Finding conservative schema evolutions by analysing API changes

1st Lynda Ait Oubelli
IRIT/INP-ENSEEIHT, ONERA/DTIS
University of Toulouse
Toulouse, France
Lynda.Ait-Oubelli@onera.fr

2nd Yamine Aït Ameur
IRIT/INP-ENSEEIHT
University of Toulouse
Toulouse, France
yamine@enseeiht.fr

3rd Judicaël Bedouet
ONERA/DTIS
University of Toulouse
Toulouse, France
Judicael.Bedouet@onera.fr

4th Benoît Chausserie-Laprée
CNES- The French Space Agency
Toulouse, France
Benoit.Chausserie-Lapree@cnes.fr

5th Béatrice Larzul
CNES- The French Space Agency
Toulouse, France
Beatrice.Larzul@cnes.fr

***Abstract*—Because verification and validation are important activities in model driven engineering (MDE), verifying interfaces preservation is considered an interesting step to understand the evolution of data models by analyzing their interfaces. The interfaces defined on a data model can be used to define model evolution correctness using observational semantics. In this paper, we propose an approach that supports rigorous analysis, verification and validation of behavioral re-factoring. Our work addresses the problem of data model evolution in a formal modelling and verification setting. We focus on data conservation in the specific context of space engineering, where data models may involve thousands of concepts, relationships and each concept has a number of fields or attributes and each relationship has a number of properties.**

***Index Terms*—data models evolution, data model transformation, data models comparison (Bi-simulation), graphs, labelled transition system (LTS)**

I. Introduction

Because project stakeholders require an easy and safe (behaviour-preserving) technique to update model-based applications, several approaches based on formal methods have been proposed [1]–[3]. This work gave rise to several formal comparison approaches [4]–[6]. In this paper, we propose an approach that supports analysis of models behavior preservation after re-factoring. It consists in checking that the APIs (Application Programming Interfaces) of a source data model still hold on the target data model. To address the problem of data model evolution, we have identified three requirements.

Accessibility. Access to model concepts shall be preserved after model refactoring i.e. source model *getters* or *setters* shall be preserved. Accessibility requirement becomes a path problem in a graph.

Cardinalities. The cardinalities defining the extensions of the relationships between source model concepts (specifying the allowed number or range of instances) shall be preserved after model refactoring.

Knowledge. In order to strengthen concepts evolution, a knowledge base can be associated to the refactoring process to define possible knowledge equivalences or relationships between model concepts. Ontologies are good candidates for such knowledge bases [7].

This paper deals with the *accessibility* requirement. It particularly focuses on the models produced in space engineering.

This paper is organized as follows. Section 2 overviews related work. The proposed approach to handle model evolution and data migration is presented in Section 3. Basic definitions are presented in Section 4. Section 5 summarizes our results, overviews our experiments and positions our approach with respect to the state of the art. Finally, section 6 concludes and provides future work.

II. Related work

The problem of model refactoring has been addressed by several authors with different perspectives.

Application Programming Interfaces (API). They offer operators to process model concepts by encapsulating modelling details. [8] proposed two categories of application interfaces: external and internal ones. External APIs are designed by library maintainers for clients usage while internal ones are used by the library

DOI reference number: 10.18293/SEKE2019-132

code itself. To automatically collect refactoring operations between two APIs versions, [8] uses the *RefFinder* tool, which identifies up to different 52 refactoring types between two API versions. The identified refactoring types are structural, only detectable by mechanical transformations. However, [9] observes that APIs breaking changes are not involved in refactorings. In this case, an application built with an older version of the component API, may fail under a new component API version. When the problem is visible, the application fails to compile or to link. Moreover, it may succeed to compile but its behaviour may be altered [10].

Refactoring. Refactoring-based migration tools are discussed in [11] where the research CatchUp tool is used to update applications. It uses refactorings descriptions to help application developers migrate their applications to a new version. It aims to update applications by recording and playing back the refactorings. Only few refactorings have full records and replay support. According to [10], refactoring at model level is inherently more challenging due to difficulties in assessing the potential impact on structural and behavioral features of the software system.

Data models comparison. Authors in [6] address the problem of user interface (UI) evolution. They focus on the user interface behaviour preservation and study the design process of a user interface resulting from the evolution of a former user interface due to the introduction of new devices and/or new interaction capabilities. Interface behaviors are described by labelled transition systems (LTS) and comparison is handled by bi-simulation of LTS. Furthermore, [4] describe how user interfaces equivalence, with respect to their interaction capabilities and appearance, can be measured. The UI divergences are highlighted, and the possibility of leaving these divergences out of the analysis is provided.

