Automating the Evolution of Data Models for Space Missions. A Model-Based Approach

Lynda Ait Oubelli, Yamine Aït-Ameur, Judicael Bedouet, Benoit Chausserie-Lapree, Beatrice Larzul

Automating the Evolution of Data Models for Space Missions. A Model-Based Approach

Lynda Ait Oubelli[1,2(✉)], Yamine Ait Ameur[2], Judicaël Bedouet[1], Benoit Chausserie-Lapree[3], and Beatrice Larzul[3]

[1] ONERA - The French Aerospace Lab, Toulouse, France
{Lynda.Ait-Oubelli,Judicael.Bedouet}@onera.fr
[2] University of Toulouse, INP, IRIT - Research Institute of Computer Science, Toulouse, France
yamine@enseeiht.fr
[3] CNES - The French Space Agency, Toulouse, France
{Benoit.Chausserie-Lapree,Beatrice.Larzul}@cnes.fr

Abstract. In space industry, model-driven engineering (MDE) is a key technique to model data exchanges with satellites. During the preparation of a space mission, the associated data models are often revised and need to be compared from one version to another. Thus, due to the undeniably growth of changes, it becomes difficult to track them. New methods and techniques to understand and represent the differences, as well as commonalities, between different model's revisions are highly required. Recent research works address the evolution process between the two layers (M2/M1) of the MDE architecture. In this research work, we have explored the use of the layers (M1/M0) of the same architecture in order to define a set of atomic operators and their composition that encapsulate both data model evolution and data migration. The use of these operators improves the quality of data migration, by ensuring full conservation of the information carried by the data.

Keywords: Model driven engineering (MDE) · Data model comparison · Data model evolution · Data migration · Composite evolution operators · Semantic transformation patterns

1 Introduction

Context

Space agencies in Europe like CNES, the French space agency, have been involved in data modelling and in the standardization of data modelling techniques for more than 20 years. They have defined some Consultative Committee for Space Data Systems (CCSDS) recommendations. These recommendations are related to syntactic and semantic data description techniques, long term data preservation, data producer and archive interface. Furthermore, CNES, jointly with the European Space Agency (ESA), have also developed tools to support the recommended approaches and to support data engineering for various space projects.

DOI: 10.1007/978-3-319-66854-3_26

One of these tools is the BEST [1] workbench that came beyond EAST, the Enhanced Ada SubseT. It allows a non-ambiguous description of data formats including syntactic and semantic information. This tool is used in the frame of space projects by scientists and engineers. It allows them to easily describe their data formats and make them evolve, to quickly produce test data conform to the format specification, to access and interpret data without having to write specific code. The formal description of space-related data is crucial and should be taken into account from the early stages of the mission when:

- the space system (one or more satellites, a ground control center, a mission center, etc.) is designed;
- the satellite, once launched, starts to send telemetries and the ground segment starts to send commands;
- a large amount of data is produced, processed, transformed and sent to end-users during the life of the satellite;

All different stakeholders have to understand the data, to be able to interpret it and to make use of it. Any misunderstanding might cause important delays in mission planning. Arguably, a complete and non-ambiguous definition of any kind of data produced is a key factor for meeting the deadlines of the project. This formal definition can be used to generate different pieces of code that will be used in the frame of the project (eg. on-board software, simulation software, etc.). Code generation is highly valuable for a project, since some parts of the application can be updated in a fast and efficient way with a minimum development cost [2]. However, data generated (Data V1) with a particular release of the application (DataModel V1) are not necessarily compatible with another release (DataModel V2). Thus, starting the creation or the generation of data that conforms to (DataModel V2) from scratch is a tedious operation.

Motivations
Research into how complex industrial data models evolve from one version to another and how their data migrate still stands as one of the most scientific challenges to be addressed. This is due to the disability of existing solutions to face with the huge complexity of changes in terms of type and number. Interpreting comparison results of two small data models may be an easy thing. Meanwhile, interpreting comparison results of two huge data models is still a hard task.

Objectives
In this research study, we intend to tackle the following objectives:

- being able to recognize many differences as a unique composite operator;
- ensuring evolution schema by finding structural and descriptive changes;
- full data migration without any loss of information.

