Towards a Formal Validation of ETL Patterns Behaviour

Bruno Oliveira[1], Orlando Belo[2], Nuno Macedo[3]

[1]CIICESI, School of Management and Technology, Porto Polytechnic, Portugal
[2]ALGORITMI R&D Centre, University of Minho, Portugal
[3]HASLab, INESC TEC & University of Minho, Portugal

Abstract. The development of ETL systems has been the target of many research efforts to support its development and implementation. In the last few years, we presented a pattern-oriented approach to develop these systems. Basically, patterns are comprised by a set of abstract components that can be configured to enable its instantiation for specific scenarios. Even when using high-level components, the ETL systems are very specific processes that represent complex data requirements and transformation routines. Several operational requirements need to be configured and system correctness is hard to validate, which can result in several implementation problems. In this paper, a set of formal specifications in Alloy is presented to express the structural constraints and behaviour of a slowly changing dimension pattern. Then, specific physical models can be generated based on formal specifications and constraints defined in an Alloy model, helping to ensure the correctness of the configuration provided.

Keywords: Data Warehousing Systems, ETL Systems, ETL Patterns, Component-Reuse, Alloy, and ETL Systems Formal Specification.

1 Introduction

The use of patterns is a recurrent practice in most software development areas, in which systems are frequently designed based on existing components, taking advantage of previous knowledge and experience. Nowadays, the importance of using reusable practices and the design techniques that promote them is recognized, contributing to higher software quality and to reduce time and money needed to its implementation and maintenance. Additionally, patterns enforce the use of well-proven practices representing knowledge of broadly accepted standards and techniques [16]. The development of ETL (*Extract-Transform-Load*) processes for *Data Warehousing Systems* (DWS) represent a specific software area and a critical component to any DWS that addresses very specific needs [10]. Each DWS implementation serves its own user community linked to a specific set of business and decision-making processes supported by specific data models. Additionally, ETL designers frequently deal with legacy systems that provide limited mechanisms for data extraction, and data inconsistencies resulting from years of business change or evolution. All these require extreme care and concern from ETL architects and

software engineers in its planning, architecture, design, and implementation. Moreover, current commercial tools that support ETL data migration implementation processes provide specialized transformation tasks resulting in very complex processes represented using proprietary notations and implemented according to specific architectures and philosophies. These practices are by nature error-prone and hard to maintain. Since the development of ETL processes shares several development phases and typical problems related to the other types of software, we believe that a pattern oriented approach can be applied through the identification of recurrent procedures or techniques, identifying the cluster of operations needed and abstracting its behaviour to provide its instantiation for specific cases. To support the complexity of the knowledge involved and the application of each pattern to specific contexts, a first approach to formalize ETL patterns is also presented using Alloy [9], a declarative specification language that supports problem structural modelling and validation. Thus, after a brief summary of related work in Section 2, we demonstrate in Section 3 the feasibility and effectiveness of a pattern-oriented approach for ETL development based on the description and formalization of one of the most common used ETL techniques: the *Slowly Changing Dimension* (SCD) with history maintenance (SCD-H). The operational requirements are identified and the respective structural constraints and behaviour formalized using Alloy. Finally, in Section 4, we evaluate the work done so far, pointing out some research guidelines for future work.

2 Related Work

Aiming to reduce ETL design complexity, the ETL modelling has been the subject of intensive research and many approaches to ETL implementation have been proposed to improve the production of detailed documentation and the communication with business and technical users. As far as we know, Köppen [11] firstly presented a pattern-oriented approach to support ETL development, providing a general description for a set of design patterns. The work focuses on important aspects defining patterns for internal composition properties and the relationship between them. However, patterns are represented only at design level, lacking the identification of the main configuration components that could be used to translate them to code. With this work, we intend to go further, encapsulating behaviour inside components that can be reused. In fact, we propose a generator-based reuse approach [5] that uses a generator system. The generator produces a specific instance that can represent the complete system or part of it, leaving physical details to further development phases. For that, we believe that a formal model that describes model constraints and behaviour is needed to support physical generation of ETL processes.