Our previous work [12] proposes an intrusive approach to manage model evolution based on structural differences. It results in a set of evolution operators from source to target models. Models are inspected to identify a set of differences and may produce false positives/false negatives.

In this paper, we propose a non-intrusive approach to handle model evolution. Instead of using a syntactical approach, we rely on API preservation. We consider that a data model evolution is correct if the source data model API is preserved in the target one. The approach is based on path access preservation and graph bi-simulation [13].

III. Handling model evolution and data migration: our approach

In order to handle the semantic data changes involved in the development and exploitation of complex systems in a critical application domain like space engineering, we need to design a rigorous protocol to control the semantic model evolution and data migration.

The approach we propose to compare a source and a target data model relies on 4 steps. Each step manipulates graphs to handle modelling language's semantic evolution. Fig. 1 depicts the defined approach.

- **Step 1. Data models refactoring (interpretation).** It identifies, in each data model, the concepts altered by the evolution. Two input data models will be compared according to these shared identified concepts. According to the latter, both source and target data models are interpreted into a shared model. We use labelled directed graphs (LDG) as ground shared model. Two LDG are produced for source (LDG_s) and target (LDG_t).
- **Step 2. Data models projection.** For each LDG produced from Step 1, a set of labelled transition systems (LTS) with different initial states is produced.
- **Step 3. LTS comparison.** The obtained LTS for both source and target data models are compared using a simulation relationship. Each target LTS shall simulate the corresponding source LTS. When all the source LTS are simulated by the target ones, concept access path preservation is ensured.
- **Step 4. Data conservation.** If step 3 succeeds, source data instances conforming to source and target data models are migrated. The migration procedure is defined depending on the kind of established simulation relation of step 3: strong simulation (source data instances are reused) or weak simulation (source data instances are refactored using the API corresponding to the path identified by the simulation relation).

IV. Formalisation of our approach

For the *accessibility* requirement identified in section I, and according to step 1, we define *LDG* as the unified ground model in which data models are transposed.

A. A formal model for checking data model evolution

In the following, C, $attr$ and Bt denote the set of data model concepts (classes, entities, etc.) of attributes and of basic types (Boolean, Integer, etc.).

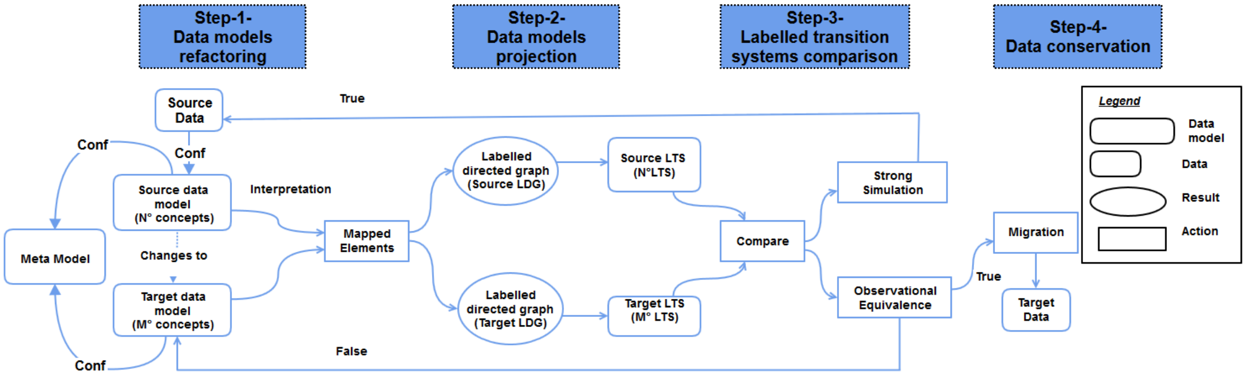
Step-1- Data models refactoring
Step-2- Data models projection
Step-3- Labelled transition systems comparison
Step-4- Data conservation
Source Data
True
Conf
Conf
Source data model (N° concepts)
Interpretation
Meta Model
Changes to
Target data model (M° concepts)
Conf
False
Mapped Elements
Labelled directed graph (Source LDG)
Labelled directed graph (Target LDG)
Source LTS (N°LTS)
Target LTS (M° LTS)
Compare
Strong Simulation
Observational Equivalence
True
Migration
Target Data
Legend
Data model
Data
Result
Action

Fig. 1. A four steps based approach for data migration.