To achieve these objectives, we decide to investigate and improve several methods to compare data models (M1 level) governed by structured data-oriented metamodels (M2 level) as shown in Fig. 1. Then, we propose to transform the obtained differences to evolution operators working at data model level (M1) as well as

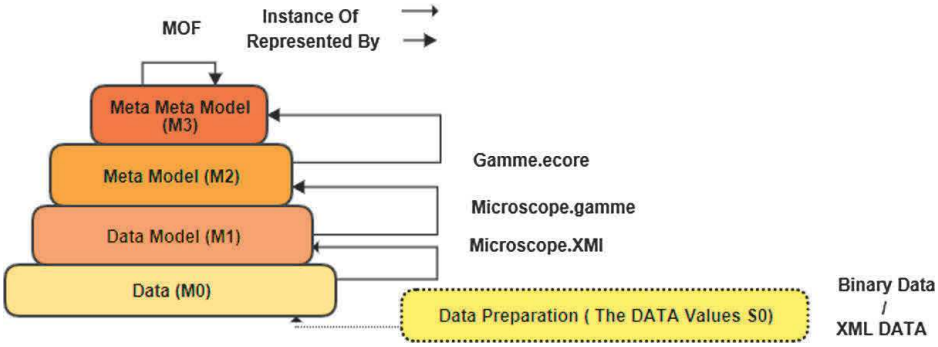


Fig. 1. Four-layer architecture of MDE

data level (M0). At the data model level, an evolution operator defines a data model transformation capturing a common evolution [3]. At the data level, it defines a model transformation capturing the corresponding migration.

Depending on the final usability goals of data and depending on the modeller's skills, the same concept may be modeled in different ways. For example, to define the structure of data with BEST, only *"composition"* relationship can be used (e.g. in the XIF meta-model[1]) whereas *"inheritance"* relationship may be preferred (e.g. in the XTCE meta-model[2]). Such a difference in the way of modeling induces a huge number of syntactic differences between a XIF data model and a XTCE data model, even if they represent the same concept.

Understanding the semantic differences that may exist between the two data models is difficult because these real differences are hidden by a large number of syntactic differences. Once we can recognize these differences as a unique composite operator, we can clearly identify other syntactic differences. This is an extreme example of data model transformation but we often meet such cases while studying how data models evolve. Indeed, modellers often need to reconstruct their model to meet new end-users needs. But, they do not want to completely break the ancient model or make it obsolete. Even if ascendant compatibility is not kept, they try to preserve information contained in old data.

The remainder of this research paper is organized as follows. The next section summarizes the previous related work for the different approaches being used for meta-model evolution and model co-evolution. The theoretical description of the proposed approach to formalize space data model evolution and data migration using semantic composite operators is presented in Sect. 3. A case study is discussed in Sect. 4 followed with conclusions and future work.

[1] The underlying language used to formally define data is XML formatted according to a CNES standard named XIF. XIF is a join implementation of two CCSDS standards: CCSDS 644.0-B-3 Data Description Language EAST Specification and CCSDS 647.1-B-1 Data Entity Dictionary Specification Language.

[2] When exchanging satellite database including telemetry and telecommand definitions with satellite manufacturers or other space partners, we may use another CCSDS standard: 660.0-B-1 XML Telemetric and Command Exchange (XTCE).

2 Related Work

Many researchers studied the discipline of evolution, which was defined by [4] as «*All programming activity that is intended to generate a new software version from an earlier operational version*». This section summarises the different existing approaches for model-based comparison, evolution and migration. Capturing and formalizing evolution stages for composite systems is a challenging task, due to the nature and characteristics of such systems. Furthermore, the rapid changes, the density and the type of differences, which change according to the evolution of end-user's requirements, are the reasons of proposition of a variety of algorithms, methods and evolution laws by scientific community.

2.1 Model Comparison

The history of comparing algorithms is composed of three stages. At the beginning, we compared character to character. For instance, *diff* programs [5], are used to solve the longest common subsequence problem (LCS). They are based on finding the lines that do not change between files. Then, we leveled up by focusing on the attributes and nodes of a structured document. A number of advanced algorithms are being available to capture differences for XML such as XDiff [6] or Aladin [7]. However, even if they give logical results, they lack the ability to recognise a huge number of complex changes in a reasonable time. Finally, we are interested in the meaning of these nodes and attributes by comparing a class node with a class node, an attribute node with an attribute node, for instance, with the assistance of the well accepted EMF framework [8].

2.2 Evolution Operators

The process of capturing and constructing evolution operators have been investigated by [3]. Authors of this paper have worked on the definition of 61 reusable evolution operators (30 are atomic and 31 are composite) with the aim of treating the coupled-evolution of meta-models (M2) and models (M1). The authors have provided a catalog based on EMOF meta-modeling formalism in which they have outlined a set of migration rules specified at a model level. However, even if they tackled the generation of evolution operators, the authors did not discuss the detection mechanism of these operators. This was treated later in [9], where the authors proposed a detection engine of complex changes. They have addressed the two challenges of variability and overlap between evolution operators. A research prototype named COPE [10] was extended into a transformation tool tailored for the migration of models in response to meta-model co-evolution named Edapt [11]. Furthermore, authors in [12] introduced the Silift, generic tool environment able to lift incomprehensible low level differences derived from EMF Compare into representations of user-level edit operations.

