Digital Continuity Based on Reinforcement Learning Model Transformations

Quentin Brilhault(✉), Esma Yahia, and Lionel Roucoules

Arts et Métiers Institute of Technology, HESAM Université, LISPEN, 13617 Aix-en-Provence, France
{quentin.brilhault,esma.yahia,lionel.roucoules}@ensam.eu

Abstract. With the importance gained by *Service-Oriented Architectures* (SOA) to simplify and decompose complex enterprise information system into autonomous, modular, reusable and, flexible model, the need to make models interoperable to ensure digital continuity has increased. However, the structural, syntactic, and semantic heterogeneity of metamodel, drastically complicates the interconnection of models and thus the digital continuity. A key element of *Model-Driven Architecture* (MDA), model transformations could be one of the solutions to promote model interoperability. They allow the description of the transformation rules that link the different metamodels concepts. However, significant efforts are needed to describe and maintain model transformations in an ever-changing digital environment. One of the key challenges of the MDA approach is the automation of model transformations. Recent work exploiting *machine learning* techniques to infer model transformations has shown very promising results. However, learning algorithms, involve too much training data and require a large and varied dataset. In our work, we want to experiment the *reinforcement learning* techniques to infer transformation rules and to counter the need to provide a considerable volume of data for machine learning.

Keywords: Digital continuity · Model transformations · Reinforcement learning · Q-Learning

1 Introduction

In a rapidly changing industrial context, from an economic, technical, and organizational point of view, ensuring and maintaining the digital continuity of information is a major challenge. In the digital factory, digital continuity ensures that all information is available and distributed to the right people, at the right time without restriction due to the heterogeneity of information systems. It allows the linking of information to guarantee its completeness and consistency throughout the design and production cycle.

Model transformations could provide a concrete solution to meet model interoperability requirements to promote digital continuity. Model transformation techniques are the cornerstone of *Model-Driven-Architecture* [1] (MDA). The MDA framework defines how a model, conforming to its metamodel, can be transformed into a model conforming

S. Gerbino et al. (Eds.): JCM 2022, LNME, pp. 442–453, 2023.
https://doi.org/10.1007/978-3-031-15928-2_39

to another metamodel [2]. The transformation is defined by a set of structural (relation between concepts having different structures) and semantic (relation between concepts having the same meaning) relations that connect the source metamodel concepts to the target metamodel concepts. These relations are called transformation rules.

The first step when it comes to establish a transformation between two metamodels is to identify the correspondence relations between their concepts. However, the proliferation of heterogeneous modeling languages complicates their ability to interconnect properly. In fact, the same information is modelled by different structures and terminologies according to the metamodels which drastically hinder the identification of the correspondence relations between the concepts of the source and target metamodels [3].

Defining and maintaining model transformations is a constraint that requires time and knowledge. It is legitimate to wonder how to automate, partially or totally, the transformation rules identification to promote model interoperability. The first approaches using *machine learning* techniques have brought very promising results but require a large set of training data. In this article, experiments have shown the possibility of using *reinforcement learning* techniques [4], and more specifically *Q-Learning*, for learning model transformations. The advantage of this technique is that it only requires a small amount of training data. Learning is achieved through the interactions between an intelligent agent and its environment. The agent learns from its actions by following the "test and learn" principle. Each action taken is rewarded with a higher (beneficial action) or lower (bad action) score depending on the result obtained. The objective of the intelligent agent is to learn the optimal policy π^* that maximizes the expectation of rewards following a feasible action sequence.

The aim of the paper is to infer, using *Q-Learning* techniques, the structural and semantic relations that connect the concepts of a source metamodel to the target metamodel ones by automatically generating reusable transformation rules.

Recent approaches have exploited reinforcement learning methods for automation of model repairing [5] and "in-place" model transformations [6]. We follow them, and we propose a learning approach by reinforcement of model transformations.

2 Related Work

The transformation model specifies the transformation rules that allow, from a source model, to lead to a target model. They are edited at metamodel level but act on model elements. The QVT[1] standard, defined by the OMG[2], provides an architecture and dedicated languages facilitating the descriptive transformation of a source model into a target model. For instance, the declarative transformation language ATL [7] is inspired by this standard. Transformation is user-defined, i.e., written by developers.

