in EVL are somewhat more accessible to the non-specialist and, secondly, implementation of more profound constraints over the model would be somewhat more challenging to represent in XSD.

Each of the constraints was captured in EVL. The idiom adopted was to create an EVL file for each class for which one or more constraints were required, an overarching EVL file importing all class-level constraints was then created - note that this approach is not mandated by Epsilon, however it was found to be beneficial when

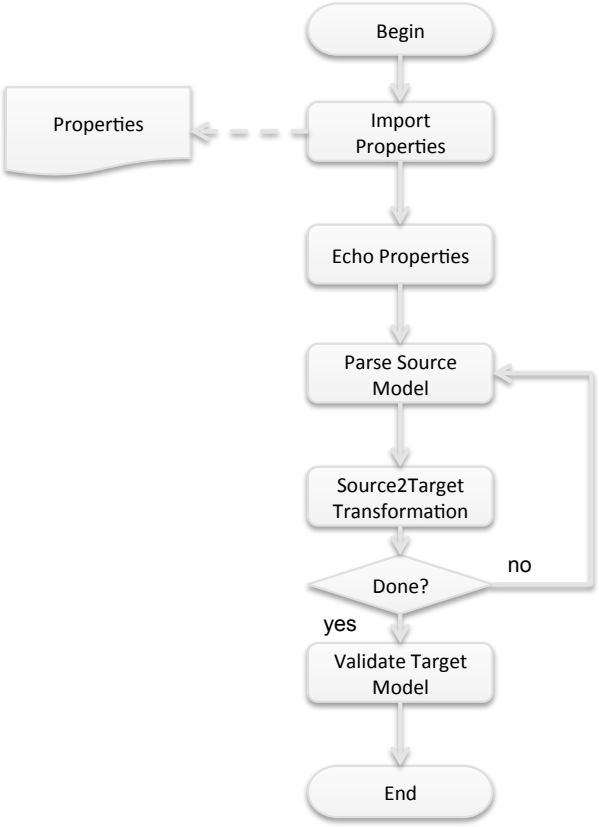


Fig. 16 Generic Structure of an Ant Workflow

integrating constraint checking within Ant workflows as discussed below.

4.6 Prototyping the Workflow

The next step was to compose the model parsing, transformation and validation steps into an automated workflow that would enable us to re-populate and validate the model quickly in response to the incremental enhancements.

We chose to prototype this using the Java-based Ant [11] framework. Ant workflows are expressed using an XML-based syntax and provide support for modularity of the workflow, whilst Epsilon provides a number of extensions to Ant to support model management operations [12]. The general structure adopted by this and other Epsilon-based projects is to define one or more property files to identify source and target components (i.e. metamodels, models, transformations, constraints) and a potentially nested set of Ant files; this allows the rapid reconfiguration and extension of the workflow as development of model structures progresses. The general structure of an Ant workflow is illustrated in Fig.16. Following completion of the prototyping we were able to confirm correct behaviour of the system with the domain experts. This paved the way for development of a fully-fledged Java-based application using the model-based approach with the Ant-based workflow being migrated to native Java code.

4.7 Querying the Models

Initial discussions with the TDL engineers identified two queries to be executed over the Network Initialisation domain model to help establish the timeslot-level interoperability between two platforms in the network design. The queries required were:

- Identify all timeslots transmitted by platform A that **may** be received by platform B
- Identify all timeslots transmitted by platform A that **cannot** be received by platform B

We chose to express these queries using Epsilon's OCL-based expression language (EOL). The idiom adopted to express queries was to create an EOL file for each class of the metamodel – note that Epsilon does not mandate this approach. Operations in EOL may have side-effects and have a simple syntax similar to Java. Each of the above queries had already been prototyped using XMF (a metamodelling tool associated with a previous metamodelling project) and validated by the TDL domain experts, it was found to be a relatively simple matter to re-write the operations in EOL (XOCL and EOL share many similarities although they are not formally grounded in the same underpinning metamodel). The result of each query is a set of TimeslotBlock objects. A TimeslotBlock contains many attributes, including the RRN, timeslot set, index slot number, net number, and Network Participation Group (NPG); these attributes provide the information required to render the result of the query in the form of the Timeslot Map. The textual representation of a timeslot block is:

- Set + '-' + TimeslotNumber + '-' + RRN + '[' + NPG + ']'

e.g. a timeslot block in Set A with an index slot number of 127, a recurrence rate number of 6, transmitted in network participation group 7 would appear as:

- A-00127-6[7]

An example of one of the queries requested by the domain experts is presented below. In this example we wish to identify all timeslot blocks transmitted by one participant that are receivable by another (possibly even the same) participant:

```
1 timeslotBlocks := self.getAllTimeslotsTransmittableTo(
      other);
```

The body of the *getAllTimeslotsTransmittableTo()* query is described below. The decomposition of the operation illustrates that calculation of the result is based upon the union of a number of rules combining: (i) timeslots explicitly transmitted by participant A to participant B, (ii) special cases, and (iii) default cases capturing the notion of (e.g.) implicit data exchange over a default net number.

```
1 getAllTimeslotsTransmittableTo(other) : Set(
      TimeslotBlock)
2  explicitlyTransmitted := self.
       GetTimeslotsTransmittableTo(other);
3  specialCaseTransmits := self.
       getSpecialCaseTimeslotsTransmittableTo(other);
4  defaultTransmits := self.
       getTimeslotsTransmittableByDefaultTo(other);
5  allTransmittable := explicitlyTransmitted +
       specialCaseTransmits + defaultTransmits;
6  return allTransmittable.sortBy(b | b.blockId);
```

At this point we were able to demonstrate to the domain experts a model-based capability linked directly to the formal source material provided by the network design authority. Although the results generated were text-based, domain experts were easily able to verify our results and the project then turned its focus to deployment of a tool that would be able to provide an intuitive user interface and render results using the domain-specific language of the timeslot map.

4.8 Rendering Results

As stated in the sections above, timeslot allocations are generally rendered using the domain-specific notation of the *timeslot map*. The timeslot map provides an information rich, graphical rendering of timeslots, timeslot numbers, NPGs, and RRNs; it also accommodates the even distribution of multiple timeslots across the carousel such that related timeslots appear as a contiguous group (see Fig.8). Source information provided by the network design authority is summarised in the form of a number of timeslot maps; one for each of the timeslot Sets. Hence use of the timeslot map as a vehicle by which the results of the model-based network analyses were to be presented was a natural choice for the domain experts. It was found to be possible to capture and populate the full structure of the timeslot map as intended by its original designers, giving confidence that we could migrate from XSLT to a contemporary template-based model-to-text transformation language.

The language of choice for this step was EGL [12], Epsilon's model-to-text transformation language, which we have used successfully on another model-based project as illustrated in Fig.17. Example results were presented to the domain experts, rendered in the form of HTML timeslot maps. Although our example timeslot maps were semantically and syntactically correct, it transpired that the network design authority publishes timeslot data using a more compact but syntactically incorrect rendering. We were requested to adopt an approach similar to network design authority, e.g. each timeslot map split into two parts, with the exception that only those RRNs presented in the vertical plane should be used. This requirement was instrumental in causing us to implement the timeslot map rendering using the Java Standard Widget Toolkit (SWT), see Fig.18.

5 Discussion

In this section we reflect on the approach adopted and decisions made in order to illustrate the lessons learned from this project. The discussion focuses on the method and technologies used.

5.1 Development Method

The body of work reported above comprised one strand of a larger programme of work funded by BAE Systems. An issue that emerged early on in the process was that whilst there were a number of TDL-related projects running concurrently, each was at a different stage in its life-cycle and saw the issues associated with the development programme differently. At the instigation of our research in 2005 we interviewed staff from six projects to capture their major areas of difficulty, our expectation was that there would be one or two hot spots that would become the focus of our work, however this wasn't found to be the case. It appeared that no two projects were the same; this was partly a consequence of where they were in the life-cycle and also a consequence of the project itself: one project might be tasked with integrating a TDL terminal, whilst another may be using a TDL terminal integrated by a third party. Clearly, this results in a differing view of both issues and priorities [15]. This particular project was selected for the following reasons:

- it was largely within the scope of one programme,
- relevant source material was available, or could be made available relatively easily,
- a domain expert was available (although access was limited),
- there was potential for future exploitation on other projects,
- we believed the problem to be amendable to a model-based approach,
- the project predicted that the work would realise an attractive ROI.