The work presented by other authors should also be referred since represent some important contributions to the area. Vassiliadis and Simitsis proposed a technique for the conceptual, logical and physical modelling of ETL processes [17]. More recently, Akkaoui [4] presented a work based on MDA (*Model-Driven Architecture*) for ETL process development, covering the automatic generation of source code for specific computer platforms using a meta-model based on BPMN (*Business Process Model*

and Notation). The bridge to execution primitives was explored using a model-to-text approach, supporting its execution through some ETL commercial tools. These and other related works [14, 18] revealed very important aspects that were taken into consideration for the approach presented here. However, they fail to provide an integrated approach that focus on the complete ETL lifecycle and to take advantage of work performed in initial development to implementation phases. Additionally, they focus on very granular tasks, resulting in very large and disorganized process, resulting in process inconsistencies and redundancy.

3 ETL Patterns: SCD-H pattern

Patterns have been used in several software development areas as a way to help developers solve recurring problems, promoting the sharing of experience and knowledge obtained across several areas. Patterns can be viewed as a three-part rule expressing the context, the forces that typically occur, and how the solution [8] resolves the forces [3]. Next, we will identify and formalize a common ETL procedure, namely a SCD-H transformation pattern, which is used to preserve changes in dimensions used to track historical data. A subset of common pattern description based structure [3][1] is used to describe its internal composition, complemented by a formalization model in Alloy. When a *Data Warehouse* (DW) is updated, some decisions must be considered in order to maintain a consistent view of its data. Any SCD process embodies well-defined policies representing design patterns to support old and new data over time in a DW dimension context. Several types of SCD can be applied [10], each one considering specific scenarios. In this work, a specific strategy of SCD that preserves history using an auxiliary history table is followed. Basically, this approach considers that any dimension integrates two distinct tables: one table to store up-to-date dimension's data and other to store previous record versions. With this strategy all current and historical records are stored with no limitations and with low process complexity, since they are easier to compute than other approaches [15] - in Fig. 1 we present a specific template that can be used to describe a SCD-H pattern.

Audit tables are used in the *Data Staging Area* (DSA) and provide the record for processing to SCD process according to the operation performed in source systems. When records are new, insert operations are triggered to the dimension table as current data, while the deleted records are marked as inactive in target dimension. A more complex process should be initiated for updated records, since the current versions of updated records should be transferred to history table, partially or totally, depending on the dimension table's SCD specification. Depending on the needs, all or some of these operations can be registered in the system's log file, identifying the record, the operation that was performed, the data source origin and temporal data. The quarantine table also plays an important role in a SCD process, since it supports unexpected scenarios that usually compromise processes. Records with structural errors or data entry errors can be redirected to quarantine tables that store all inconsistent records, identifying (at least) the record, the error that occurred and the

temporal data associated. These records can be posteriorly analysed by specific procedures or even manually in order to be reintroduced in ETL workflow or deleted, helping to identify important scenarios that can be useful to check ETL integrity.

Name: Slowly changing dimension, with history maintenance (SCD-H).

Classification: Transformation pattern.

Problem: Dimension tables store data that can change slowly over time. In such cases, we need to track these changes to support historical data reporting. How can current and historical data be properly stored?

Context: DWS are built based on the concept of temporal data. The facts stored in fact tables are linked to a specific date in which they occurred, as well as other dimensions that can be also affected by time. Data coming from information sources can be modified to reflect a change in a certain point of time that should be preserved in a specific dimension, in order to maintain the consistency of the data warehouse.

Solution: The SCD process begins when the data that was changed is available for populating the target system. Typically, this data is stored using specific audit tables that keep records composed by the attributes (*Att*) with history maintenance (*SCDAtt*) for a specific dimension table along with the operation (*Operation*) that record was subject, the operation date (*Date*), and the dimension surrogate key (*Skey*): $auditData = \langle Skey, SCDAtt_1, \dots, SCDAtt_n, Operation, Date \rangle$.