Definition 1: A Labelled Directed Graph $ldg \in LDG$ is a graph $ldg = (V, E)$ where

- $V = C \times \mathbb{P}(attr \times Bt)$ is a non-empty set of nodes. Each node represents a concept and its attributes.
- $E \subseteq V \times label \times V$ is a set of directed edges denoting the relations between the concepts.

For any $e = (v_s, l, v_t) \in E$, v_s and v_t represent the source and target node of edge e. Node $v = (c, \{(a_1, t_1), \ldots, (a_n, t_n)\}) \in C \times \mathbb{P}(attr \times Bt)$ defines

- c as a concept (class, entity, relation, etc.), with
- $\{(a_1, t_1), \ldots, (a_n, t_n)\}$ as a set of typed attributes.

We have considered (1) as a $label \subseteq \{isa, refs, haspart, partof, ref, cast, prop\}$ the set of relations for: inheritance *is_a*, aggregation *refs*, composition *haspart*, reflexive composition *partof*, references between concepts *ref*, casting *cast* and association property *prop*. Other relations may be studied for other analyzed data modelling language.

Definition 2: A labelled transition system lts is a structure $lts = (S, s0, T, \rightarrow)$ where S is a finite number of states, $s0 \in S$ is an initial state, T denotes a set of labels and $\rightarrow \subseteq S \times T \times S$ is a transition relation. The specific label $\tau \in T$ denotes empty label used to model internal actions, i.e., non observable actions in our approach. We note LTS as the set of lts and T^* as the set of all possible sequences built on labels of T [13].

LTS is the projection of graph LDG on each concept, i.e. each graph ldg has many lts with different initial states corresponding to different concepts.

Step1. *Interpretation*

Interpretation is the process that produces a graph $g \in LDG$ from a conceptual model $cm \in CM$ where CM is a set of conceptual models like UML, Entity-Relation (ER), XIF[1].

We denote $CM \xrightarrow{Int} LDG$ and $g = Int(cm)$ the function that describes this process. Each concept (e.g. a class for UML diagrams, an entity for ER, an element for an XIF data model) resp. each concept relation (e.g. inheritance, class association, an entity relation) of cm is interpreted by a node resp. by an edge in the graph g.

Step2. *Projection*

Projection is the process that produces a set $LTS = \{lts\_1, \cdots lts\_n\}$ of $lts \subseteq LTS$ from a graph $ldg = (V, E) \in LDG$ where n correspponds to the number of nodes in g. We denote $LDG \xrightarrow{Proj} LTS$ and $LTS = Proj(g)$ the function that describes this process. The following transformation rules for projection define a lts.

Nodes of $V = C \times \mathbb{P}(attr \times Bt)$. For each node $v = (c, \{(a_1, t_1), \ldots, (a_n, t_n)\}) \in C \times \mathbb{P}(attr \times Bt)$ in g,

- the concept $c \in C$ defines a state $c \in S$
- each type $t_i \in Bt$ defines a state $t_i \in S$
- each attribute a_i defines a transition $(c, a_i, t_i) \in \rightarrow$

Edges of $E \subseteq V \times label \times V$. Each edge $e = (v_s, l, v_t) \in E$ where $v_s = (c_s, \{(a_{s1}, t_{s1}), \ldots, (a_{sn}, t_{sn})\})$ and $v_t = (c_t, \{(a_{t1}, t_{t1}), \ldots, (a_{tn}, t_{tn})\})$ defines a transition $(c_s, l, c_t) \in \rightarrow$.

Initial states for each lts. Finally, each node $v_i \in V$ of the graph $g = (V, E)$ defines the initial state of $lts_i \in \{lts_1, \cdots lts_n\}$.

The projection results in a set of labelled transition systems associated to any data model. Therefore, analysis techniques defined for labelled transition systems can be applied. In particular, our approach uses lts comparison

[1] XML Interchange Format (XIF): A standard in space engineering to define space data models [14].

techniques based on the definition of a simulation relationship.

lts as a model for APIs

An *api* in a set of *API* is made of operations op_i like *getters*, *setters*, *testers* etc. to respectively access, modify or query concepts or attributes of a data model. We note $api = \{op_1, \cdots, op_m\} \in API$.