2.3 Data Migration

Nowadays, the majority of research teams [9–12] have tackled the phenomenon based on the M2/M1 architecture layers of MDE, where the authors presented a semi-automatic process to co-evolve model-to-model transformations upon metamodel evolution. Authors in [13] introduced a comparison study between various model migration tools such as AML, COPE, Ecore2Ecore and Epsilon Flock. Indeed, each one of them has a set of criteria, that will support users to select the most appropriate tool according to their needs.

3 Proposed Approach

To ease the migration process of data in a critical domain, where each data value vulnerability is so height, a solid protocol for controlling the evolution of data models is required. Therefore, deriving and capturing the set of changes might even be largely different from one version to another.

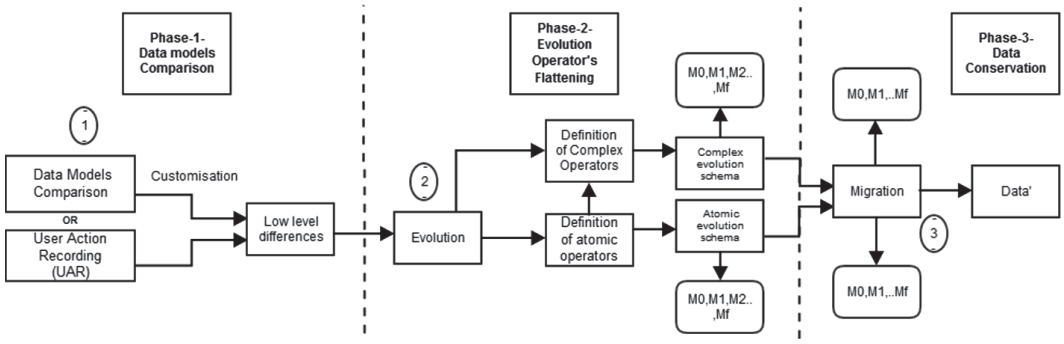


Fig. 2. The pipeline architecture of model-based approach for data migration

In this paper, we propose a novel model-based approach for automating the evolution of data models. An overview of the proposed approach is illustrated in Fig. 2, where three different successive processes (data models comparison, evolution operator's flattening and data migration) should be able to reuse the results of each other. We introduce a pipeline protocol in order to guarantee the conservation of data values during the migration.

It is worth noting that the proposed scenario based on three processes can be applied to other domains, where the main goal of the migration process is data preservation. However, in other cases, where the main purpose is functionalities conservation processing, many other approaches that tackled the comparison, evolution and co-evolution processes are available. They act separately at M2/M1 level in different ways with different tools.

3.1 Phase 1. Data Models Comparison

As shown in Fig. 3, the comparison process is the first step in our approach. It requires two data models (M1 level) that conform to the same meta-model (M2 level). After customizing EMF Compare's matching, differencing and filtering engines, the comparison results will be a syntactic delta which contains equals i.e. commonalities as well as differences *(ResourceAttachementChange, Attribute Change, ReferenceChange and FeatureMapChange)*. These are based on four atomic operators:*Add, Delete, Change and Move*. Lot of efforts are being made by [14] to improve the algorithms of EMF Compare in order to help the developers to customize the engines according to their requirements. But the framework provides low level comparison results that are not always logic or difficult to understand by human as mentioned in [12].

Another alternative to get the differences between the old and the new data models would be to record end-user's actions on the editor (UAR) using action observers. These methods immediately produce a complete evolution schema.

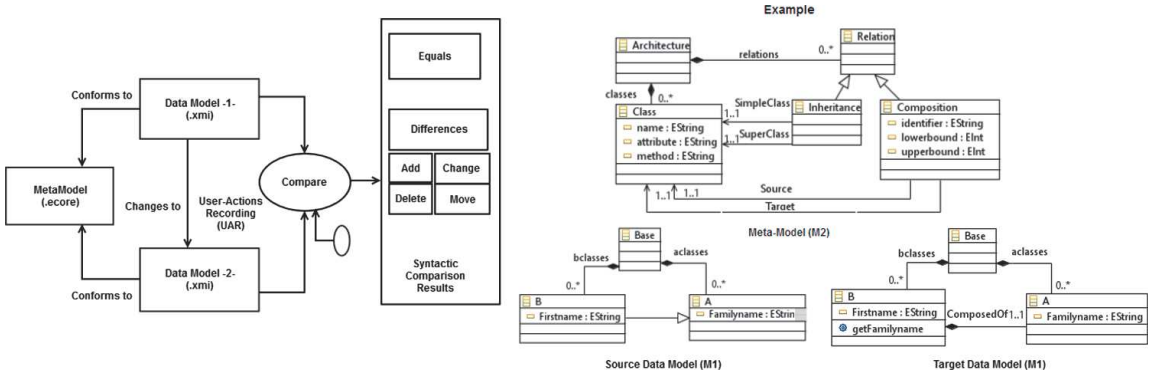


Fig. 3. Overview of the input/output for the comparison process.