Fig. 1. An example of a template for describing a SCD-H pattern

Moreover, log track techniques also have an important role in a SCD-H pattern since they are used to recover the system from unexpected situations. For example, if some critical error happens, the process can restart only from the point when the error occurs, useful mainly when large volumes of data are involved. After processing all records, the audit table will be cleared and the records processed will be loaded into the target dimensional table. In order to support all these structural and operational constraints, we developed a specific model for representing the structure of the pattern (Fig. 2). This model represents a subset of the ETL patterns meta-model for the representation of the SCD structure and all the objects supporting its structure. The SCD class is a specific type of transformation patterns that can be specialized in several types if necessary. We distinguish only two types: one that holds no historical data (data is simply replaced by the new one) and another that keeps the history of the changes occurred in the dimension table. To support the SCD–H pattern, a specific set of metadata should be provided to produce the instances of the patterns. The *DataObject* class represents the several types of data source/target objects that can be used in SCD-H context. The *Audit table* holds data extracted from data sources and respective operation and time data, the *Dimension* that stores current data, the *History* that stores historical data for each record, the *Quarantine* that represents the object that stores noncompliant data, and the *Log* object that stores all operations performed. Each one of these objects will have fields from several types, revealing specific operational characteristics of each data object. For example, the state of each record (to signal active and inactive attributes) is identified using a specific field.

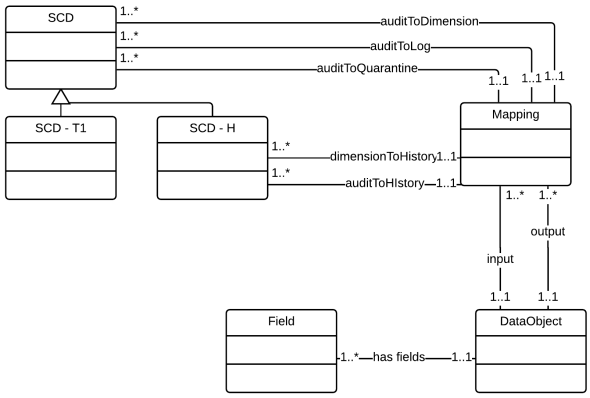

Fig. 2. Class diagram for SCD pattern structure representation

The correspondences between fields of each data source should be defined according to the constraints defined for each mapping: the *auditToQuarantine* relationship represents the mapping between audit table and dimension table, which is related to the source of data and target dimension, the *auditToQuarantine* that represents the mapping between the dimension table and the quarantine table to store fields that for some reason could not be stored in the target dimension, and the *auditToLog* that represents the relationship between the audit table and the target log object storing the operations performed, including the non-successful ones. The *dimensionToHistory* and *auditToHistory* relationships describe specific features of the SCD-H pattern, representing the mapping between dimension fields to history dimension fields, and audit table fields to history dimension fields, respectively. Although the pattern structure can be described in a straightforward manner using class models like the one presented in Fig. 2, such is not the case for several richer structural constraints that specify the integrity of the pattern. These must be considered not only at the level of the pattern configuration but also by the pattern operational behaviour. To provide a simple and solid specification of a SCD pattern amenable to being automatically analysed, its description was embedded into a formal specification.