Initiated by Varró [8], the *Model Transformation by Example* (MTBE) approach aims to semi-automatically derive a set of transformation rules from a set of transformation pairs given as examples (source models and associated target models). The derivation of transformation rules is guides by a pre-alignment manually defined between source

[1] About the MOF Query/View/Transformation Specification Version 1.3 (omg.org).
[2] OMG | Object Management Group.

and target model elements. This mapping between models specifies how an element of the source model is transformed into an element of the target model. The approach [8] is then implemented by applying the principles of *inductive logic programming* to automate the generation of model transformations by example [9]. Wimmer et al. [10] propose a similar approach to generate the transformation rules at the metamodel level in ATL language based on the inter-model mapping specified manually.

Dolques et al. [11] propose a machine learning approach based on the principles of *Relational Analysis of Concepts* (RCA) which allows the classification of the objects according to their properties. RCA takes into consideration the objects described according to their relations with other objects. Therefore, inferring the transformation rules is equivalent to finding the common characteristics shared by the elements of the source and target models. Their approach is initialized by manual predefined inter-model mapping. Saada et al. [12] propose to enrich the approach by automatically generating operational transformation rules using a transformation language executable by a rules engine.

The work of Baki and Sahraoui [13] shows that the search space is too large to determine an optimal solution when it comes to inferring complex transformation rules. To reduce search space, the previously established inter-model relationships are then grouped by categories into different pools. For each identified categories, the use of *genetic algorithms* attempts to associate each inter-model relationship with a transformation rule that best transforms the elements of the source model into elements of the target model.

Finally, the work initiated by Burgueño et al. [14] takes advantages the field of *machine learning* by applying supervised learning methods for pattern recognition. They use the LSTM method (*Long Short-Term Memory*), an extension of neural networks, which has the property of long-term memory that can remember previously established mapping to build more complex mapping.

3 Discussion

The analysis of the state of the art presented previously made it possible to identify certain limits which inhibit the possibilities of model transformations automation:

Descriptive Transformation Rules. Writing descriptive, “ad-hoc”, hand-written transformation rules in a transformation language is a very time-consuming activity that requires both a good understanding of the structure and semantics of the source and target domain, as well as good knowledge of transformation languages to formalize the solution. Moreover, considerable efforts are required to describe and maintain the model transformations in a context where the digital environment is highly heterogeneous and in continuous change.

Manual Pre-alignment Between Source and Target Models. Most of the research applying the MTBE approach are essentially based on manually pre-established correspondence relations between the elements of the source and target models. This pre-alignment is not always easy to provide, it often requires a business heuristic. It is clear that the impact of the human is omnipresent in the process of describing model transformations. The lack of automation is a significant gap that falls to the development of automatic interoperability between models.

An Application of Neural Networks for Learning Model Transformations. Burgueño et al. [14] showed that learning model transformations requires a large amount of source and target models during the training phase with an *artificial neural network* (ANN). This constraint is amplified when it comes to learning complex n-m relationships subject to transformation conditions. The diversity of the training dataset is also a limiting factor. The ANN can predict only the scenarios that it has been previously learned. Finally, the duration of the training depends on the number of models in the dataset, and on the number of elements per model.

4 Overview of the Proposed Approach

The proposed approach follows three main steps: (1) the transformation of a class diagram into a graph (see 4.1), (2) the extraction of its structure and its information (see 4.2), and (3) the learning phase (see 4.3). The Fig. 1 show the whole learning process.

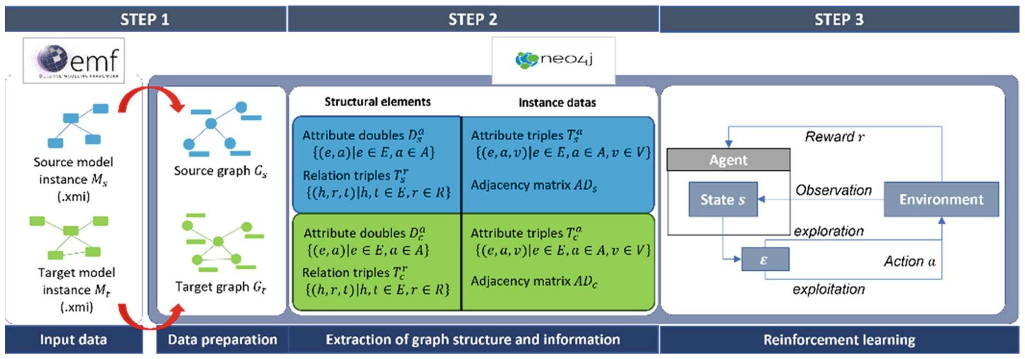
STEP 1
STEP 2
STEP 3
emf
neo4j
Source model instance M_s (.xmi)
Target model instance M_t (.xmi)
Source graph G_s
Target graph G_t
Structural elements
Instance datas
Attribute doubles D_s^a $\{(e,a)|e \in E, a \in A\}$
Relation triples T_s^r $\{(h,r,t)|h,t \in E, r \in R\}$
Attribute triples T_s^a $\{(e,a,v)|e \in E, a \in A, v \in V\}$
Adjacency matrix AD_s
Attribute doubles D_c^a $\{(e,a)|e \in E, a \in A\}$
Relation triples T_c^r $\{(h,r,t)|h,t \in E, r \in R\}$
Attribute triples T_c^a $\{(e,a,v)|e \in E, a \in A, v \in V\}$
Adjacency matrix AD_c
Reward r
Agent
State s
Observation
Environment
exploration
ε
Action a
exploitation
Input data
Data preparation
Extraction of graph structure and information
Reinforcement learning