3.2 Phase 2. Evolution Operators Flattening

The kinds of differences delivered by the previous phase are at a low level. They are not easy to interpret. Thus, we transform the differences to a set of atomic operators. These operators are independent of the previous phase, defined according to the treated metal-model and help to move to upper level. Most of the time, each difference is represented by one atomic operator. Each cascading difference is represented by the most priority operator. For example: *Add Attribute A to Class C* and *Add Class C to Model M* can be considered as one atomic operator: *Add Class C to Model M*. Indeed, adding a new class implies the addition of all its properties. Furthermore, sometimes we can find differences that cause the same change. For instance, in the case of eOpposite references in Ecore meta-model, modifying one of the two references will automatically update the other side of the opposition. Both changes will be detected as two differences and can be seen as one atomic operator.

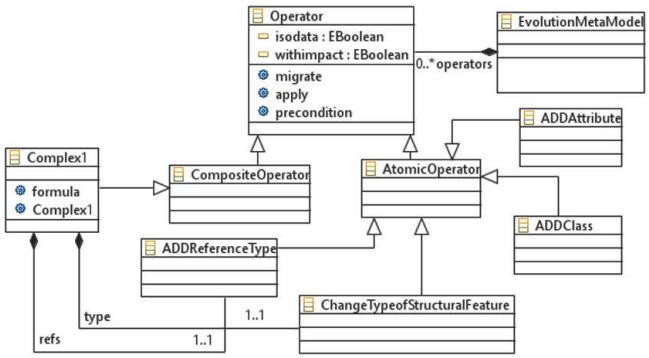


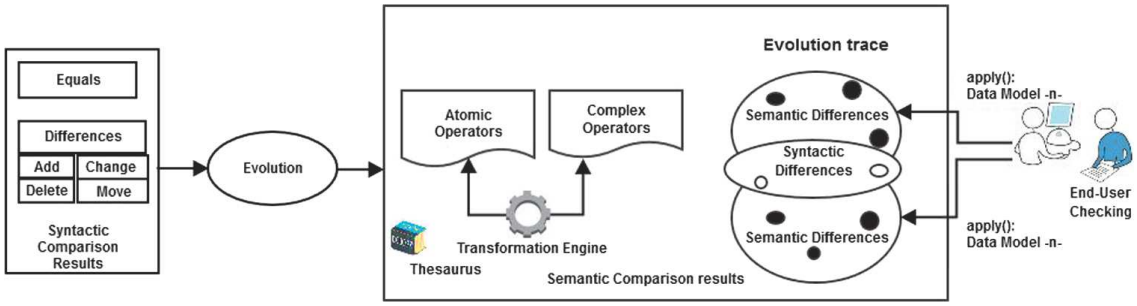
Fig. 4. Evolution atomic and composite operator's meta-model

Once atomic operators are found, we instantiate composite operators (see Fig. 4) as compositions of atomic ones, in order to rise from syntactic differencing delta to semantic one. In practice, we can find implicitly one or many composite operators derived from a composition of many atomic operators. For example, *Pull up attribute A* is considered as a composite operator that is composed of the following atomic operators: *Delete attribute A from class C, Add attribute A to class C'*. Another interpretation of this composite operator is: *Pull up attribute* is composed of the atomic operator *Move attribute A from Class C to its mother class C'*.

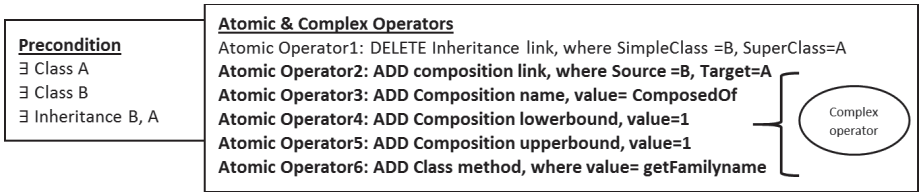
In the proposed approach, our goal is to build an evolution schema with the maximum number of composite semantic operators and minimum number of atomic syntactic operators. Each atomic operator is specified by a precondition an apply and a migrate operation. For example, the atomic operator *Add attribute A to class C* requires the existence of *class C*. Each composite operator is characterized by four properties: *a formula, a precondition, an apply and a migrate*:

- a formula is a sequence of atomic operators that define a composite operator. Identical formulas do not necessarily lead to a composite operator or to the same composite operator, depending on the precondition.
- a precondition responsible for managing the dependency between atomic operators in term of existence, order and priority. For example: *Add Literal L* requires the precondition *Exist Enumeration E*, which checks the existence of the enumeration E in the old data model or the existence of the operator *Add Enumeration E*.
- an apply is used to make evolve the old data model. Apply calls are used to check if the new data model is equivalent to the final evolved old data model.
- a migrate is a property where each composite evolution operator encapsulates its own migration behavior in order to ensure data conservation.