Alloy is a lightweight formal specification language, whose Analyser provides support for automatic assertion checking, within a bounded universe, by relying on off-the-shelf constraint solvers. The flexibility and object-oriented flavour of the Alloy language render it well-suited to specify and analyse software design models, addressing both complex structural constraints [2][12][7] and behavioural constraints imposed by transformations [6][13]. An embedding of ETL pattern specifications into Alloy would not only provide a formal specification of their structure and behaviour, but would also allow their fully automatic verification, ensuring that the pattern preserves the consistency of the system. Fig. 3 presents an excerpt of the Alloy specification of ETL patterns and related concepts, formalizing the meta-model structure and providing a new degree of detail to the class model presented before in Fig. 2.

```
abstract sig Field{}
sig SKField, ControlField, VariationField, DescriptiveField extends Field{}
sig DateField, OperationField, ErrorField extends ControlField{}

abstract sig DataObject{fields: some Field, keys: some SKField, (...)}
fact dataobject {all o : DataObject | o.keys in o.fields,(...)}

pred consistentDataObject[s:State,o:DataObject] {
 all r1,r2 : o.rows.s |
  (all f : o.keys | f.(r1.values) = f.(r2.values)) => r1 = r2
    (...)
}
sig Mapping {inData: one DataObject, outData: one DataObject, association:
Field -> Field}
fact mapping {
 all m : Mapping | m.association in m.inData.fields -> m.outData.fields
    (...)
}
pred consistentMapping[s:State,m:Mapping] {
    consistentDataObject[s,m.inData]
    consistentDataObject[s,m.outData]
    (...)
}
abstract sig Pattern{}
abstract sig TransformationPattern extends Pattern{}
abstract sig SCD extends TransformationPattern{
    auditToDimension : one Mapping,
    auditToQuarantine: one Mapping,
    auditToLog: one Mapping
}
fact scd {(…)}
sig SCDH extends SCD {dimensionToHistory: one Mapping,
    auditToHistory:one Mapping}
fact scdh {all scd : SCD {
       scd.dimensionToHistory.inData in Dimension
    (...)
}
```

Fig. 3. Basic Alloy specification to support ETL patterns

The class hierarchy can be easily described using Alloy signatures, introducing sets of elements of a certain type in the model. Abstract signatures are used to describe abstract concepts that should be refined by more specific elements, which is the case of top-level signatures *Field*, *DataObject* and *Pattern*. These abstract signatures can then be specialised through extension into concrete objects, mirroring the hierarchy of the model from Fig. 2. Signatures may contain fields of arbitrary arity, embodying the associations between the different artefacts of the specification. For instance, similarly to the Fig. 2 representation, data objects represent repositories that are used to read and write data and contain a set of field declarations that is shared by its extensions: each *DataObject* element is related to a non-empty set of *Field* elements (imposed by the keyword some) that represent data fields used to characterise records, and a set of *SKField* elements used to express the SK fields. These can then be restricted by additional constraints through Alloy facts, like the one stating that every *SKField* is selected from the data object Field elements. Signature mapping represents the association between fields from two data sources, establishing the relationship between attributes from two different sources through binary field association. Additional constraints impose, for instance, that a mapping is only valid if it

associated fields from the input (*inData*) and the output (*outData*) data sources. Facts represent constraints that are enforced in the system. However, certain constraints should not be enforced but rather preserved by the ETL procedure - this denotes the notion of correctness in the specification. The behaviour of the patterns should then be checked to assess whether they guarantee the consistency of the system. These predicates include properties like the uniqueness of the values in SK fields and the referential integrity between the SK fields of the data sources associated to the mappings. Other relevant signatures, omitted in the figure, are specified in a similar manner, like those regarding the target and historical dimension and their respective constraints, as well as the log and quarantine objects.

```
sig State {}
sig Value {}
sig Row {values: Field -> lone Value}
abstract sig DataObject{(...),rows: Row -> State}
fact rows {
    all s : State, o : DataObject, r : o.rows.s | r.values.Value = o.fields
}
pred addToDimension [s,s': State, r,r': Row, scd: SCD] {
    r in scd.auditToDimension.inData.rows.s
    r' not in scd.auditToDimension.outData.rows.s
    scd.auditToDimension.inData.rows.s' =
        scd.auditToDimension.inData.rows.s - r
    scd.auditToDimension.outData.rows.s' =
        scd.auditToDimension.outData.rows.s + r'
    all f : scd.auditToDimension.association.Field |
        f.(r.values) = f.(scd.auditToDimension.association).(r'.values)

    (...)

}

assert addToDimensionCorrect {

    all s: State, s': s.next, scd: SCD, r: Row |

        (consistentSCD[s,scd] and addToDimension[s,s',r,scd]) =>

            consistentSCD[s',scd]

}
```