Fig. 1. Overview of the training phase of transformation models

The output elements obtained after the learning phase are the trained Q-tables in attributes and in relations expressed by the flattened entities.

The prediction phase of a target model from a new source model (conform to source metamodel learned) will carried out by reusing the generated Q-tables.

4.1 Input Data: Dataset Presentation

Following the principles of the MTBE approach, a couple of source and target models in form of instance diagrams are provide to the system, unlike previous works presented in the state of the art which seek to infer the transformation rules from models only. It is assumed that, sometimes, it is easier to identify the correspondences between attributes (and by extension between classes) based on the similarity of their value (i.e., data of the instance of the model), rather than on the names of the concepts themselves (names of classes, relations, and attributes) [15].

By applying the principles of reverse engineering, the authors assume finding the structure and the terminology of the metamodel concepts linked to the instance diagrams of the source and target models. This will allow inferring the transformation rules at the metamodel level.

To illustrate the proposed approach, the commonly used example *Family2Person* transformation model present on the Zoo ATL[3], is used. The Fig. 2 presents the source *Family* the target *Person* metamodel used to validate the approach.

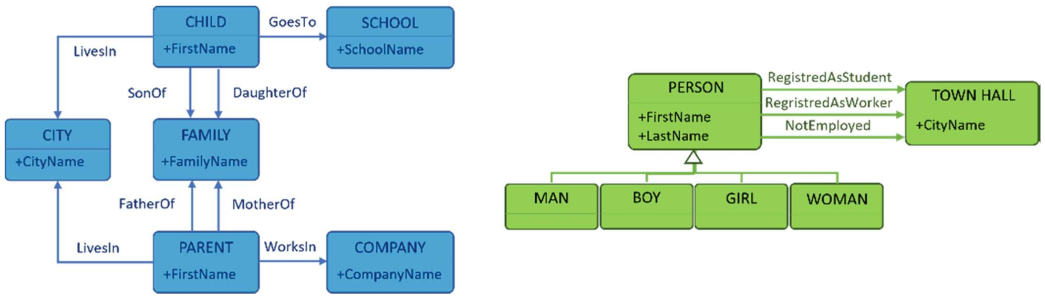

Fig. 2. Family source metamodel (blue) and Person target metamodel (green)

These metamodels are composed of classes ([CHILD], [TOWN HALL] …) which are themselves composed of attributes (*FirstName*, *CityName*…). Classes are linked together by relationships (*GoesTo*, *RegistredAsStudent*…).

The Table 1 presents the transformation rules that link the *Family* metamodel to the *Person* metamodel. The approach must be able to derive the characteristics of source and target metamodels to infer theses transformation rules.

Table 1. Transformation rules for Family2Person_extanded

Rules	Description
Child2Girl (n–1, σ_r)	If [CHILD] –[*DaughterOf*] → [FAMILY] then class [GIRL] is created with *FirstName.GIRL* = *FirstName.CHILD* and *LastName.GIRL* = *FamilyName.FAMILY*
Child2Boy (n–1, σ_r)	If [CHILD] –[*SonOf*] → [FAMILY] then class [BOY] is created with *FirstName.BOY* = *FirstName.CHILD* and *LastName.BOY* = *FamilyName.FAMILY*
Parent2Woman (n–1, σ_r)	If [PARENT] –[*MotherOf*] → [FAMILY] then class [WOMAN] is created with *FirstName.WOMAN* = *FirstName.PARENT* and *LastName.WOMAN* = *FamilyName.FAMILY*
Parent2Man (n–1, σ_r)	If [PARENT] –[*FatherOf*] → [FAMILY] then class [MAN] is created with *FirstName.MAN* = *FirstName.PARENT* and *LastName.MAN* = *FamilyName.FAMILY*
City2TownHall (1–1)	If [CITY] then class [TOWN HALL] is created with *CityName.TOWNHALL* = *CityName.CITY*
Child2Student (n–m, σ_r)	If [CHILD] –[*GoesTo*] → [SCHOOL] then relation *RegistredAsSudent* is created between class [GIRL] or [BOY] to class [TOWN HALL]
Parent2Worker (n–m, σ_r)	If [PARENT] –[*WorksIn*] → [COMPANY] then relation *RegistredAsWorker* is created between class [WOMAN] or [MAN] to class [TOWN HALL]
Parent2Unemployed (n–m, σ_r)	If [PARENT] –NOT[*WorksIn*] → [COMPANY] then relation *NotEmployed* is created between class [WOMAN] or [MAN] to class [TOWN HALL]