Moreover, to avoid conflicts, we suggest to integrate the end-user to resolve ambiguity during the building of composite operators, as shown in Fig. 5(a). He/she can modify the found evolution schema and apply a checking operation on his/her evolution schema; after calling the different *apply*, the final evolved data



(a) Operator Flattening



(b) Example

Fig. 5. The passage from syntactic comparison delta to semantic one

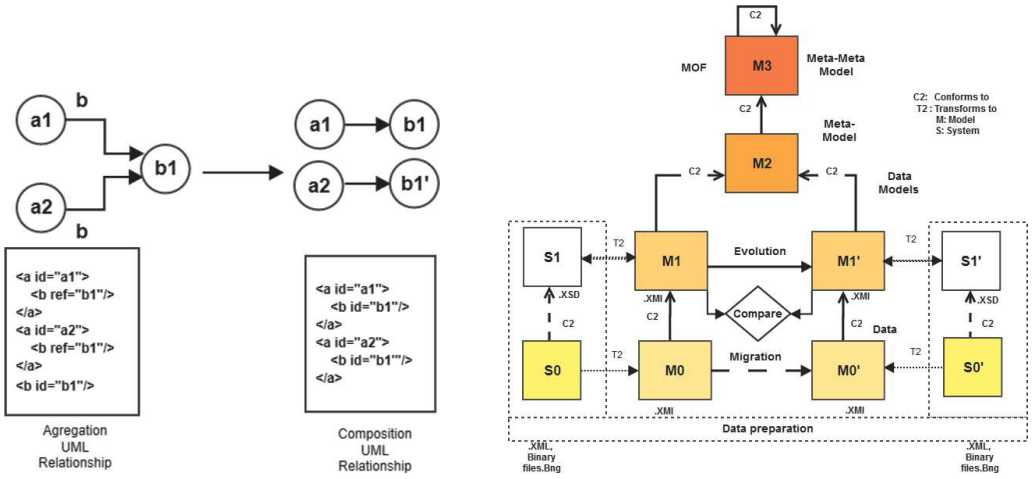
model M must be identical to the original new data model. If some differences are found between these two data models, the evolution schema based on atomic and composite operators is wrong and needs to be corrected by the end-user.

3.3 Phase 3. Data Conservation

In our study, we classify the operators into two categories: safe operators without impact and operators with impact. The impact is considered at the M0 level. For example, the operator *Add Attribute* may need or not a migration of existing and corresponding object instances.

Furthermore, inspired by information theory concepts defined by [15], where the author distinguish between data representing cost side of the system, and information representing the value side, we propose a sub-classification of the operators with impact into two other sub-categories: operators where migration is data-conservative and operators where it is not. Figure 6(a) illustrates the concept by an evolution example of a UML composition relationship to an aggregation UML relationship. In this case we change the data structure but we do not lose any information and we do not need to create additional pieces of information as shown in (Fig. 7).

The last phase of our approach is the migration process as shown in (Fig. 6(b)). We migrate the existing instance attributes in data from the source M0 to the target M0'. We define M0 as instances of the model M1 and S0 as physical data, which represent the M0 instances into different formats.



(a) Example (b) Migration process

Fig. 6. Conservation of data values during the migration process

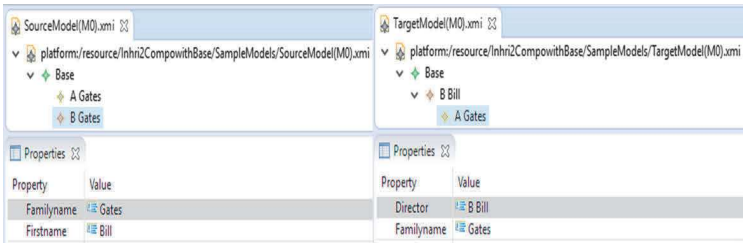


Fig. 7. Example of data values conservation during data models reconstruction

4 Case Study from the MICROSCOPE Project

In this section, we will see how the previous approach is applied to a particular revision of a data model used in the MICROSCOPE project [16]. This project studies the universality of free fall in space thanks to two differential accelerometers supplied by the French National Aerospace Research Center, ONERA, and embedded in a microsatellite from the CNES. While CNES models the telecommands (TC) and the telemeasures (TM), ONERA models the processing of telemeasures. In both cases, there is an intensive use of model-driven engineering to handle the complexity of such project thanks to two respective in-house metamodels Best [1] and GAMME [2]. In the following, we detail the comparison, evolution and migration strategies that could be used to co-evolve parameters used during the telemeasures processing, following the pipeline architecture.