Over the period of development of the tool, the interaction with the end-user domain experts was characterised by a series of meetings, each addressing further issues, in parallel with elaboration and corrections of the metamodels and prototype tool (which are one and the same). Such dialogue focused around the vision of the tool that could automate the previously manually intensive activities, and so provide a positive RoI, once the tool achieved an appropriate level of maturity. This was found to be a valuable activity that served to elicit requirements leveraging from development that, in the authors' opinion, are unlikely to have been articulated without the stimulus provided by the engineers seeing 'something' working in, what at first may have appeared to be, an unrelated domain. Examples are:

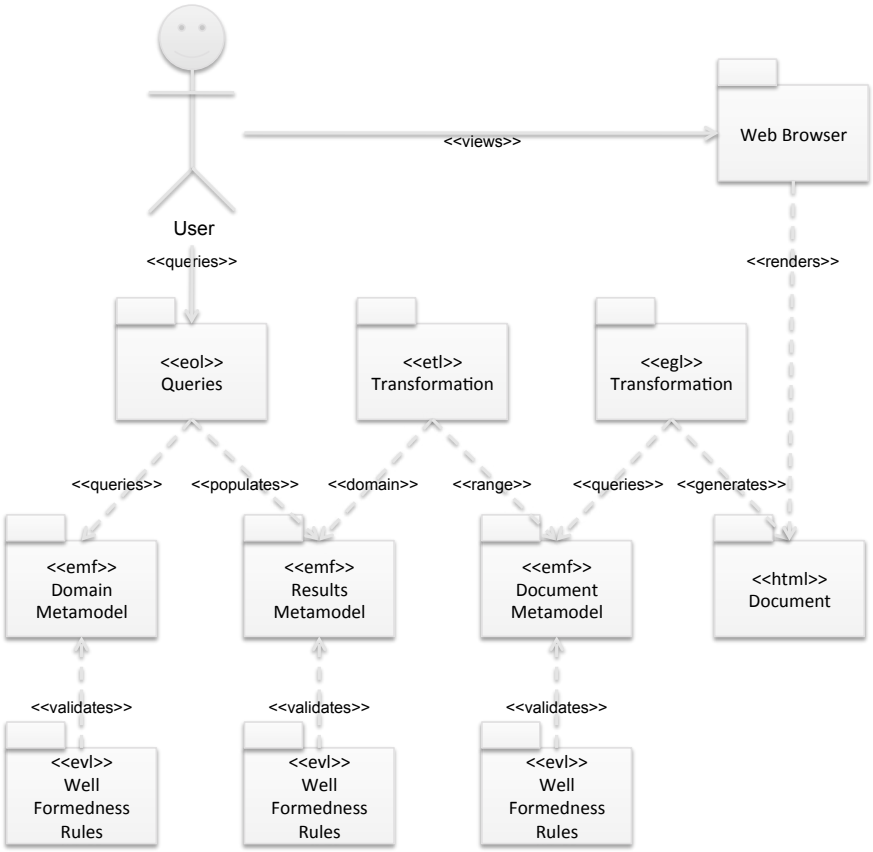


Fig. 17 Example Family of Metamodels

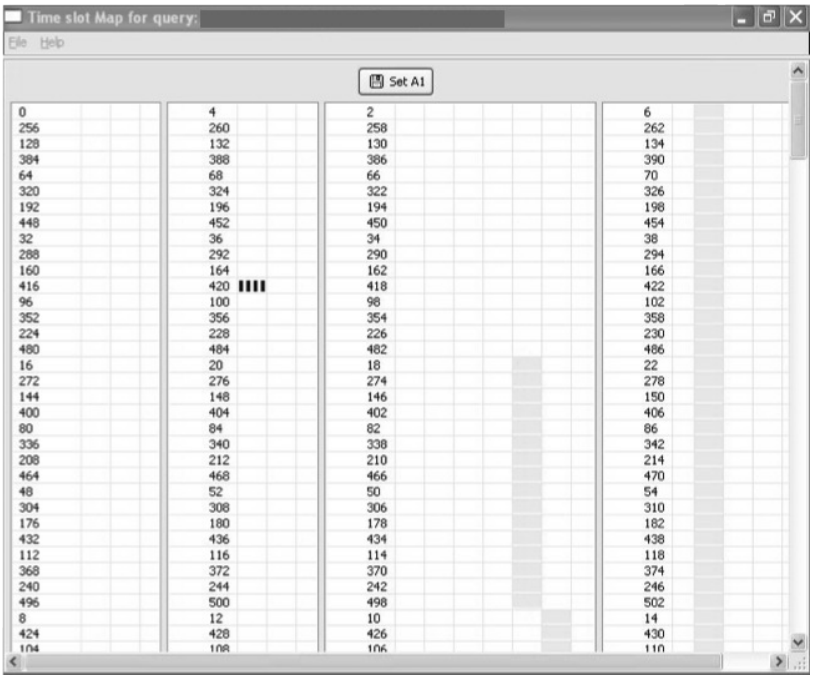


Fig. 18 Generated Timeslot Map

– extension of the timeslot interoperability analysis tool to provide hardware support,
– extension of models of the TDL message structure to reconstitute captured binary TDL traffic into messages, DFI/DUIs, etc.,
– integration of models with a commercial requirements management tool.

The adoption of a model-based architecture provided a powerful vehicle for both requirements capture and development. The model-based approach allowed the developers to describe their understanding of the problem domain in a domain-specific but largely technology-independent way, based on the (static) graphical view provided by models structured in terms of the easily accessible concepts of packages and classes. More detailed discussions could be supported by the executable nature of the models. Domain experts could view the output generated by the models which could then be used to drive discussion of the semantics of classes via the associated operations. Model-based tooling facilitated such discussions by providing a tightly integrated solution that was structured clearly and easy to navigate. Furthermore, this allowed us to present an interconnected view of large-grained functionality, indicating existing capability and showing how future capability could leverage from this to meet specific requirements, e.g. integration of static and dynamic views of the TDL, extension via new models, execution of queries, and rendering of results.

In most cases working closely with the TDL engineers for short periods of time was possible. This strategy allowed us to understand the nature of their requirements in detail and allowed us to demonstrate early prototypes of models and tools developed. In the case of the Timeslot Interoperability tool we first demonstrated a simple command-line driven prototype generating text-based results, this enabled us to refine and validate the model and also provided the TDL engineers with confidence that we had understood the requirement and developed a sound basis for a solution. A result of this demonstration was the emergence of a requirement to render the textual results in a graphical form; the domain-specific notation used to display timeslot information is the Timeslot Map. Example renderings of results using XSLT to generate HTML were developed using example data. Development of the network initialisation analysis domain and the rendering of results progressed in parallel, the intention being that the two would ultimately be connected via a horizontal model transformation (e.g. see 20). A small number of subsequent meetings were required to clarify the requirements for the formatting of the graphical output to be generated (the timeslot map); it was at this point that the requirement for an additional query was identified by the TDL engineers. The model-based approach allowed this requirement to be accommodated with ease as no changes were required to the model(s), transformations or constraints.

Towards the end of the development phase there was a change to the network design published by the JDLMO. Somewhat to our surprise, the Network Initialisation Data Catalogue for the new network design failed to parse successfully (see Fig.15). Investigation revealed that the network design authority does not make use of standard platform names across network designs, e.g. one network design may refer to (say) a Harrier as 'HAR', whilst another design may refer to the same platform as 'GR9'. Unfortunately, our parser did not allow for such variability of token names, hence the parser was extended to capture the list of tokens from the catalogue before parsing the network design section of the catalogue. Once again, however, there were no changes to the underlying model(s). A more significant issue arose some months following delivery of the tool with the identification of an implicit requirement relating to the support of an additional type of network design (referred to as a *closed network*). Closed networks are developed on an ad hoc basis to facilitate local testing of a platform and differ slightly in structure, e.g. there is no explicit definition of a community. It was found to be possible to accommodate the subtle differences of the closed network via a simple relaxation of one of the grammar rules of the ANTLR parser in Fig.15 and an associated constraint in the schema for the generated XML document. Once again, no changes were required to the underlying model(s) because they were successfully shielded by the supporting document parsers.