Note that some rules are subject to transformation conditions that can guide the transformation of a concept. This would help to hypothesize that the transformation is guided by specific concept characteristics. For instance, the transformation of the class

[3] ATL Transformations | The Eclipse Foundation.

[CHILD] to the class [GIRL] and [BOY] depends on the association (*DaughterOf* or *SonOf*) that the class [CHILD] has with the class [FAMILY]. Thus, the transformation of the class [CHILD] is conditioned by one of its relations. A condition on a relation is then defined by σ_r

The Fig. 3 presents the instance diagrams of the source and target models conforming to the metamodels shown in Fig. 2.

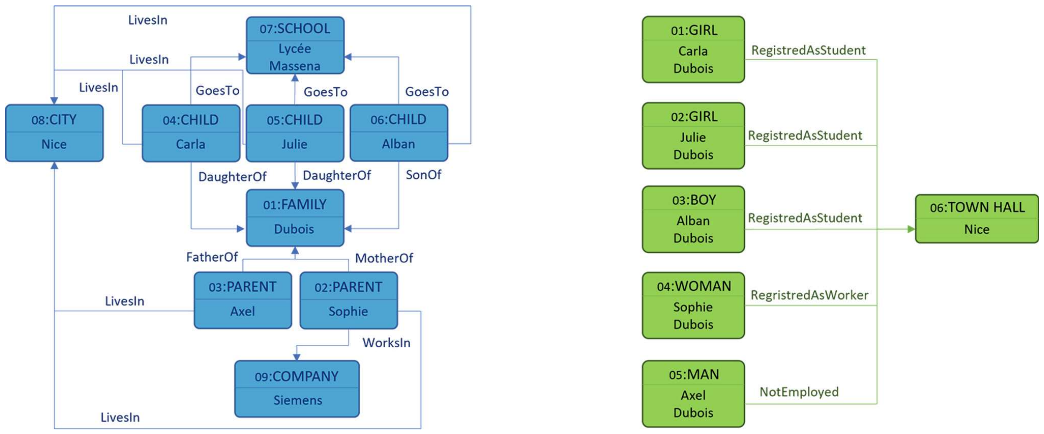

Fig. 3. Instance of the Family (in blue) and Person (in green) models

4.2 Extraction of Graph Structure and Graph Information

Data Preparation. Neo4j[4] graph database is used for the proposed approach. It provides an ergonomic environment for visualizing relational data. Besides, Neo4j provides the *Cypher*[5] query language for navigating through graphs that eases the manipulation of the structure and data. Then, Neo4j provides to the developer the latest algorithms published in the literature, especially the GraphSAGE[6] algorithm for calculating node embeddings. We plan to exploit these valuable algorithms in our future work.

Regarding the import of source and target model instance diagrams in graph format in Neo4j, the *UMLtoGraphDB* [16] is used to transform a conceptual model expressed in UML language (*Unified Modeling Language*) into a graph representation.

A data graph can be defined by a set of nodes (or entities), and by a set of oriented or undirected relations (links between nodes). Within the studied framework, the relations between the entities will be oriented. Each node can contain one or more attributes that are typed according to the data format (numeric, character, date, etc.). A representation G of a data graph is defined by $G = (E, R, A, V)$, where E be the set of entities (defined by their type, i.e., their label) belonging to G, R the set of relations (defined by their type, i.e., their label) linking the entities together, A the set of attributes that make up the entities and V the values of the attributes.

[4] Graph Data Platform | Graph Database Management System | Neo4j.
[5] Cypher Query Language - Developer Guides (neo4j.com).
[6] GraphSAGE - Neo4j Graph Data Science.