This case study interests in the way of combining two telemeasures: class *Signaux* with two attributes *signal1* and *signal2*. Both attributes were typed by an enumeration called *Signal*, identifying the different telemeasures. Then, it was decided that it is possible to combine other signals than a telemeasure. As shown in Fig. 8 the modeller decided to reconstruct the data model by replacing the two attributes by two composition relationships towards a new class, called *DataAbstract*. The original attributes *signal1* and *signal2* are factorized in a class

DonneesSession, inheriting from *DataAbstract* and owning a *signal* attribute of type*Signal*. In this way, end-users can combine two telemeasures, a telemeasure with another signal or two signals. As a result:

- a new abstract class named *DataAbstract* is added;
- new two classes named *DonneesSession, AutresDonnees* inheriting from *DataAbstract* are added;
- a new attribute named *signal* of type *Signal* is added to the class *DonneesSession*;
- a new attribute named *signalExt* of type *String* is added to the class *AutresDonnees*;
- the types of *signal1* and *signal2* are changed from *Signal* to *DataAbstract*.

4.1 Comparison Process

After using the default engines of EMF Compare 3.0.1, we find many false-negatives (i.e. undetected correspondences) and false-positives (i.e. unexpected correspondences) in the matching results. 247 differences are obtained. However, when moving to release 3.2.0 of EMFCompare, we get better matching results. Moreover, after customizing the matching engine policy (based on id), we get only 17 differences. The idea is to find a unique name to identify each element in the model (for example, the unique name of an attribute is its name concatenated to the name of the container class). EMFCompare considers the two compared versions as two graphs. At the beginning, it makes a matching between ancestors. Each matching is composed of zero or many sub-matches. In other words, EMFCompare uses a top-down matching approach. Next, a differencing engine is employed to obtain all the differences using all the matched elements. Then, we decide to keep some interesting differences: when a data model element is updated or when an added data model element has an impact on data. Therefore, we neglect hidden changes given by cascading EMF *diffs*. Finally, we get 7 differences.

4.2 Evolution Process

In fact, the seven differences provided by EMF Compare (Table 1) are satisfying in term of correctness and precision. However, it is not easy to exploit them. Thus, we decide to transform these low level differences to atomic operators (Table 2) by applying a java transformation template on EMFCompare differencing engine and by running an evolution operator's defined at the meta-model level.

In the end, we obtain three *ADD Classes* as atomic operators and two composite operators *ChangeTypeofStructuralFeature*, that concern type changes of the attributes *signal1* and *signal2* from *Signal* to *Data Abstract*. In this case, our evolution schema is hybrid, i.e. it is a mixture of atomic and composite operators.

Table 1. Syntactic comparison results between two data models obtained by EMF compare customization.

EMFCompare diff	Kind	Matching
ReferenceChangeSpec	CHANGE gType	(GStructuralFeature: signal1 in GClass: Signaux) matched to (GStructuralFeature: signal1 in GClass: Signaux)
AttributeChangeSpec	CHANGE type	(GMetaModel_Element_property: null in GAttribute: Signal1) matched to (GMetaModel_Element_property: Composition in GReference: Signal1)
ReferenceChangeSpec	CHANGE gType	(GStructuralFeature: signal2 in GClass: Signaux) matched to (GStructuralFeature: signal2 in GClass: Signaux)
AttributeChangeSpec	CHANGE type	(GMetaModel_Element_property: null in GAttribute: Signal2) matched to (GMetaModel_Element_property: Composition in GReference: Signal2)
ReferenceChangeSpec	ADD gContents	(GClass: null in GModel: Lap1nDataModel) matched to (GClass: DataAbstract in GModel: Lap1nDataModel)
ReferenceChangeSpec	ADD gContents	(GClass: null in GModel: Lap1nDataModel) matched to (GClass: DonneesSession in GModel: Lap1nDataModel)
ReferenceChangeSpec	ADD gContents	(GClass: null in GModel: Lap1nDataModel) matched to (GClass: AutreDonnees in GModel: Lap1nDataModel)

In the evolution meta-model (Fig. 8) we define four atomic operators *ADDClass, ADDAttribute, ChangeTypeofStructuralFeature and ADDReferenceType*. Each composite operator is a transformation pattern composed of a set of atomic operators. For example, Complex1 as shown in Table 3 is composed of two atomic operators: *ChangeTypeofStructuralFeature* and *ADDReferenceType*. The sequence of both operators is the formula. As shown in Fig. 8, a precondition checks the dependency between atomic and composite operators where:

- *F'* respresents the new gStructuralFeature in the gClass DonneesSession: *signal*;
- *new((f.gType).allFeatures)* represents the new gType of the two new gStructuralFeature signal1 and signal2: *Data Abstract*;
- *F'.gType* represents the gType of the gattribute signal: *Signal*;

Table 2. From differences to atomic operators.