5.2 Implementation Technology

The deployed timeslot interoperability analysis tool followed the Model-View-Controller pattern, comprising a Java front end, providing the user interface, and a model with a Java interface generated from a number of Eclipse-based Epsilon components as illustrated in Fig.19. Having developed and tested the metamodel it was a simple process to generate the relevant Java interface classes from EMF to allow the Java View-Controller components to bind with the metamodel, this allows the model to be populated via the parsed documents and to be queried based on user-defined criteria; the results of each query being rendered in the form of a timeslot map as illustrated in Fig.18. The strict separation of the Model from the View-Controller had the benefit of keeping the View-Controller clean, avoiding pollution with unnecessary concepts from the model. The workflows developed in Ant for the prototyping and initial testing were manually transcribed into the Java View-Controller components. Automated transformation of the Ant workflows would be a potentially useful feature, hence it is suggested that a workflow metamodel could

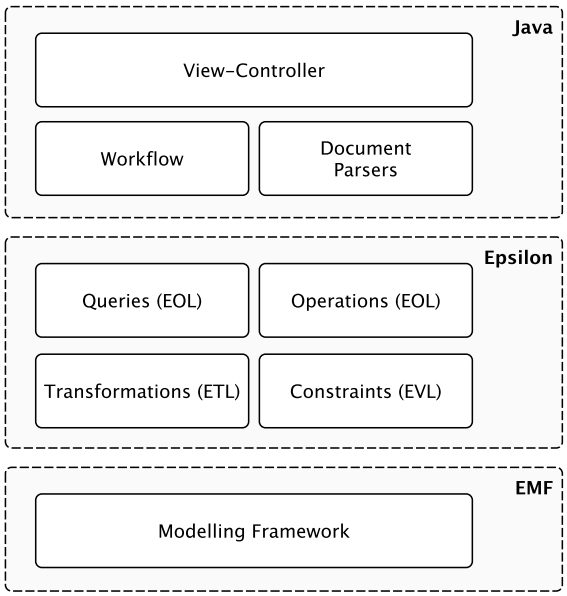


Fig. 19 Tool Architecture

be developed and EGL scripts written to support multi-target generation. A majority of the issues encountered with faulty workflows during prototyping were a result of the rather terse form of Ant and the relatively poor level of error reporting; we would have some reservations regarding the deployment of Ant workflows to the end user's desktop, although Ant was extremely useful during prototyping activities.

We investigated the rendering of the timeslot map via a document-based XSLT approach and demonstrated an HTML rendering based on hand-transcribed source (XML) data. This demonstration provided a strict implementation of the timeslot map and drove out the non-standard use of the timeslot map structure adopted by the TDL community that would be easier to adopt via Java and SWT; this decision was also supported by the practicalities of the skills available at the time. We are confident that we could develop a metamodel of the timeslot map and adapt our XSLT or migrate to EGL such that results of the timeslot interoperability queries could be transformed into timeslot map models and further transformed into HTML for rendering via a web browser following the idiomatic presentation style adopted by the TDL community. We have used such horizontal transformations between domains successfully elsewhere in our TDL modelling work, and output is not restricted to HTML, e.g. we have generated code for Graphviz[5] and Promela for use with the SPIN model checker[6] for other projects. Ultimately, our decision to render the results directly in Java via SWT was driven by three factors:

- lack of a suitably skilled metamodelling engineer to undertake the work within the window of opportunity,
- lack of a compelling business case to reuse the timeslot map metamodel in future work,
- non-standard rendering of timeslot map data complicating the document transformation.

This raises an additional observation: the learning curve required for an experienced software engineer to become productive with the Epsilon/EMF metamodelling technologies versus the availability of software engineers skilled in the ubiquitous, and mature, technology of Java and, in particular, SWT. At the time of development of the timeslot interoperability tool Epsilon was still under active development, whilst there were surprisingly few compatibility issues as development of the toolset progressed, there were some issues regarding the support for debugging and error reporting; as Epsilon has matured so these observations have been addressed. However, our development of Epsilon-based metamodels became, to some extent, a victim of their own success, resulting in a skills shortfall that, in turn, gave rise to business opportunities that could not be exploited fully. It should be noted that such opportunities have not been restricted to the TDL domain and have encompassed a broad range of domains, including, for example, the development of metamodels to support the modelling of business processes. Hence, whilst Epsilon/EMF provides a stable foundation for the development of robust and extensible solutions, one must balance this against the effort required to develop a critical mass of skilled metamodel developers; clearly, a background in Java technologies (particularly under Eclipse) would be a major advantage, however we believe that metamodelling requires a different mind-set to that required for more traditional technologies. The major advantages we have observed in the migration to a *model-based approach* in this and other related projects are:

1. abstraction
2. validation
3. modularity and reuse

5.2.1 Raising the Level of Abstraction Although the project described in this paper comprised a number of complex, low-level aspects it was relatively easy to identify the major domains of relevance and begin articulating our understanding via a textual and graphical modelling language. We found the Emfatic textual modelling language [10] to be easily accessible to programmers and this allowed the rapid creation of metamodels, while the graphical rendering of such models allowed the TDL domain experts to confirm our understanding of the problem domain. This allowed us to progress quickly to example models, partially populated via hand-transcribed data, e.g. as is supported by HUTN [7].

[5] http://www.graphviz.org
[6] http://spinroot.com/spin/whatispin.html

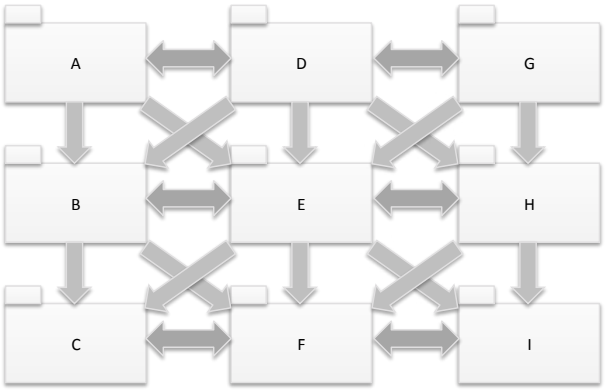


Fig. 20 MDA View

5.2.2 Validation Following feedback from the TDL domain experts we were then sufficiently confident in our manually-populated models to develop the necessary parsers to transform actual source data files into a ubiquitous machine processable format (XML). Use of XML also provided the benefit of allowing us to confirm the validity of documents generated via a relevant XML schema, this schema also provided a vehicle by which the document could be parsed into EMF with ease. The ability to confirm the validity of output at each stage of the process was believed to be useful in order to prevent errors propagating that would, potentially, be difficult to identify as a result of the volume of source data and its very terse format. The definition of constraints over the metamodels further allowed us to identify errors introduced via the model transformations.

5.2.3 Modularity and Reuse We have found that the model-based approach when applied in a disciplined manner tends to result in a number of relatively small metamodels (less than 40 classes), indeed most of our metamodels are somewhat smaller than this. Clearly, the advantage of building small, domain-specific models is that the model is easy to understand in isolation, however the complexity comes when one wishes to reuse one metamodel with another, or transform an instance of one into an instance of another. This underlines both the flexibility and opportunities for model reuse but also emphasises the importance of developing constraints over metamodels to catch errors at the point of introduction. Modularity and reuse are themes within Model Driven Architecture (MDA) that one sees as being associated strongly with the evolutionary approach [14]. However, whilst the typical view of MDA appears to comprise a vertically integrated collection of models organised in terms of the Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM), we see a more general and loosely coupled suite of metamodels as illustrated in Fig.20.

The Epsilon toolkit provides a collection of languages (based on a common core) to operate on metamodels in the EMF. The user has considerable freedom of use of the languages with regards to structure. At one extreme one may package all operations on all classes in the metamodel in a single EOL file, at the other extreme one may package individual operations in their own EOL file. Following experience with more conventional modelling tools and programming languages we created a separate EOL file for each class for which one or more operations were required. Similarly, we created an EVL file for each class for which one or more constraints were required. We developed a library of operations via files of class-based EOL operations, this allowed the complexity of the bodies of EVL constraints to be simplified greatly. One could define helper operations within the EVL files however we felt that the library-based approach to be more effective as it also allowed for re-use by other components, e.g. queries over the model. This approach proved to be highly maintainable, one might otherwise find the same or similar operations occurring in multiple files. Transformations tend to span the model hierarchy, hence we created a single ETL file for each source-to-target transformation required. Having control over the design of the source document parsers allowed us to define a document schema (XSD) that was sympathetic to our intended target metamodel. This helped to constrain the complexity of the source to target transformation rules, although design of the XSD to EMF model transformations requires some care. Again, the use of EOL operations helped to constrain the size of the transformations, and the use of EVL helped to ensure that such transformations resulted in the generation of valid models. Separate folders were created for the metamodel definition (Emfatic used to generate ecore), generated models, operations (EOL), constraints (EVL) and transformations (ETL); this is a general idiom we have used and is illustrated in Fig.21. A common structure was adopted for the Ant workflows dealing with the various model management activities as illustrated in Fig.22. Using a common structure helped to reduce errors and enforced a degree of discipline. Such workflows are the leaves of what can sometimes extend to a tree, with the higher-level workflows providing orchestration. One of the disadvantages we have observed with the model transformations offered by ETL is that they are programmatic, there is no higher-level view illustrating (e.g.) the relationship between source and target metamodels. Whilst such a view might not capture the complexity of certain transformations, it may be possible to use such a view to auto-generate the basic structure of the transformation, e.g. the skeletal transformation rules.