For a given $lts \in LTS$, we say that an $api \in API$ of a given concept c is valid if and only if for each operation $op_i \in api$ there exists a path, starting from the initial state corresponding to the concept c, which accesses each input and output concepts used by any $op_i \in api$. We say that lts satisfies the *api* API and note $lts \models_a api$.

This definition can be extended to the APIs of any concept in a graph $g = Int(cm)$ resulting from the interpretation of a conceptual model cm. We say that a set $Api \subseteq API$ of APIs defined on g is satisfied if and only if for each $api \in API$ there exists a $lts_i \in Proj(ldg)$ such that $lts_i \models_a api$. We note $g \models_g Api$.

Step 3. *LTS comparison*

Let g_s and g_t be two *ldg*. Let $lts_s \in Proj(g_s)$ and $lts_t \in Proj(g_t)$ be two *lts* with an initial state associated to the same concept c and *api* an API defined on the lts_s on the concept c.

We say that *api* is preserved on lts_t if and only if lts_t simulate lts_s (written as $lts_t \sim lts_s$). Informally, all the paths in lts_s are also paths in lts_t i.e. *api* is still satisfied in lts_t. Formally, we write

$$lts_t \models_a api \Longleftrightarrow lts_t \sim lts_s \wedge lts_s \models_a api$$

Step4. *Data conservation*

Let g_s and g_t be two *ldg* and *Api* a set of API defined on g_s such that $g_s \models_g Api$. We say that a set *Api* of APIs is preserved on g_t if and only if for all $api \in Api$ such that $\exists\ lts_s \in proj(g_s) \wedge lts_s \models_a api$ there exists a $lts_t \in proj(g_t)$ such that $lts_t \sim lts_s \wedge lts_s \models_a api$. Formally, we write

$$\begin{gathered} g_t \models_g Api \\ \Longleftrightarrow \\ \forall api \in Api, \exists\ lts_s \in proj(g_s), \exists\ lts_t \in Proj(g_t). \\ \text{such that } lts_s \models_a api \wedge lts_t \sim lts_s \end{gathered}$$

Definition 3: Finally, we say that a **conceptual model** cm_t **is a correct evolution of a conceptual model** cm_s **with respect to a set** *Api* of APIs if and only if

$$g_s = Int(cm_s) \models_g Api \Longrightarrow g_t = Int(cm_t) \models_g Api$$

Once the conceptual model cm_t is proved to be a correct evolution of cm_s, instances can be migrated. The APIs of the source conceptual model are used to rebuild the instances in the target data model. Some of the produced instances may be partially valued in case cm_t is richer than the source data model.

B. Example

Below, we apply the defined methodology on the example of a conceptual UML class diagram depicted on Figure 2. The objective is to check if the class diagram on the right hand side of Figure 2 is a correct evolution of the one on the left hand side.

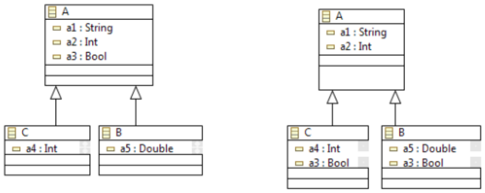

Fig. 2. An example of a data model evolution.

Step1. *Interpretation*

An example of evolution of an UML class diagram is given in Figure 2. The source and target data models are interpreted using the *Int* function leading to two *ldg*. The source data model contains three concepts A, B and C. Concept B and concept C inherit from concept A. Concept A has three attributes $a1$, $a2$ and $a3$. Concept C has one attribute $a4$ and concept B has one attribute $a5$. In the target data model, we decide to push-down the attribute $a3$ from concept A to both concepts B and C.

Step2. *Projection*

We project the source and target *ldg* to labelled transition systems. Since three nodes are identified at the *ldg* level, we obtain three *lts* for both source and target *ldg* as shown on Figure 3 for the model on the left hand side of Figure 2. The initial state of each *lts* is one of the three nodes of the associated *ldg*.

Step 3. *LTS comparison*

In this case study, strong equivalence is not ensured. However, each target *lts* weakly simulates the corresponding source *lts*. One may notice that the opposite does not hold.

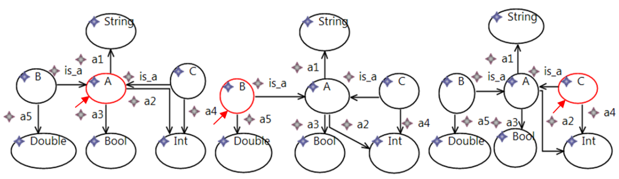

Fig. 3. Projection of a *ldg* to a set of *lts*.