Extraction of Structure and Data from Source and Target Graphs.
Once the instance diagrams were imported and then converted to a graphs format in Neo4j, the python library[7], as well as the cypher query language is used to extract the structure and information contained in the source and target graphs. Capturing the structure of a model is equivalent to extracting the metamodel to which it conforms. This will facilitate the definition of transformation relations between the concept of the source and target metamodels. In this sense, it is important to extract from the source G_s and target G_t graphs the following elements:

Attribute Doubles. D_s^a and D_t^a such that $D^a = \{(e, a)|e \in E, a \in A\}$. . The attribute doubles capture the internal structure of an entity, i.e., the attributes that make up the entity (the entity with the label [GIRL] has the attribute doubles $<$ GIRL, *FirstName* $>$ and $<$ GIRL, *LastName* $>$ according to the target metamodel).

Relation Triples. T_s^r and T_t^r such that $T^r = \{(h, r, t)|h, t \in E, r \in R\}$ with r a relationship oriented from the entity h to the entity t. . Relation triples capture the adjacent structure of an entity, i.e., neighboring entities (an entity with label CHILD has relation triples $<$ CHILD, *GoesTo*, SCHOOL $>$, $<$ CHILD, *LivesIn*, CITY $>$ and $<$ CHILD, *DaughterOf*, FAMILY $>$ according to the source metamodel). The relation triples are determined from the adjacency matrices AD_s and AD_t which expresses the structure, i.e., the oriented relations between the entities in the source and target graphs.

Flattened Entities. To express the subtleties of the structure of the graphs, we have chosen to represent each entity by its flattened shape. A flattened representation consists of capturing the internal (entity attributes) and external (inherited, i.e., relationships with adjacent entities) characteristics of an entity. We have therefore defined the flattened shape of an entity as follows $F_e = (D_e^a, T_e^r)$ where D_e^a corresponds to the attribute doubles of the entity e and T_e^r corresponds to the relation triples of the entity e. Thanks to flattened representations, the context of each entity is fully expressed, and the structure of the graph is fully captured in a flattened form. The Fig. 4 presents the flattened entities obtained for the metamodel *Family* and the metamodel *Person* based on the instances of the source and target models.

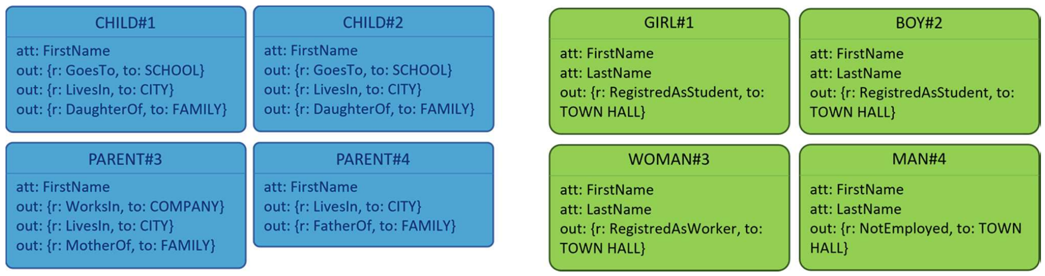
CHILD#1
att: FirstName
out: {r: GoesTo, to: SCHOOL}
out: {r: LivesIn, to: CITY}
out: {r: DaughterOf, to: FAMILY}
CHILD#2
att: FirstName
out: {r: GoesTo, to: SCHOOL}
out: {r: LivesIn, to: CITY}
out: {r: DaughterOf, to: FAMILY}
PARENT#3
att: FirstName
out: {r: WorksIn, to: COMPANY}
out: {r: LivesIn, to: CITY}
out: {r: MotherOf, to: FAMILY}
PARENT#4
att: FirstName
out: {r: LivesIn, to: CITY}
out: {r: FatherOf, to: FAMILY}
GIRL#1
att: FirstName
att: LastName
out: {r: RegistredAsStudent, to: TOWN HALL}
BOY#2
att: FirstName
att: LastName
out: {r: RegistredAsStudent, to: TOWN HALL}
WOMAN#3
att: FirstName
att: LastName
out: {r: RegistredAsWorker, to: TOWN HALL}
MAN#4
att: FirstName
att: LastName
out: {r: NotEmployed, to: TOWN HALL}

Fig. 4. Flattened representation of source (in blue) and target (in green) graph entities

[7] Neo4j Python Driver 4.4—Neo4j Python Driver 4.4.

Attribute Triples. ${}_{id}T_s^a$ and ${}_{id}T_t^a$ such that ${}_{id}T^a = \{id(e, a, v)|e \in E, a \in A, v \in V\}$ where *id* number is a unique identifier associated with each entity (the entity with *id* number 5 has attribute triple < CHILD, *FirstName*, Julie >). In the case where an entity e has several attributes a, the *id* number makes it possible to associate all the values v with a single entity.