5.3 Metamodel Instantiation

At the time of development of the metamodels, the only route by which we could instantiate our metamodels with source data (bulk population) was via XML. The

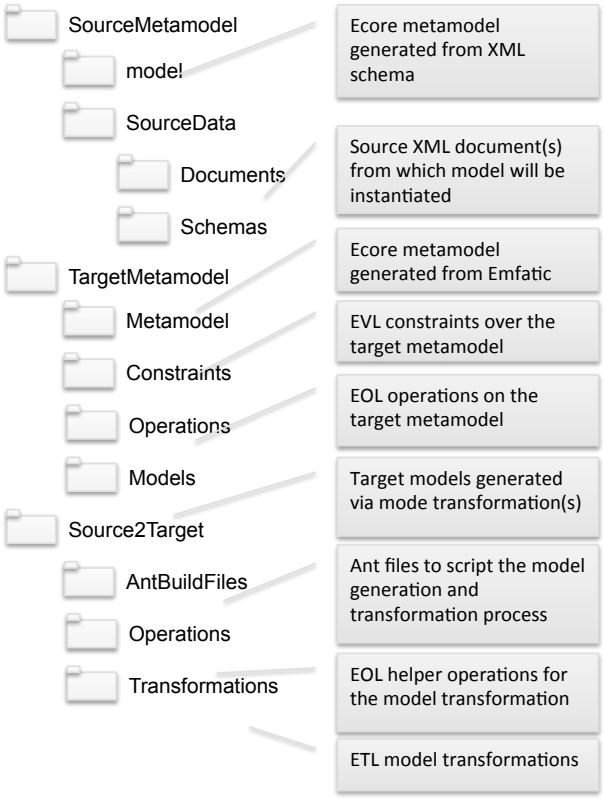


Fig. 21 Example Directory Structure

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="GenericWorkflow" default="Main">
        <!-- Ant Taskdefs for ant-contrib -->
        <!-- Import common properties -->
        <!-- Import Source model properties -->
        <!-- Import Target model properties -->

        <!-- Print out the property definitions for diagnostics -->
        <target name="PrintProperties"/>

        <!-- Load the models required for the transformation -->
        <target name="RegisterModels"/>

        <!-- Generate the target model -->
        <target name="GenerateTargetModel"/>
                <!– Transform the Source model into the Target model -->
                <runtarget target="LoadSourceModel"/>
                <runtarget target="LoadTargetModel"/>
                <epsilon.etl src="${EtlFile}"/>
        </target>

        <!-- Provide referenced model components to ETL transformation -->
        <target name="LoadSourceModel"/>
        <target name="LoadTargetModel"/>

        <!-- Run constraint checks on the generated Target model -->
        <target name="ValidateTargetModel"/>

        <!-- Delete those models we no longer require -->
        <target name="TidyingUp"/>