Step4. *Data conservation*

For data migration, we can assert that the obtained *ldg* and *lts* are conform to Definition 3. The functions of the APIs can be used for data migration.

V. Case Studies

Our approach has been deployed in the space engineering domain. We have studied several case studies with complex data models. In this section, we review the case of the Microscope data model. The Microscope space mission aims at testing the universality of free fall, for the first time in space [15]. In the following, we consider an extract of the data model used to parameterize the telemetry processing and especially to combine two telemetries.

Step1. *Interpretation*

As shown in Figure 4, it was decided to refactor the data model by replacing two attributes by two composition relationships towards a new class, called `AbstractData`.

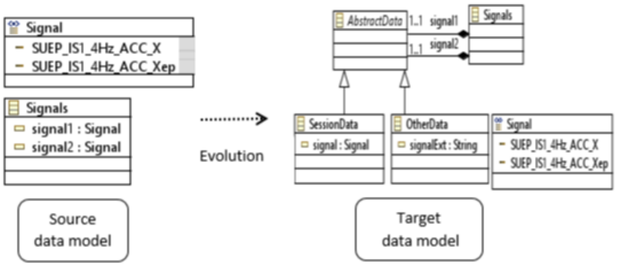

Fig. 4. An extract of a data model evolution in Microscope.

The original attributes `signal1` and `signal2` are factorized into a class `SessionData`, inheriting from `AbstractData` and owning a `signal` attribute of type `Signal`. This way, end-users can combine two telemetries with a known signal or not. Thus, we can identify the following evolutions:

- a new abstract class named `AbstractData` is added;
- two new classes named `SessionData`, `OtherData` inheriting from `AbstractData` are added;
- a new attribute named `signal` of type `Signal` is added to the class `SessionData`;
- a new attribute named `signalExt` of type `String` is added to the class `OtherData`;
- the types of `signal1` and `signal2` are changed from `Signal` to `AbstractData`.

As explained previously, the source and target data models are transformed into two LDG. In the source LDG, the class `Signals` become one concept. In the target LDG, three new concepts appear : `AbstractData`, `SessionData` and `OtherData`.

Step2. *Projection*

As shown in Figure 5, we project the source and target LDG to one source *lts* lts_s and one target *lts* lts_t.

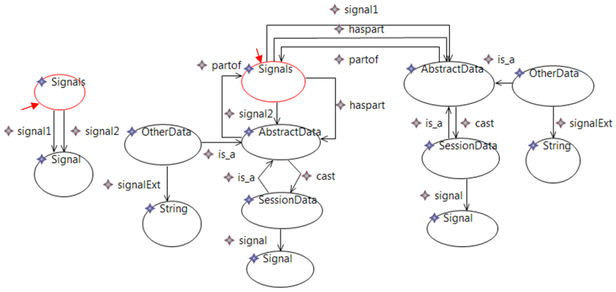

Fig. 5. An extract of the projection of the *ldg* to a set of *lts* in Microscope.

As the `Signals` concept is the only concept that exists in both sides, the initial state of each *lts* is `Signals`.

Step 3. *LTS comparison.*

As shown in Figure 5, from the initial state `Signals`, all the source transitions are feasible if we cast the target `signal1` and `signal2` to the concept `SessionData`:

$$signal1_{source} = ((SessionData)signal1_{target}).signal$$

$$signal2_{source} = ((SessionData)signal2_{target}).signal$$

Thus, lts_t simulates lts_s since the following weak simulation binary relationship $R = \{< Signals, Signals >\}$ exists. One may notice that the opposite is false because of the transition `signalExt`.

Step4. *Data conservation*

During the migration, the values of the old instances of the class `Signals` are preserved, through the creation of two new instances of the class `SessionData`, initialized with these values.

VI. Conclusion and Future work

Tool support: The use of the CADP[2] toolbox helps a lot. Our investigations proved that we have been able to get the diagnosis of the comparison results.

Using template transformations, we were able to transform the Microscope data models to an LDG, then to obtain an internal representation of labelled transition system in `AUT` (automaton) format. Finally, the simulation between the two automatons is checked, using observational equivalence relationship, thanks to the CADP `BISIMULATOR` module.

Results and Discussion: In this paper, we presented a semantic observational approach for treating data models evolution. The main interest of the proposed approach is the transposition of the information accessibility in a data model at a logical interface level into a path problem in a labelled directed graph. The approach proved capable to capture all evolutions of a data model into a single logical operator instead of a no-exhaustive list of evolution operators.