Fig. 4. Excerpt of the Alloy specification for SCD-H pattern for dynamic behaviour simulation

The pattern signature is then specialised to represent SCD patterns, containing the expected mappings, like *auditToDimension*, *auditToQuarantine* and *auditToLog*. These fields describe the relationship between the audit object used for the SCD process and the respective target repositories: a dimension object that will receive data from audit table, a quarantine object that holds non-conformed data that was excluded from the dimension object, and a log object that will preserve all operations performed by the audit, dimension and quarantine objects. These mappings enable the preservation of variation, control, surrogate key and data fields between data objects, which are imposed by facts defined over both the SCD and the SCD-H signatures. For instance, for the *dimensionToHistory*, the correspondent mapping is established between a dimension object data as input and a history object data as output. Additionally, several signature facts were used to enforce the correctness of each mapping, i.e., that the mapping associates fields of the same nature. This means that related fields are correctly mapped and that invalid relationships are avoided. A

predicate is then defined to embody the notion of consistent SCD. In this case, this amounts for checking whether the mappings of SCD are themselves consistent, as defined above. This property can then be automatically processed by the Alloy Analyser, either to simulate instances that conform to the specification or to check for the correctness of concrete instances. The specification presented in Fig. 3 embodies the static constraints of ETL patterns. However, we are interested in checking the correctness of the process as data is collected from the sources and ETL procedures are executed. In Alloy, dynamic behaviour is encoded through well-established idioms, like the local state idiom followed in this work [9]. Roughly, a new signature State is introduced in the specification, representing different states of the system, modelling the notion of evolving time. Artefacts that are expected to evolve in time are then appended with a State element, denoting their value in each instant of time. In our scenario, only the data contained in each data object are expected to change - the structure of the data objects and the patterns is static. This dynamic version of the specification is presented in Fig. 4, where a *Row* signature was introduced to describe the association between a *Field* and a specific *Value*. Data objects were then extended to contain a dynamic row field representing its data in each instant of time. An additional fact restricts rows to assign values to the fields of the parent data object.

The system evolution is then modelled through declarative predicates that relate two different states. Thus, if the static components of the specification represent the pattern configuration, dynamic components define the pattern behaviour based on those configurations. Fig. 4 depicts, as an example, the predicate *addToDimension* that models the behaviour of the pattern insert operation, involving the audit and target dimension objects. Being declarative, these definitions are usually comprised by pre-, post- and frame conditions. In this case, the pre-conditions restrict the operation to be applied only if the surrogate keys of the audit row that will be processed do not exist in the dimension table. Otherwise the update operation should be applied instead. The post-conditions state that the row is removed from the audit table and inserted in the target dimension, preserving the data as defined by the associations of the SCD mappings. The frame conditions state that every other artefact, like the history table, stays unchanged. The two states are represented by the predicate *s* and *s'* parameters - whenever the rows field is annexed with the *s* elements it represents the data in the pre-state, while rows at *s'* represents the data in the post-state. Once these operation predicates are defined, the specification can be automatically analysed by the Alloy Analyser to check whether they preserve the consistency constraints defined above within a bounded universe. These are defined in the assert clause in Fig. 4, which tests whether the execution of the *addToDimension* operation preserves the consistency of the system. The check command effectively instructs the Alloy Analyzer to verify this assertion for a specific scope. It has shown that the three defined operations indeed preserve the specified consistency predicates, providing an increased level of confidence on the correctness of the procedure. Additional constraints and level of detail can be added to the specification to verify more complex properties.