        <!-- Run the workflow -->
        <target name="Main" depends="RegisterModels">
                <runtarget target="PrintProperties"/>
                <runtarget target="GenerateTargetModel"/>
                <runtarget target="ValidateTargetModel"/>
                <runtarget target="TidyingUp"/>
        </target>
</project>
```

Fig. 22 Generic Ant Workflow

main source document (the INDE file) is an ASCII file, although rather terse it was a simple task to create a parser under ANTLR to convert the file into XML. Since we wished to validate all work products at the earliest opportunity an associated XML schema was written; again, this was a simple task. The availability of an XML schema allowed us to auto-generate an EMF metamodel in Eclipse, and this, in turn, facilitated the writing of a transformation in ETL to instantiate our Network Initialisation domain model. However, we have encountered a situation on another (unrelated) project where an XML schema could not be written for the relevant XML source document. In such instances we would parse the source XML document directly from Epsilon via EOL, although the absence of an XML schema makes debugging more challenging as one discovers the scope of variability of the source document. Therefore, we would tend to advocate the XSD-based auto-generation of a source metamodel followed by a transformation into the target metamodel and subsequent validation via constraints. We would also encourage the reader to validate the source XML document before parsing into the model to guard against propagation of errors.

5.4 Deploying to the Users' Desktops

The tool is developed under Eclipse and deployed to the user community in the form of a JAR file packaged with a small number of support files. Deployment was very simple, requiring only that the user's desktop machine provide a suitable Java Virtual Machine (JVM). The tool was delivered on a single CD-ROM containing all data required for its use; the user had only to unzip to a folder on the files system. Whilst the customer's target environment was Windows-based, the tool could be deployed onto a Unix-based environment with ease. Although we envisage that there would be productivity advantages in migrating to an Eclipse plug-in based RCP architecture, we do not anticipate that we would be able to leverage deployment benefits via the use of update sites as a result of the nature of the deployments because these tend to be on classified standalone networks. Although this introduces a maintenance overhead (manual deployment), the highly modular nature of the models is expected to provide some mitigation.

The tool deployed to users' desktops comprises a Graphical User Interface (GUI) through which the user is able to select and load Network Initialisation Data Catalogue models, once loaded the user may then interact with the model. For the sake of expediency, we chose to implement the tool's workflow in Java, and, for reasons given earlier, we also chose to implement the GUI in Java. The tool's architectural model is illustrated in Fig.19, and follows the well-known Model-View-Controller pattern [13]. The Java GUI code inter-

faces with the underpinning metamodel via simple calls to the Epsilon system.

The user interface was kept very simple, providing a set of combo-boxes from which the user could drill into the model to identify the relevant Community, Participant, and Data Set to be used for the queries. Each of the three queries over the model is provided via a button and a text window is provided to give feedback to the user, for example constraint violations would be presented here. An example of the tool's GUI is provided in Fig.23. Graphical results of the queries are presented in the form of a DSL, the timeslot map.

5.5 Summary of Experience

Development of the network initialisation analysis tool proceeded well, and it has proved to be a successful project, providing benefits to the customer and also prompting the identification of new model-based opportunities in other projects in a diverse set of domains, such as business process modelling. At the beginning of the project the customer requirements were not immediately clear, the TDL domain experts required a tool to perform the analysis task based on user-defined criteria for two complex queries. Also, the form in which the results should be presented was not clear, initially it appeared that a textual report would be adequate. Working closely with the TDL domain experts in the initial phase of the project served to de-risk the project by providing the customers with confidence of our model-based approach, demonstrating the art of the possible whilst also clarifying the requirements and confirming our understanding. As this was our first model-based project to adopt use of the Epsilon toolkit we also worked closely with the relevant domain experts at the University of York, this helped to answer many questions and also ensured that we made best use of the components provided by the toolkit. There have been two updates to the tool since its initial deployment in 2010, the first was a minor correction to the Java View-Controller component, the second was a more significant extension to the Java View-Controller component and a minor extension to the ANTLR INDE file parser and associated schema to support the requirement to accommodate standalone networks. Neither of these changes required any revision to the underlying metamodel of the network initialisation domain, we feel this illustrates the benefit of enforcing strong separation between the View-Controller and the Model components. Applying constraint checks at each stage in the process also serves to improve the robustness of the solution, this has been achieved via a combination of the XML schema and constraints over the models. This approach also simplifies fault finding greatly. For example, the tool was able to identify the use of unexpected platform names by the network design authority when the network design was up-issued; the architecture helped to pinpoint the location of the error and prevent propagation to downstream components.

User feedback confirms that the tool has fulfilled the customer's requirement and the customer has indicated that he would be willing to place further work with us in the future, although this is clearly dependent upon identifying a suitable business opportunity with an attractive ROI. As has been described above, the tool has seen the incorporation of a new requirement to support an alternative network structure (the *closed* network) and now supports all forms of network architecture used by the customer's programme. Demonstration of the tool as part of an end of year presentation to TDL-related projects elicited further opportunities for additional model-based tool development within the TDL domain and department reports have resulted in metamodelling opportunities in other domains, such as ontology and business process modelling.

We did not observe any issues regarding performance throughput or model size. Generation of the network initialisation data catalogue model from the source data takes in the region of 5 minutes and is performed via the user navigating to the relevant source files and clicking a button to parse the data into the tool. Models are parsed from source data only once, thereafter the models are serialised as XMI files. The EOL queries, whilst quite complex, run in a matter of seconds, including rendering of results by Java/SWT, representing a significant improvement over the manual process that appeared to require in the region of 30 minutes by a domain expert (the manual approach did not include rendering of results as a timeslot map). Hence, the tool provides both a considerable saving of effort and makes a skilled task accessible to less experienced engineers; an added benefit is that timeslot analyses are more likely to be performed as a matter of course because it is now a simple task. We also report that we are using the same metamodelling tooling with models more complex and several times larger than that reported here, again with no significant performance issues, in fact model transformation generally appears to perform better than XSLT and is also arguably more readable.

Notwithstanding the difficulties associated with the integration of Ant workflows with the Epsilon toolkit, there was a general lack of state of the art support for debugging model management programs[7]. This was most evident when developing model transformations using ETL (and is similarly the case when merging models using EML and when performing model to text transformations via EGL). We found it necessary to eject text

[7] Since this work was carried out, Epsilon has been augmented with Eclipse-based debugging tools.

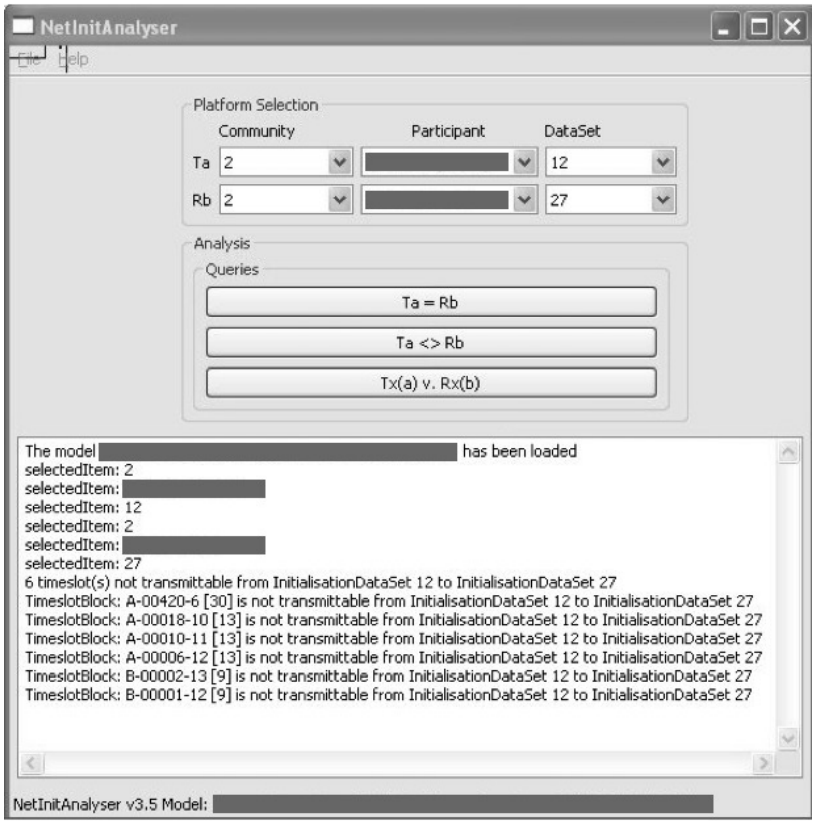


Fig. 23 Graphical User Interface

from each transformation rule to provide a bread crumb trail of execution to aid debugging, since failure of an ETL script generally does not result in the saving of a partial/erroneous model and much of the debugging information relates to the exception trace of the internals of Epsilon. Whilst an admittedly primitive approach to debugging, we defined a 'debug flag' in the Ant workflow such that we could choose to run the transformation in a debug mode when diagnostic information was required. It would have been preferable to be able to set breakpoints in the ETL rules and to be able to inspect the source and target trees as is possible in some XSLT engines. ETL, EML, and EGL components tended to be developed in a top-down style as a result of the limited support for debugging.

The provision of pre and post-condition processing blocks in ETL transformations was found to be a useful feature. We generally use the pre-condition processing block to eject diagnostic information regarding the models to be processed and generated, such as a count of the number of elements of a certain type; again, this is guarded by the debug flag mentioned above. The post-processing block is used for finalising of the generated model, for example traversing the model to ensure all appropriate slots have been populated, or ejecting diagnostic information such as the number of elements created, which may assist with debugging. The basing of the Epsilon languages on a common core provides the ability not only of importing EOL operations in an ETL transformation, but also the ability to declare helper operations directly within the ETL file itself; this we found to be useful for operations of relevance only to the transformation.

In Table 1 we summarise the general characteristics of the EOL operations provided to support the Network Initialisation Data Catalogue model. In Table 2 we present a similar view of the EVL constraints defined over the model. Finally, in Table 3 we present the characteristics of the ETL transformations developed to populate the target metamodel from the two source documents. It should be noted that the operations in the

Table 1 Class metrics - EOL

Number of Classes	35
Number of EOL files	35
Number of EOL helper classes	3
Average LOC for EOL file	25.03
Average no. Operations per class	3.23
Average LOC per Operation	7.75

three helper classes were ultimately embodied into Epsilon EOL as primitive operations, such as the operations