Atomic Operator	GModelElement	Value
ADDgContents	GClass	DataAbstract
ADDgContents	GClass	DonneesSession
ADDgContents	GClass	AutreDonnees
ADDReferenceType	type	Composition
ChangeTypeofStructuralFeature	GAttribute	DataAbstract
ADDReferenceType	type	Composition
ChangeTypeofStructuralFeature	GAttribute	DataAbstract

– *old(F).gType* represents the old gType of the two old gStructuralFeature signal1 and signal2: *Signal*.

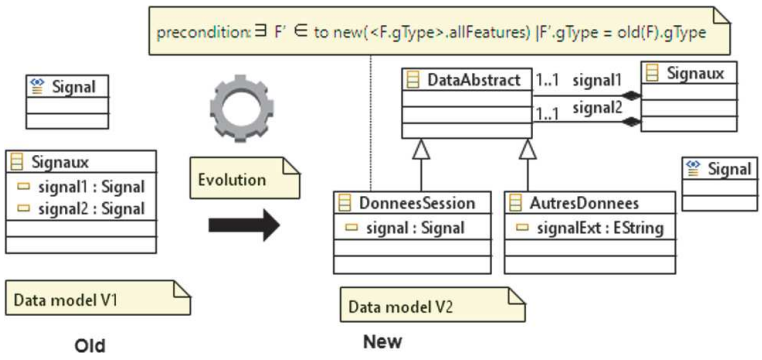


Fig. 8. Evolution from the old to new data model

We use this precondition to check our evolution schema with five operators. In the end, if the precondition is true, a full data conservation is immediately achieved. By calling the apply() of each operator, we can compare the transformed old (the final old data model) with the new version used during the comparison process. In our case the results are positive, i.e. the proposed evolution schema based on evolution operators is correct.

Table 3. Evolution schema based on atomic and composite operators

Atomic Operator	GModelElement	Value
ADDgContents	GClass	DataAbstract
ADDgContents	GClass	DonneesSession
ADDgContents	GClass	AutreDonnees

Composite Operator	GModelElement	Value
Complex1	ADDReferenceType ChangeTypeofStructuralFeature	Composition DataAbstract
Complex2	ADDReferenceType ChangeTypeofStructuralFeature	Composition DataAbstract

4.3 Migration Process

In our case, migration concerns composite operators only. We need to migrate the values of the gattributes as shown in Fig. 9. Initially, we have *signaux* an instance of the *gclass Signaux*. In the end, we arrive to three instances, one is instantiated from the *gclass Signaux* and the two others are instantiated from the *gclass DonneesSession*.

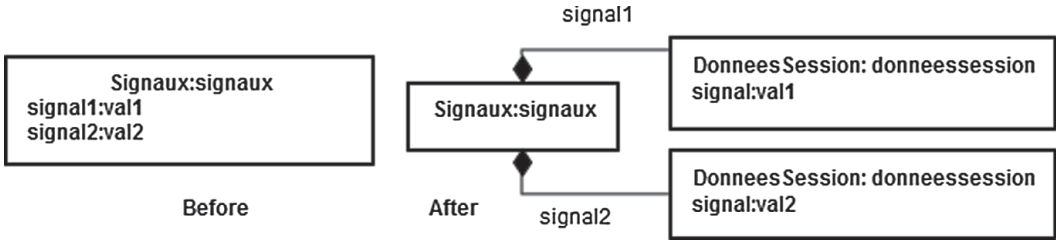


Fig. 9. Data conservation during the migration process

5 Conclusion and Future Work

The data models in space industry evolve very rapidly from simple data models to more and more complex ones. A number of technologies, methodologies and platforms have emerged to ease and formalize the comparison between different revisions, the synchronization between the evolutions of UML, EMF metamodels (M2) and the co-evolution of models. In fact, there are many model evolution and migration tools and each one has its own characteristics. But it is still difficult for end-users to extract the correct evolution. Moreover, most of these tools do not interest in the evolution of data models (M1) and co-evolution of data model instances (M0).