4 Conclusions and Future Work

This paper proposed a pattern-oriented approach that allows for the implementation of ETL processes with a higher level of abstraction. With this pattern-oriented approach, the knowledge and best practices revealed by several works can be put in practice using a set of software patterns that can be applied to the ETL development life cycle: from model representation to its physical implementation primitives. A specific design pattern - SCD-H - was identified along with its skeleton, representing its structure, constraints and behaviour according to specific rules. The presented approach keeps a specific template and instance as separated layers, since users need to provide data about data, i.e., users describe data repositories and its contents in structural terms and a specific generator will generate the respective code based on the primitives previously established and using specific data. The SCD-H pattern can be applied to any ETL scenario. To support this process an excerpt of a formal specification in Alloy was presented, providing an automatic way for analysis and searching for false assertions through the generation of counterexamples. That way, patterns not only become easier to use than in the common granular approach, but can also be easily reused to produce better software, because models can be checked using a powerful simulation engine before its execution. This work is a first attempt to specify formally ETL patterns in Alloy, representing the static and behavioural specifications of a SCD-H pattern. At short term, a more complete Alloy specification for ETL patterns will be developed, particularly in what is concerned to behaviour formalization, assertion checking and exception and error handling scenarios. Additionally, several other patterns need to be formalized to provide a complete package specification for ETL patterns. A complete validation engine is also planned, translating the pattern configuration to Alloy models, allowing the ETL designers to seamlessly and automatically check the consistency of the developed patterns.

Acknowledgments

This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2013. Our thanks to Alcino Cunha, from HASLab R&D Centre, for the comments and suggestions he did during the preparation of this paper.

References

1. Aalst, W.M.P. Van Der, Hofstede, A.H.M. Ter, Kiepuszewski, B., Barros, A.P., "Workflow Patterns". Distrib. Parallel Databases. 14, pp. 5–51, 2003.
2. Anastasakis, K., Bordbar, B., Georg, G., Ray, I., "On challenges of model transformation from UML to Alloy". Softw. Syst. Model. 9, pp. 69–86, 2010.
3. Appleton, B., "Patterns and Software: Essential Concepts and Terminology". Object Mag. Online, 1, 2000.

4. Akkaoui, Z., Zimànyi, E., Mazón, J.-N., Trujillo, J., "A model-driven framework for ETL process development". Proc. ACM 14th Int. Work. Data Warehous. Ol. 45–52, 2011.
5. Biggerstaff, T.J., "A perspective of generative reuse". Ann. Softw. Eng. 5, 169–226, 1998.
6. Büttner, F., Egea, M., Cabot, J., Gogolla, M., "Verification of ATL transformations using transformation models and model finders". In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp. 198–213, 2012.
7. Cunha, A., Garis, A., Riesco, D., "Translating between Alloy specifications and UML class diagrams annotated with OCL". Softw. Syst. Model. 14, pp. 5–25, 2013.
8. Gabriel, R.P., "Patterns of Software Tales from the Software Community". Architecture. 239, 1996.
9. Jackson, D., "Software Abstractions: Logic, Language, and Analysis". MIT Press, 2012.
10. Kimball, R., Caserta, J., "The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data", Willey, 2004.
11. Köppen, V., Brüggemann, B., Berendt, B., "Designing Data Integration: The ETL Pattern Approach". Eur. J. Informatics Prof. XII, 2011.
12. Kuhlmann, M., Gogolla, M., "From UML and OCL to relational logic and back". In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp. 415–431, 2012.
13. Macedo, N., Cunha, A., "Least-change bidirectional model transformation with QVT-R and ATL". Software & Systems Modeling, pp. 1-28, 2014.
14. Muñoz, L., Mazón, J.N., Trujillo, J., "A family of experiments to validate measures for UML activity diagrams of ETL processes in data warehouses". Inf. Softw. Technol. 52, 1188–1203, 2010.
15. Santos, V., Belo, O., "No need to type slowly changing dimensions". In Proceedings of IADIS International Conference Information Systems 2011, Avila, Spain, 11 - 13 March, 2011.
16. Sommerville, I., "Software engineering", Pearson Addison Wesley, 7th edition, 2004.
17. Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitis, M., "A framework for the design of ETL scenarios". In Proceedings of the 15th international conference on Advanced information systems engineering. pp. 520–535. Springer-Verlag, Berlin, Heidelberg, 2003.
18. Wilkinson, K., Simitsis, A., Castellanos, M., Dayal, U., "Leveraging business process models for ETL design. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp. 15–30, 2010.