pad() and *trim()* on class String. We also adopted an idiom of providing a *toString()* operation on each class as an aid to debugging - this was subsequently integrated with exeed [19] to provide a more readable model editor output whilst working in Eclipse. Hence, the number of operations per class could actually be somewhat reduced.

Table 2 Class metrics - EVL

Number of Classes	35
Number of EVL files	26
Average LOC for EVL file	39.00
Average no. Constraints per class	2.66
Average LOC per Constraint	14.68

Table 3 Class metrics - ETL

Number of ETL files	2
Average LOC for ETL file	204.50
Average no. transformation rules per file	16.50
Average LOC per transformation rule	12.39

It can be seen form the metrics presented above that the Epsilon language components serve to provide a parsimonious implementation of a complex domain. The completed application comprised a total of approximately 675 lines of Java to provide the GUI and source document parser interface, in addition to approximately 2450 lines of EOL/ETL/EVL; hence, a fully-featured graphical application required approximately 3KLOC. Initial prototyping leads us to believe that migration to an Eclipse plugin-based RCP architecture would reduce the Java LOC count quite considerably as much of the GUI processing is sported directly by the Eclipse plug-in framework. Not only is the solution believed to be highly modular, enforced by the strict separation of the model from the view-controller, but there are clearly opportunities to reuse the domain-specific metamodels as we seek to build more profound and integrated TDL domain capabilities. Horizontal transformations into domain-independent metamodels, such as document metamodels, graph metamodels, etc. support the reuse of such metamodels. A majority of the effort expended in metamodel reuse generally lies in the development of the transformations, and is related to the semantic distance between the models concerned.

Whilst the model-based network initialisation analysis tool has been well received by the customer there are believed to be some areas where an alternative approach might be followed in subsequent work. The approach followed was largely pragmatic, based on de-risking the programme by demonstrating components of the overall solution to the customer to validate our understanding of the problem domain and to drive out and, where necessary, firm up the customer's requirements. As a result of this approach, and the migration to the Epsilon technology we have a technologically fragmented solution: document parsing via ANTLR, GUI development using Java and SWT, and model management using Epsilon. A more coherent approach would be adopt a more native Eclipse-based approach, such as migrating the document parsers to using the Xtext technology [17]. Another area where we believe there is room for improvement is the GUI layout. Whilst functional, a more graphically rich approach has been investigated using the Eclipse plug-in architecture leading to development of an RCP application. This prototype tool allows more user interaction with the model, allowing the user to explore the network design model for the source and target platforms using a tree viewer ([18]) rather than via context-dependent combo-boxes. The user may select one item in each of the two trees and click a button to perform the required interoperability analysis function, or click on some item to browse via a properties view. This prototype tool has the advantage of being both more user-friendly and open to adaptation via its plug-in based architecture.

5.6 Comparing the Approach

We cannot offer a direct comparison between application development following a model-based approach and that following a more typical object-oriented software development paradigm because this was not one of the goals of the project (we did not develop an equivalent system using more typical technologies), hence this section is illustrative. We have developed software intensive systems using typical object-oriented paradigms, such as that described by Texel & Williams [42], using the UML and various popular languages, e.g. Python, Ada95, C++. One of the characteristics of the project reported here is its tactical nature. Prior to this project we had been modelling various aspects of the TDL domain, such as the message structure (e.g. see Fig.4). However, network initialisation was a new theme (to us), and our experience of TDL was not directly applicable. Hence, none of our existing models were immediately relevant to undertaking the project, rather it was the expectation that, leveraging from our metamodelling experience, we would be able to bring powerful new tools to bear on the problem domain and provide an attractive ROI. Another characteristic of the project was the lack of a detailed set of requirements, the general end goal of the project was understood by all parties, but the strategy to achieve this goal wasn't clear at the outset. Consequently, we adopted a prototyping approach to de-risk and clarify requirements.

One of the first activities to be undertaken was for us to try to identify and understand the source data to be used to support the analysis of the timeslots. As Fig.15 shows, two distinct, but related documents were identified, for which parsers were required in order to make the source data available to the analysis tool. The output of the parsers were XML documents for each of which an XML schema (XSD file) was created; this allowed machine validation of the documents generated by the parsers.

Given that we are able to parse and validate the source material and hold it in a machine processable form (XML and an associated schema), we were able to realise an immediate benefit from the EMF tooling via the auto-generation of the necessary XML parsers. It takes only a few minutes to generate an EMF model from an XML schema (XSD), complete with source document (XML) parser from which the model may be populated. This has two benefits: (1) it is much more rapid than writing a parser by hand in a conventional XML-aware language such as Java or Python, and (2) it forces the developer to create an XSD, thereby providing the opportunity to validate source data prior to population of the model. Clearly, there is a benefit associated with identifying errors closer to the point of origin as this reduces the scope of the debugging task.

Development of the Network Initialisation metamodel was performed largely in isolation from the domain experts. It was apparent that the modelling activity required a finer level of detail than the domain experts had considered relevant, this was particularly the case when dealing with inheritance hierarchies and the introduction of abstract classes, but less so when discussing associations between entities. As a consequence, the metamodel was 'written' by the modeller using Emfatic [10] and class diagrams were generated from the Ecore metamodel for discussion with the domain experts, where necessary. The choice between generating Ecore metamodels from Emfatic versus class diagrams is one of personal preference, at least one member of the team preferred the graphical view, whilst the modeller preferred the textual view; however the EMF tooling allows the generation of one view from the other, hence maintaining consistency between views is trivial. However, we see little difference in terms of productivity. In contrast with more conventional approaches a major benefit provided by the Eclipse tooling is that the (meta)model is the code; there's no clear distinction between modelling and software development. Hence, both (meta)model and code (both operations and constraints) appear indivisible. Eclipse provides users with the ability to auto-generate simple editors that may be used to create and validate example models. Additionally, Epsilon supports the use of HUTN [7] where models may be written in terms of the Abstract Syntax and instantiated against the metamodel. Our development process is illustrated in Fig.13. Again, the choice of HUTN over the use of a graphical editor is a result of personal preference. However, we feel either approach to be generally quicker than hand-crafted code as the Eclipse tooling provides the model parser for free, with the added benefit of remaining within the same development environment. Hence, changes to the metamodel may be tested very quickly and easily. Again, we would suggest that the model-based approach is superior to the more classical code-based approach by adapting to changes more easily.

Development of model transformations is provided via ETL. Transformation rules must be coded via a set of *rules* using EOL plus ETL extensions, there is no graphical alternative. When writing transformations it is often necessary to write helper operations using EOL directly. EOL is a powerful high-level language deriving from the OCL [8]. ETL is a rule-based pattern matching language that builds on EOL. For a separate project we had reason to transform a number of XML documents using XSLT and Python. Although the two projects are not directly comparable, we believe the creation of model transformations in ETL to be more rapid and the resulting transformations far more compact than achieved via a more conventional approach. However, ETL suffered from a lack of debugging tools.

Checking of the well-formedness of models is provided by the EVL; again this is an extension to the EOL and is similar to the OCL [8]. Many modern programming languages provide built-in support for run-time checking of types and also user-defined exception blocks. Hence, one may argue that EVL does not offer any development advantages over and above the abstraction provided by its high-level, and in terms of development effort this may be true. The benefit of EVL over more conventional approaches is, rather, the separation of all exception handling concerns into one or more components such that the other language components (ETL, EML, EGL, etc.) are not littered with exception handling code that can compromise their readability. However, we are aware that Aspect Oriented Programming (AOP) takes the idea of separation of cross-cutting concerns much further [43].

Reflecting on the productivity benefits we perceive as being offered by a model-based approach over a more conventional approach, and as summarised above, we would venture to suggest that the model-based approach offers a productivity benefit in the region of an order of magnitude, given similar skilled levels of practitioners in conventional or model-based approaches. We would suggest further that benefits are also realised during the maintenance phase of a project as a result of the explicit linkage between model and code (the model *is*

the code). The introduction of a new requirement (a new query over the model) was accommodated easily, and changes to accommodate *closed* networks served to highlight the ease with which changes could be accommodated in the ANTLR grammar (itself a model) compared to changes to a document parser written in a conventional programming language (in this case Java). Finally, the well-focussed nature of the domain-specific metamodels makes the opportunity for reuse possible. For example, horizontal transformations provide the opportunity to reuse metamodels from different domains, such as the document metamodel. Vertical transformations support the reuse of other TDL metamodels, such as the Data Dictionary and Message Catalogue metamodels, to provide more profound capabilities within the TDL domain; for example checking the correctness of the standard itself.