Thus, in this research study, we propose a protocol at M1/M0 level to automate the migration process of our data based on the comparison and on the evolution of their data models. Because reconstruction of composite operators based on the atomic ones provide a rich source of evolution strategy, an approach is being investigated to formalize comparison, evolution and migration processes using a model-based setting. This can be a major milestone for data model evolution with data values conservation as a primary priority during the migration.

Several other research directions to pursue our work can be investigated. First, we are interested in extending our approach to handle during the evolution process semantic meaning of elements by the integration of end-user via a graphical user interface (GUI) and a definition of a thesaurus, that concerns space data models. The thesaurus contains words that belong to the same lexical field. During the comparison process, it is used to check weather the meaning of

data model elements is kept or not, in case we have syntactic changes. Furthermore, we want to validate our approach on more complex data models in term of number and type of changes. We also want to tackle the conservation of APIs. Instead of working on a complex formula to recognize an evolutionary operator, we could look at whether the data API is conserved.

Acknowledgments. Authors would like to express their gratitude to all members of the TCS team: Patrice Carle, Romain Kervarc, Rémi Lafage, Antoine Ferlin and Rémi Plantade from ONERA. This paper relies on their excellent comments and their constructive suggestions.

References

1. National Center for Space Studies (CNES) and The European Space Agency (ESA).: Best (2016). http://goo.gl/3awkpg
2. Bedouet, J., Huynh, N., Kervarc, R.: GAMME, a meta-model to unify data needs in simulation modeling (WIP). In: Proceedings of the Symposium on Theory of Modeling and Simulation-DEVS Integrative M and S Symposium, p. 14. Society for Computer Simulation International (2013)
3. Herrmannsdoerfer, M., Vermolen, S.D., Wachsmuth, G.: An extensive catalog of operators for the coupled evolution of metamodels and models. In: Malloy, B., Staab, S., van den Brand, M. (eds.) SLE 2010. LNCS, vol. 6563, pp. 163–182. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19440-5_10
4. Lehman, M.M.: Software evolution. In: Marciniak, J.L. (ed.) Encyclopedia of Software Engineering. Wiley, Hoboken (1994)
5. Hunt, J.W., MacIlroy, M.D.: An Algorithm for Differential File Comparison. Bell Laboratories, New York (1976)
6. Wang, Y., DeWitt, D.J., Cai, J.Y.: X-Diff: an effective change detection algorithm for XML documents. In: Proceedings of the 19th International Conference on Data Engineering, pp. 519–530. IEEE (2003)
7. National Center for Space Studies (CNES): Aladin. https://logicels.cnes.fr/content/best
8. Toulmé, A.: Intaloi Inc. Presentation of EMF compare utility. In: Eclipse Modeling Symposium, pp. 1–8 (2006)
9. Khelladi, D.E., Hebig, R., Bendraou, R., Robin, J., Gervais, M.P.: Detecting complex changes and refactorings during (Meta) model evolution. Inf. Syst. **62**, 220–241 (2016)
10. Herrmannsdoerfer, M., Benz, S., Juergens, E.: COPE - automating coupled evolution of metamodels and models. In: Drossopoulou, S. (ed.) ECOOP 2009. LNCS, vol. 5653, pp. 52–76. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03013-0_4
11. Vissers, Y., Mengerink, J.G.M., Schiffelers, R.R.H., Serebrenik, A., Reniers, M.A.: Maintenance of specification models in industry using Edapt. In: 2016 Forum on Specification and Design Languages (FDL), pp. 1–6. IEEE (2016)
12. Kehrer, T., Kelter, U., Ohrndorf, M., Sollbach, T.: Understanding model evolution through semantically lifting model differences with SiLift. In: 2012 28th IEEE International Conference on Software Maintenance (ICSM), pp. 638–641. IEEE (2012)

13. Rose, L.M., Herrmannsdoerfer, M., Williams, J.R., Kolovos, D.S., Garcés, K., Paige, R.F., Polack, F.A.C.: A comparison of model migration tools. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010 Part I. LNCS, vol. 6394, pp. 61–75. Springer, Heidelberg (2010). doi:10.1007/978-3-642-16145-2_5
14. Brun, C., Pierantonio, A.: Model differences in the eclipse modeling framework. UPGRADE Eur. J. Inform. Prof. **9**(2), 29–34 (2008)
15. McDonough, A.M.: Information Economics and Management Systems. McGraw-Hill Book Co., New York (1963). p. 11
16. Baghi, Q., Métris, G., Bergé, J., Christophe, B., Touboul, P., Rodrigues, M.: Gaussian regression and power spectral density estimation with missing data: the MICROSCOPE space mission as a case study. Phys. Rev. D **93**(12), 122007 (2016)