4.3 Reinforcement Learning of Transformation Rules

Among the different reinforcement learning algorithms, the *Q-Learning* is chosen. The experiences and knowledge accumulated by the agent are stored in a table called Q-table which is structured by a set of lines which characterizes all the possible states, and by a set of columns which represents all the possible actions. Thus, it allows for each state s, to determine the expectation of the rewards for each action a.

The Fig. 5 presents the main characteristic elements of reinforcement learning, namely, states, actions, the environment, and the reward function.

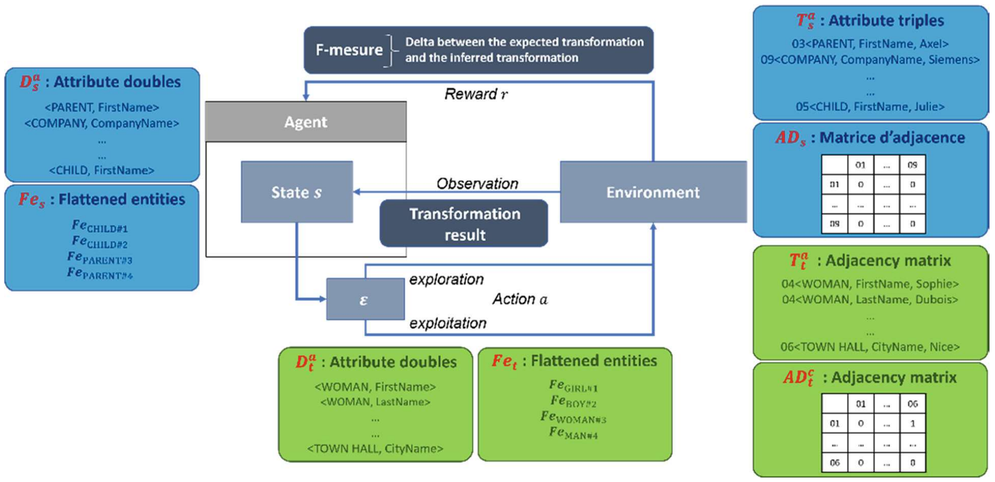
F-mesure
Delta between the expected transformation and the inferred transformation
Reward r
Agent
State s
Observation
Transformation result
Environment
ε
exploration
Action a
exploitation
D_s^a : Attribute doubles
<PARENT, FirstName>
<COMPANY, CompanyName>
<CHILD, FirstName>
Fe_s : Flattened entities
$Fe_{CHILD\#1}$
$Fe_{CHILD\#2}$
$Fe_{PARENT\#3}$
$Fe_{PARENT\#4}$
T_s^a : Attribute triples
03<PARENT, FirstName, Axel>
09<COMPANY, CompanyName, Siemens>
05<CHILD, FirstName, Julie>
AD_s : Matrice d'adjacence
T_t^a : Adjacency matrix
04<WOMAN, FirstName, Sophie>
04<WOMAN, LastName, Dubois>
06<TOWN HALL, CityName, Nice>
AD_t^c : Adjacency matrix
D_t^a : Attribute doubles
<WOMAN, FirstName>
<WOMAN, LastName>
<TOWN HALL, CityName>
Fe_t : Flattened entities
$Fe_{GIRL\#1}$
$Fe_{BOY\#2}$
$Fe_{WOMAN\#3}$
$Fe_{MAN\#4}$

Fig. 5. Reinforcement learning applied to learning transformation rules

The Environment. It includes the instance of the source model and target model in the form of attribute triples T_s^a and T_t^a as well as their adjacency matrix AD_s and AD_t.

The States. They are defined as being the source patterns in attribute in the form of doublets D_s^a, and in the form of flattened entities Fe_s to express relationships between entities.

The Actions. Performing an action result in the selection of a target pattern. In other words, in the case of a model transformation, carrying out an action corresponds to choosing the target pattern which corresponds to a source pattern. The target attribute patterns are defined by doublets D_t^a, where the flattened entities Fe_t express relationships between entities. Thus, for a source attribute pattern there exists, potentially, a target attribute pattern to match with (similarly for source and target flattened entities). Choosing an action follows the ε-greedy policy, which means that the intelligent agent can either explore its environment by executing a random action or perform an action by exploiting the knowledge acquired and stored in the Q-table. The more value of ε decreases, the more the agent will be incited to exploit his knowledge.