Ultimately, the success of this project may be determined by the realisation of the ROI predicted at its outset. However, its success may also be measured by the interest garnered by the model-based approach within BAE Systems. Achievement of the ROI is largely outside our control, the figure for ROI was based on figures provided by the sponsoring project based on expected utilisation of the tool which was, in turn, driven by their development plan. The tool has been well-received by the project and, if they make the expected level of use of the tool, the predicted ROI will be achieved. Whilst this is not the only model-based project to have been undertaken on behalf of BAE Systems, it has helped to generate significant interest across various strands of the business and the model-based approach continues to be pursued.

5.7 Future Work

We embarked on a modelling programme to address some of the shortcomings identified in the TDL domain, with a focus on Link 16/TADIL-J due to its general applicability to BAE Systems' air platforms. A majority of our modelling activities related to the standard itself, working bottom-up: Data Dictionary, Message Catalogue, and behavioural specifications, with the aim of adding rigour (e.g. see [15]). Hence, modelling of the network initialisation domain was initially felt to be a tactical decision to pursue a clear opportunity to deploy a model-based tool within a relatively short timeframe (less than 12 months from the opportunity being identified to fielding of the solution), although with no explicit connection to our emerging metamodels of the Link 16 standard itself. The tool has been in regular use since it's initial deployment in 2010 and has been subject to one significant upgrade to accommodate *closed networks*, although we see little opportunity for further significant enhancements with this customer. There are other TDL-related programmes running within the air sector of the business, however use of the INDE file format is currently unique to one project. Thus there is limited opportunity for deployment of additional copies of the existing tool, although there does exist the possibility of extending the tool to support additional file formats as the underlying metamodel of timeslots is expected to be common to all TDL projects. We have also discussed the possibility of performing additional queries over the network design model, such as calculation and analysis of the electromagnetic spectrum for a given platform or configuration of platforms. This is believed to be a useful addition to support 'quick look' queries because there are legal restrictions imposed by the Civil Aviation Authority (CAA) via the Frequency Clearance Agreement (FCA); infringements are monitored centrally.

Subsequent prototyping was performed to migrate the solution to an Eclipse plugin architecture supporting deployment in the form of a configured Rich Client Platform (RCP). The strict separation of the Model from the View-Controller components provided a relatively simple migration to the Eclipse plugin architecture, requiring very little application code, considerably less than was required to write the initial tool. No changes were required to the metamodels, document parsers or Epsilon components. Use of expandable/collapsable tree browsers provided an improvement to the user interface, allowing us to make the query definitions more easily understandable to the novice and also supporting the browsing of model properties, e.g. as one navigates the model from a Community to Groups and/or Participants, to the Initialisation Data Sets, down to the individual Timeslot instances.

One of the major benefits we see in pursuing a model-based approach is the possible integration of the network initialisation metamodel with our existing Link 16 metamodels to provide more profound capabilities by linking the static message structures with the run-time blueprint provided by the network design authority (see Fig.5), this would provide a significant level of insight into the platform implementation. Also the observation that a number of TDL-related programmes adopt similar, rather than identical technologies suggests that there may be scope for the development of a Product Line Architecture (PLA) of tools based on a common framework of metamodels. Such a PLA may be achieved more easily via migration to the Eclipse plug-in model, which we have prototyped successfully by migrating the Network Initialisation Data Catalogue metamodel into a plugin-based RCP architecture. In this way tools could be customised for specific programmes, thereby providing only the functionality required, as opposed to tools providing the superset of functionality, some of which may be irrelevant to a particular programme.

6 Related Work

This section discusses related work in the areas of TDL modelling and the use of Model-Driven Engineering (MDE) in industry.

6.1 TDL Modelling

Sorroche [23] discusses modelling of Tactical Data links and identifies its importance in combat modelling. Moon et al. [24] identify modelling of a discrete event system as a key component of combat modelling. There is relatively little work reported in the literature regarding the formalism or application of model based approaches (i.e. use of executable models) to Tactical data Links. A notable body of work is that of the identification and application of the Discrete Event System Specification (DEVS) formalism, introduced by Zeigler [25–27]. DEVS is a formal specification of a discrete event system that enables a large discrete event system to be specified by hierarchically decomposing the system into modules. The decomposition results in two types of specifications: the hierarchical structure of the system modules and the internal structure, or the state transition, of the system modules. Mak et al. [29] discuss an automated testing framework based on DEVS modelling and simulation formalism, XML and System Entity Structure (SES). This framework is part of the Automated Test Case Generator (ATC-Gen) funded by the United States Department of Defense Joint Interoperability Test Command (JITC) to support the mission of standards compliance and certification. ATC-Gen development is composed of four stages:

1. Rule Capturing
2. Rule Set Analysis
3. Rule Formalisation
4. Test Generation

The rule capturing stage captures and formalises the MIL-STD-6016C [1] in XML format. A rule set analyser determines useful relationships among rules. Rule formalisation consists of selecting and formulating the test sequences. Finally, tests are generated as DEVS C++ test models, these are executed against the System Under Test (SUT) using a Test Driver. The Test Driver performs SUT conformance testing by inducing the testable behaviour expressed in the models into SUT and checking the responses for accuracy [29]. This work is interesting because it seeks to formalise the Link 16 standard such that it is processable by a machine. It is noted, however, that the underlying philosophy of the application of the DEVS approach differs from that discussed by this paper insofar as the DEVS approach provides a testing oracle built around a reference implementation of the Link 16 standard, candidate terminals are subjected to testing against this oracle in order to evaluate conformance against the standard. Although the authors identify a number of issues against the Link 16 standard [28] they do not accommodate the evaluation of interoperability between two terminals, rather interoperability is assumed given conformance with the standard. Also, the approach is intended to confirm (or refute) interoperability of an actual terminal; clearly there are significant costs involved in developing a terminal to the required state, hence the identification of any significant issues are likely to be quite expensive to rectify. In contrast our approach supports the investigation of interoperability criteria from models that could be developed much earlier in the life-cycle.

6.2 MDE in Industry

Mohagheghi et al. [30] report that there is little empirical evidence of the acceptance of MDE in industry and that there are very few reports on industry experience with MDE [31]. They discuss the findings of an empirical study undertaken by the authors, using developers from four large industries participating in a research project. Their findings suggest that model-based approaches were useful for investigating some problems of complex systems, although the methodology and tools were immature and not generally perceived as easy to use. MDE experiences within Telefonica [34] and SAP are described in [32]. The common challenges faced by both companies are the skill set or training in MDE (meta-modelling, domain specific languages) required by developers (domain experts) and the lack of tool support. In our case, the software engineers with MDE expertise were heavily involved in the project, the issue was gaining sufficient TDL domain expertise required to undertake the modelling work. We were able to obtain this expertise by consulting with the TDL domain experts, particularly in the earlier phases of the project.

Kirstan et al. [35] describe the results of an initial qualitative case study evaluating costs and benefits of MDE of embedded software systems in the car industry. Results of the qualitative case study led to a global study by Broy et al. [44] that concluded that MBD can bring significant cost savings in the car industry which is driven by functional evolution and hardware revolutions. Experiences from three industrial cases (SAP, Telefonica and WesternGeco) are summarised by Mohagheghi et al. in [33]. The four major benefits of MDE are listed as follows:

1. Abstraction and hiding details of complexity
2. Communication with non-technical staff
3. Simulation and model-based execution
4. Model-based testing

We concur with the benefits regarding abstraction, communication and testing based on our experience discussed in this article. However, we would add to this

the opportunities for model reuse via model transformations.

Hutchinson's thesis [36] provides an empirical assessment of MDE in industry that includes the analysis of several case studies. The thesis concludes that many different MDE approaches are being actively used in industry and delivering significant benefits. All variants of MBD were covered in his thesis, including both domain-specific modelling languages and UML-based methods. Four papers arising from his thesis are [45–47] and [48] that point to important social, organizational and managerial factors affecting MBD success or failure. The authors in [45] report that the success or failure of MBD is affected as much by cultural considerations as by purely technical ones. Whittle et al. point out in [48] that domain knowledge is crucial in MBD and the chances of MBD being successful increase when team members have domain knowledge, modeling as well as coding skills. We concur with this as domain expertise could have resulted in faster progress of our project with fewer clarifications and misunderstandings. Whittle et al. also report on key findings relevant to education, that is, on how MBD needs to be taught [48, 49]. According to their findings, industry success is more likely when MBD is applied bottom-up while MBD is generally taught top-down.