Finally, the proposed approach is generic, it is not defined for a single specific data modelling language. It applies to any data modelling language provided that an interpretation of each data model by a *ldg* (from which a set of *lts* is produced) is defined.

Concluding remarks: We believe that addressing the problem of model evolution based on model behavior is promising. Interfaces defined on data models are used to define model evolution correctness using observational semantics. They are also used to prove the existence or the non-existence of composite operators having the property to preserve information contained in original instances.

Relying on labelled transition systems has three potential advantages. First, the overall system is often easier to understand due to the formal and precise nature of the representation scheme. Secondly, the behavior of the system can be analyzed using labelled transition systems theory and associated techniques, which includes tools for analysis. Finally, techniques developed for the comparison of parallel programs can also be applied.

Future work: As a perspective of this approach, we expect to realize a comparative study between the proposed approach and the previous one by comparing traces found by a graph comparison algorithm to structural differences found previously in [12]. We also intend to extend our work to address the evolution of models in presence of cardinalities. Finally, integrating domain knowledge through the introduction of a domain ontology helps in identifying semantic equivalence at concepts levels and thus address heterogeneous models evolution.

Acknowledgment

Authors would like to express their gratitude to Dr. Raquel Araujo OLIVEIRA for her comments and her constructive suggestions.

References

[1] A. Ferdjoukh, A. Baert, E. Bourreau, A. Chateau, R. Coletta, and C. Nebut, "Instantiation of meta-models constrained with ocl: a csp approach," in *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD).* IEEE, 2015, pp. 213–222.

[2] J. E. Rivera, F. Durán, and A. Vallecillo, "Formal specification and analysis of domain specific models using maude," *Simulation*, vol. 85, no. 11-12, pp. 778–792, 2009.

[3] A. Narayanan and G. Karsai, "Using semantic anchoring to verify behavior preservation in graph transformations," *Electronic Communications of the EASST*, vol. 4, 2006.

[4] R. Oliveira, S. Dupuy-Chessa, and G. Calvary, "Equivalence checking for comparing user interfaces," in *Proceedings of the 7th ACM SIGCHI Symposium on Engineering interactive Computing Systems.* ACM, 2015, pp. 266–275.

[5] R. Oliveira and J. Dingel, "Supporting model refinement with equivalence checking in the context of model-driven engineering with uml-rt." in *MODELS (Satellite Events)*, 2017, pp. 307–314.

[6] A. Chebieb and Y. Ait-Ameur, "A formal model for plastic human computer interfaces," *Frontiers of Computer Science*, vol. 12, no. 2, pp. 351–375, 2018.

[7] J. Euzenat, P. Shvaiko *et al.*, *Ontology matching.* Springer, 2007, vol. 18.

[8] R. Khatchadourian and H. Masuhara, "Defaultification refactoring: A tool for automatically converting java methods to default," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 2017, pp. 984–989.

[9] R. G. Kula, A. Ouni, D. M. German, and K. Inoue, "An empirical study on the impact of refactoring activities on evolving client-used apis," *Information and Software Technology*, vol. 93, pp. 186–199, 2018.

[10] D. Dig and R. Johnson, "How do apis evolve? a story of refactoring," *Journal of software maintenance and evolution: Research and Practice*, vol. 18, no. 2, pp. 83–107, 2006.

[11] J. Henkel and A. Diwan, "Catchup! capturing and replaying refactorings to support api evolution," in *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* IEEE, 2005, pp. 274–283.

[12] L. Ait-Oubelli, Y. Aït-Ameur, J. Bedouet, R. Kervarc, B. Chausserie-Laprée, and B. Larzul, "A scalable model based approach for data model evolution: Application to space missions data models," *Computer Languages, Systems & Structures*, vol. 54, pp. 358–385, 2018.

[13] R. Milner, *Communication and concurrency.* Prentice hall New York etc., 1989, vol. 84.

[14] V. Hémery, B. Larzul, and B. Chausserie-Laprée, "Best-ng: a new modeler for describing the satellite's database," in *2018 SpaceOps Conference*, 2018, p. 2305.

[15] Q. Baghi, G. Métris, J. Bergé, B. Christophe, P. Touboul, and M. Rodrigues, "Gaussian regression and power spectral density estimation with missing data: The microscope space mission as a case study," *Physical Review D*, vol. 93, no. 12, p. 122007, 2016.

[2] https://cadp.inria.fr/