The Reward. The reward r depends on the result obtained after applying a certain action $d_t \in D_t^a$ and $fe_t \in Fe_t$ for a state $d_s \in D_s^a$ and $fe_s \in Fe_s$. In our case, it is a question of evaluating whether the result of the transformation of the source pattern into a target pattern is correct. Therefore, we have defined the calculation of the reward as being the calculation of the *F-measure* which considers the *precision* P, i.e., the ratio between the number of expected items produced and the total number of items produced, and the *recall* R, i.e., the ratio between the number of expected produced items and the total number of expected items, such as:

$$F = 2\frac{P * R}{P + R} \tag{1}$$

To evaluate if the result of the transformation is conforming to the expected result, we apply the transformation to the elements of the source instance diagram corresponding to the source pattern. The result of the transformation will be, then, compared to the elements of the target instance diagram corresponding to the chosen target pattern.

Optimization and Learning Process. When learning starts, all the state-action pairs of the Q-tables are initialized to zero. The value for a state-action couple, called Q-value, is updated according to the interaction conducted by the agent in its environment and is calculated by the Bellman equation.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r + \gamma max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)) \tag{2}$$

$\gamma max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1})$ allows inferring the Q-value for a state s_t and action a_t based on the best action a_{t+1} of the next state s_{t+1}. In the case of model transformation, $\gamma max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1})$ is used to measure the interdependencies of the actions conducted before. In other words, this coefficient makes it possible to assess whether the action that has just been conducted is consistent with past actions.

5 Experiment

To measure the performance of the approach, the evaluation is based on two case studies presented in Table 2.

Table 2. Training dataset

Transformations	MM_s	MM_t	M_s	M_t	iM_s	iM_t
Family2Person	- (3)(4)(3)	- (2)(0)(4)	-	-	(1) (6)(5)(6)	(1) (5)(0)(10)
Family2Person_extanded	- (6)(7)(6)	- (5)(3)(9)	-	-	(1) (9)(14)(9)	(1) (6)(5)(11)

Table read instructions: for the family2Person transformation, the source metamodel. MM_s is composed of 3 classes, 4 relationships and 3 attributes (second line); the symbol '-' (first line) means that the source metamodel is not given as input to the algorithm. iM annotation corresponds to instance models which are the only data given as input to the system.

Table 3. Results of experiments

Approach	*Transformations*	*Input data*	*Training performances*				*Predict. Perf.*
			Training time (s)	*P*	*R*	*F*	*F*
QLearning	*Family2Person*	iM_s iM_t	11.40	1.0	1.0	1.0	1.0
QLearning	*Family2Person_extended*	iM_s iM_t	32.2	0.74	1.0	0.85	1.0

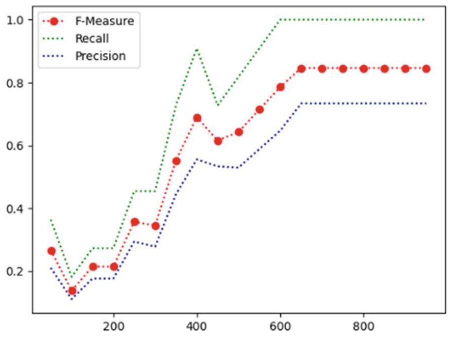

Fig. 6. Learning performance for Family2Person_extended transformation

The Fig. 6 shows the evolution of *F-measure*, *recall* and *precision* over episodes during the training phase of *Family2Person_extended* transformation. As we can see, the performance metrics increase with the learning and peak at 1 for the recall, 0.74 for the precision and 0.85 for the F-measure (see Table 3). This increase is due to the decrease of Epsilon ε which means that the intelligent agent exploits more the acquired past knowledge during the learning phase.

A *recall* of 1 means that all the expected elements (classes, relationships, and attributes) were perfectly inferred without loss. However, the *precision* is not perfect. This is explained by the fact that the [TOWN HALL] class appears in each target flattened entities (see Fig. 4). Therefore, each time a source flattened entity is associated with a target flattened entity, a [TOWN HALL] class is created.

Regarding the training duration, it can be significantly reduced by (1) adjusting the number of episodes necessary during the learning phase (in our case we did it over 1000 episodes); (2) improving the source code of the reward function which checks the overall consistency of the actions taken by reducing the calculation time.

Finally, concerning the prediction of a target model, from a completely different source instance diagram (conforming to the learned source metamodel), the reuse of Q-tables (in attributes and in relations thanks to flattened entities) makes it possible to infer the entire target instance diagram from a new source instance diagram. This means that all the transformation rules (see Table 1) between the source metamodel and the target metamodel have been correctly learned.