Torchiano et al. [37] report on the benefits and problems of software modelling and model-driven techniques on the basis of a survey (web-based questionnaire) with a sample of 155 software professionals within the Italian industrial sector. The sample consisted of industries across a number of domains such as IT (67%), Services (9%), Telecommunications (6%), Manufacturing (4%) and others (14%). According to the authors, simple models provided benefits such as:

- support in design definition
- improved documentation
- higher quality

Model-driven techniques played a significant role in productivity, platform independence and standardisation. The problems reported were a mix of technological and human factors, i.e. a lack of supporting tools and lack of domain competencies.

Several other case studies within industry and their benefits are identified in the literature. Baker et al. [38] provides a large case study describing the experience of 15 years of MDE at Motorola, demonstrating significant benefits in quality and productivity. Krogmann's case study in [39] compares conventional software development with Eclipse-based model driven development. This case study demonstrates that a model-driven approach could be carried out in 11% of the time of the conventional approach, while simultaneously improving code quality. Another industrial case study [40] reported that the application of MDE in their project resulted in a performance (productivity) increase of 2.6 times compared with a legacy project. Karna et al. in [41] show that the benefits of Domain-Specific Modelling provide an increase of at least 750% in developer productivity and greatly improve quality of code and development process.

7 Conclusions

Model-Driven Engineering (MDE) has been applied successfully to a complex subject matter domain, military Tactical Data Links, as a vehicle to produce end-user support tools to fill gaps in a TDL tool suite. The capabilities of the current generation of metamodelling tools, such as Eclipse, EMF, Epsilon and XML/XSD, have sufficient industry strength to be able to support this need. They also make it easier to develop a solution that is more open to future extension, possibly in ways that were unforeseen at the time of their inception. In our experience, MDE can be applied successfully when skilled engineers and subject matter experts are able to work together closely, however, increased adoption of MDE raises issues related to the availability of a different skill base compared to that required for a conventional software development project.

8 Acknowledgments

The support of BAE Systems, Military Air and Information (MAI) business unit is gratefully acknowledged. This body of work would not have been possible without the support of the Tactical Data Link community within MAI. At the time of the undertaking of this body of work both Dr. Holmes and Dr. Ajit were employed by BAE Systems.

The support of The University of York is gratefully acknowledged. BAE Systems and The University of York have a long history of collaboration. Staff at the University provided valuable guidance and support regarding the use of the Epsilon technologies. The development of Epsilon has been partially supported through a number of EC co-funded FP6 and FP7 projects including ModelWare, ModelPlex, MADES and INESS.

The support of CGI IT (UK) Ltd. is gratefully acknowledged.

The support of The University of Northampton is also gratefully acknowledged.

References

1. US Department of Defense, Tactical Data Link (TDL) 16 Message Standard, MIL-STD-6016C, 2004

2. NATO Standardization Agency, Standardization Agreement, Tactical Data Link Exchange - Link 16, (STANAG) 5516, Ed. 4
3. Asenstorfer, J., Cox, T., Wilksch, D., Tactical Data Link Systems and the Australian Defence Force (ADF) - Technology Developments and Interoperability Issues, DSTO-TR-1470 February 2004
4. TADIL J, Introduction to Tactical Digital Information Link J and Quick Reference Guide, June 2000, http://www.globalsecurity.org/military/library/policy/army/fm/6-24-8/tadilj.pdf
5. Joint Data Link Management Organisation (JDLMO), Network UKJP0005B, Initialisation Data Catalogue, Network Design Cell, Final Release Version number 1.0 August 2008
6. Tactical Data Link Network Design Station, TNDS Interface Control Document, Issue 2.9 (UK), HP Enterprise Services Defence & Security UK Ltd. February 2011
7. OMG, Human-Usable Textual Notation (HUTN) Specification, Version 1.0, August 2004
8. OMG, Object Constraint Language, Version 2.2, February 2010
9. Parr, T., The Definitive ANTLR Reference: Building Domain-Specific Languages, Pragmatic Programmers, May 2007
10. Daly, C., Emfatic Language Reference, http://www.eclipse.org/epsilon/doc/articles/emfatic/
11. Holzner, S., Ant: The Definitive Guide, O'Reilly, 2005
12. Kolovos, D., Rose, L., Paige, R., The Epsilon Book, http://www.eclipse.org/epsilon/doc/book/
13. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., A System of Patterns - Pattern-Oriented Software Architecture, Wiley, 1996
14. Object Management Group, Jishnu Mukerji, Joaquin Miller, MDA Guide, 2001, http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf
15. Johnson, J., Holmes, C., Improving System Dependability via a Model-Based Approach to Standards, MIT BAE Systems Fall Conference, October 30-31, 2007
16. Johnson, J., Holmes, C., Model-based approach to a complex requirement-design domain: TDLs, 7th Annual Conference on Systems Engineering Research 2009 (CSER2009), Loughborough University, April 20-23, 2009
17. Xtext 2.4 Documentation, April 2013, www.eclipse.org/Xtext/documentation.html
18. Clayberg, E., Rubel, D., Building Commercial Quality Plug-Ins, 2nd Ed., Addison-Wesley, 2006
19. Dimitrios S. Kolovos, Richard F. Paige, Fiona A.C. Polack. Agile Model Editing in the Eclipse Modeling Framework using Executable Metamodel Annotations. In Proc. 1st Towers of Models Workshop, TOOLS EUROPE 2007.
20. Dave Steinberg, Frank Budinsky, Marcelo Paternostro, Ed Merks. *EMF: Eclipse Modelling Framework*. Eclipse Series. Addison-Wesley Professional, second edition, December 2008.
21. Dimitrios S. Kolovos, Richard F.Paige and Fiona A.C. Polack. The Epsilon Object Language. In *Proc. European Conference in Model Driven Architecture (EC-MDA) 2006*, volume 4066 of *LNCS*, pages 128–142, Bilbao, Spain, July 2006.
22. Dimitrios S. Kolovos, Richard F. Paige, Fiona A.C. Polack. A Framework for Composing Modular and Interoperable Model Management Tasks. In *Proc. Workshop on Model Driven Tool and Process Integration (MDTPI), ECMDA*, Berlin, Germany, June 2008.
23. Sorroche, J., Modeling Tactical Data Links, in Engineering Principles of Combat Modeling and Distributed Simulation, pp. 537-578, Anonymous, Wiley & Sons, Inc., 2012
24. Gon Kim, T., Moon, I., Combat Modeling Using the DEVS Formalism in Engineering Principles of Combat Modeling and Distributed Simulation, pp. 479-510, Anonymous John Wiley & Sons, Inc., 2012
25. Zeigler, B., P., Ed., Multifaceted Modeling and Discrete Event Simulation, London: Academic press, 1984
26. Kim, T., G., Zeigler, B., P., The DEVS formalism: Hierarchical modular system specification in an object oriented framework, in Proceedings of the 19th Conference on Winter Simulation - WSC '87, pp. 559-566, New York, 1987
27. Zeigler, B., P., Tag, K., G., Praehofer, H., Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, San Diego, [Calif.]: Academic, 2000
28. Zeigler, B., P., Simulation-based Testing of Emerging Defense Information Systems, 2006 http://acims.asu.edu/wp-content/uploads/2012/02/AuburnTalk.ppt
29. Mak, E., Mittal, S., Hwang, M., Nutaro, J., J., Automated Link-16 Testing Using the Discrete Event System Specification and Extensible Markup Language, The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, vol. 7, pp. 39-62, 2010
30. Mohagheghi, P., Gilani, W., Stefanescu, A., Fernandez, M, A., An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases, Empirical Software Engineering, vol. 18, pp. 89-116, 2013
31. Mohagheghi, P., Dehlen, V., Where Is the Proof? - A Review of Experiences from Applying MDE in Industry, Model Driven Architecture Foundations and Applications, vol. 5095, pp. 432-443, 2008
32. Mohagheghi, P., Fernandez, M., Martell, J, Fritzsche, M., Gilani, W., MDE adoption in industry: Challenges and success criteria, in , M. V. Chaudron, Ed. Springer Berlin Heidelberg, 2009, pp. 54-59
33. Mohagheghi, P., Gilani, W., Stefanescu, A., Fernandez, M., A., Nordmoen, B., Fritzsche, M., Where does model-driven engineering help? Experiences from three industrial cases, Software & Systems Modeling, 2011
34. Evans, A, Fernandez, M., A., Mohagheghi, P., Experiences of Developing a Network Modeling Tool Using the Eclipse Environment, Model Driven Architecture - Foundations and Applications, vol. 5562, pp. 301-312, 2009
35. Kirstan, S., Zimmermann, J., Evaluating costs and benefits of model-based development of embedded software systems in the car industry, Results of a qualitative case study, in Proceedings Workshop C2M: EEMDD "from Code Centric to Model Centric: Evaluating the Effectiveness of MDD (ECMFA.2010), 2010
36. Hutchinson, J., An Empirical Assessment of Model Driven Development in Industry, 2012
37. Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., Reggio, G., Relevance, benefits, and problems of software modelling and model driven techniques, A survey in the Italian industry, J. Syst. Software, 2013

38. Baker, P., Loh, S., Weil, F., Model-driven engineering in a large industrial context - Motorola case study, in , Briand, L., Williams, C., Eds. Springer Berlin Heidelberg, 2005, pp. 476-491
39. Krogmann, K., Becker, S., A case study on model-driven and conventional software development: The palladio editor, Software Engineering, pp. 169-176, 2007
40. Kapteijns, T., Jansen, S., Brinkkemper, S., Houet, H., Barendse, R., A comparative case study of model driven development vs traditional development: The tortoise or the hare, in 4th European Workshop on "from Code Centric to Model Centric Software Engineering: Practices, Implications and ROI", Netherlands, 2009
41. Karna, J., Tolvanen, J., Kelly, S., Evaluating the use of domain-specific modeling in practice, in Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling, DSM, Florida, USA, 2009
42. Texel, P., Williams, C., Use Cases Combined with Booch, OMT, UML, Prentice Hall PTR, 1997
43. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Videira Lopes, C., Loingtier, J-M., Irwin, J., Aspect-Oriented Programming, Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Springer-Verlag LNCS 1241, Finland, June 1997
44. Broy, M., Kirstan, S., Krcmar, H., Schatz, B., What is the benefit of a model-based design of embedded software systems in the car industry?, In Emerging Technologies for the Evolution and Maintenance of Software Models, IGI Global, Hersley, 2011
45. Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S., Empirical assessment of MDE in industry, ICSE 2011, pp. 471-480
46. Hutchinson, J., Rouncefield, M., Whittle, J., Model-driven engineering practices in industry, ICSE 2011, pp. 633-642
47. Hutchinson, J., Whittle, J., Rouncefield, M., Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure, Sci. Comput. Program., vol. 89, pp. 144-161, 2014
48. Whittle, J., Hutchinson, J., Rouncefield, M., The State of Practice in Model-Driven Engineering, IEEE Software, vol. 31, pp. 79-85, 2014
49. Whittle, J., Hutchinson, J., Mismatches between Industry Practice and Teaching of Model-Driven Software Development, MoDELS Workshops 2011, pp. 40-47