6 Conclusion

In this article, a concrete solution to model interoperability problems was provided to guarantee the digital continuity. A model transformation learning approach was proposed to infer structural and semantic relationships between models. More particularly the reinforcement learning techniques using the *Q-learning* was exploited to infer the transformation rules between two metamodels.

The results obtained from the two case studies are very promising since the learning is done within a reasonable time with only one source instance diagram and one target instance diagram. Besides, the learning phase makes it possible to extract all the rules which link two metamodels.

In future work, the difficulty of the used datasets will be increased by adding more restrictive transformation conditions to learn (condition in attribute, and modification of the attribute value for instance). Another objective is to set a complete benchmark allowing the comparison of the proposed approach performance with the latest approaches using *machine learning* techniques. Finally, the association of *Deep Learning* methods with *reinforcement learning* will be exploited to ameliorate the expected performance.

Acknowledgments. The authors would like to thank the Centre National d'Etudes Spatiales (CNES) and Thales Alenia Space for their financial, material and above all human support. A proof of concept within Thales Alenia Space is already under study to demonstrate the performance of the approach in a real industrial context.

References

1. Soley, R.: Model driven architecture. Object Manag. Group, 12 (2000)
2. Warmer, J.B., Kleppe, A.G.: The object constraint language: Getting your models ready for MDA. Addison-Wesley Professional (2003)
3. Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schoenboeck, J., Schwinger, W.: From the heterogeneity jungle to systematic benchmarking. In: Dingel, J., Solberg, A. (eds.) MODELS 2010. LNCS, vol. 6627, pp. 150–164. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21210-9_15
4. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction.pdf (2016)
5. Barriga, A., Rutle, A., Heldal, R.: Personalized and automatic model repairing using reinforcement learning. In: Proceedings of the 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Munich, Germany, pp. 175–181 (2019). https://doi.org/10.1109/MODELS-C.2019.00030

6. Eisenberg, M., Pichler, H.-P., Garmendia, A., Wimmer, M.: Towards reinforcement learning for in-place model transformations. In: Proceedings of the 2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS), Fukuoka, Japan, pp. 82–88 (2021). https://doi.org/10.1109/MODELS50736.2021.00017
7. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Bruel, J.-M. (ed.) MODELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006). https://doi.org/10.1007/11663430_14
8. Varró, D.: Model transformation by example. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MODELS 2006. LNCS, vol. 4199, pp. 410–424. Springer, Heidelberg (2006). https://doi.org/10.1007/11880240_29
9. Balogh, Z., Varró, D.: Model transformation by example using inductive logic programming. Softw. Syst. Model. **8**, 347–364 (2009)
10. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards model transformation generation by-example. In: Proceedings of the 2007 40th Annual Hawaii International Conference on System Sciences (HICSS 2007), Waikoloa, HI, pp. 285b–285b (2007). https://doi.org/10.1109/HICSS.2007.572
11. Dolques, X., Huchard, M., Nebut, C., Reitz, P.: Learning transformation rules from transformation examples: An approach based on relational concept analysis. In: Proceedings of the 14th IEEE International Enterprise Distributed Object Computing Conference Workshops, p. 7 (2010)
12. Saada, H., Dolques, X., Huchard, M., Nebut, C., Sahraoui, H.: Generation of operational transformation rules from examples of model transformations. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) MODELS 2012. LNCS, vol. 7590, pp. 546–561. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33666-9_35
13. Baki, I., Sahraoui, H.: Multi-step learning and adaptive search for learning complex model transformations from examples. ACM Trans. Softw. Eng. Methodol. **25**(3), 1–37 (2016). https://doi.org/10.1145/2904904
14. Burgueño, L., Cabot, J., Li, S., Gérard, S.: A generic LSTM neural network architecture to infer heterogeneous model transformations. Softw. Syst. Model. **21**, 139–156 (2021). https://doi.org/10.1007/s10270-021-00893-y
15. Ali, A., Nordin, A., Alzeber, M., Zaid, A.: A survey of schema matching research using database schemas and instances. Int. J. Adv. Comput. Sci. Appl. **8**(10) (2017).https://doi.org/10.14569/IJACSA.2017.081014
16. Daniel, G., Sunyé, G., Cabot, J.: UMLtoGraphDB: Mapping conceptual schemas to graph databases. In: Comyn-Wattiau, I., Tanaka, K., Song, I.-Y., Yamamoto, S., Saeki, M. (eds.) ER 2016. LNCS, vol. 9974, pp. 430–444. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46397-1_33