Journal of Theoretical and Applied Information Technology
30th November 2013. Vol. 57 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

INTRODUCING THE OPEN SOURCE METAMODEL CONCEPT

AHMED MOHAMMED ELSAWI[1], SHAMSUL SAHIBULDIN[2], ABDELHAMID ABDELHADI[3]
[1,2,3] Software Engineering Department, Faculty of Computing, UniversitiTeknologi Malaysia, Malaysia
E-mail: [1]elsawi@gmail.com, [2]shamsul@utm.my, [3]abhamidhn@uofk.edu

ABSTRACT

The Model Driven Architecture (MDA) is a model centric software engineering methodology that aiming to enhance software productivity, reusability, maintainability and quality by focusing on models and metamodels in place of conventional code. By adopting the separation of concern these models defined in different levels of abstraction where each model syntactically conforms to a particular metamodel. Under the MDA context this work presents a novel approach for representing models and metamodels. Benefiting from the knowledge representation capability and the open structure of the Entity Attribute Value (EAV) model, we represent metamodels and its instance models in a single EAV designed repository in to support of model transformations and introducing a new concept of what we call it an Open Source Metamodel. Also this work demonstrate an integration between UML static and behavioral models.

Keywords : *Metamodel Representation, Model Representation, Entity-Attribute-Value, EAV, Open Source Model, MDA.*

1. INTRODUCTION

Early, in software development lifecycle, models have been employed to address structural elements in the design phase, as well as in the testing phase for models checking and verification. Although, these stages are tightly interconnected with each other, but the absence of a unified way to express different level of abstraction concepts limited the use of models for design and system documentation [1].The Model Driven Architecture use Models and Metamodels as a keystone in software development process. The metamodel represents the conceptual model of a design language, while the instance generated from such particular design in a design language is called Instance Model [2]

The development lifecycle in MDA divided to platform independent model (PIM) and platform specific models (PSM). Both models are working in different level of abstractions[3]. UML/MOF are a common OMG standard tools that normally used in model driven development to design models and metamodels. Model transformation is one of the main activity in model driven software that normally serve in transform high level models to low level models using model transformation tools such as Query-View-Transformation (QVT) and Atlas Transformation Language (ATL). Together with Computer Aided Software Engineering (CASE) tools UML and other transformation tools are closely related to database schema. The database supporting such tools is often called a repository[4].

MDA Models can be expressed visually or textually[5].The visual representation of models is normally concerned with the functional requirements. Hence, in some cases some non-functional requirements can be addressed through transformation rules or at the level of the model by the adoption of UML Profiles and/or Templates[6]. For the textual representation of models,[5, 6] suggested the embedding of the transformation rules at the model level in an XMI textual annotation to cover both, functional and non-functional requirements. Typical model representations (Visual and Textual)are imprecise, incomplete, lack models interoperability, and as such do not lead to running applications[1].

In this paper we propose a novel approach for representing models and metamodels using the Entity-Attribute-Value (EAV)concept [7]. The approach combined both, models and metamodels representation in a single EAV designed repository. Announcing the born of new concept called an Open Source Metamodel.

In Section 2 of this paper, list out the related work. The Entity-Attribute-Value concept highlighted in Section 3. Section 4 presents Models and Metamodels representation. In Section 5 we

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

introduced the Open Metamodel Concept. The results and discussion is in Section 6. Conclusions and future work are discussed in section 7.

2. RELATED WORKS

The work on[2]is closely related to our work. However, both models and metamodels presented separately using a conventional database model. While our work combined metamodels with their instant models by employing EAV concept, benefiting from its open structure flexibility, as there are no limits on number of attributes per entity. Therefore, there is no need to redesign the schema upon models or metamodels grow. Also the self-describing data and the simple physical data format of EAV make it much practical when representing models and metamodels. This is beside the ''Object-at-a-time'' queries against a highly complex logical schema that are significantly easier to implement with EAV than with conventional structure.

3. ENTITY-ATTRIBUTE-VALUE CONCEPT

EAV is widely used in the medical and clinical information system as a general purpose means of knowledge representation. The Attribute-value pairs concept are an esteemed way of representing information on an object, originated on 1950s on the LISP association lists[7]. An example of attribute-value pairs showing a particular student information would be: ((IndexNoA3) (ProgramCS101) (GPA3.1) (Year2012) (Status Active)).

Unlike the conventional database the EAV design does not support or conform to rules of database normalization [8], where the attribute-value pairs become triples with the entity (the thing being described, identified with a unique identifier of some sort) repeating in each row of a table.

Extensible Markup Language (XML) [4] syntax is related to attribute-value pairs. XML elements, delimited within open- and close-tags for ease and accuracy of parsing, can represent either entities or attributes. They can contain sub-elements nested to arbitrary levels; sub-elements may be regarded as attributes with complex structure. For convenience, atomic data describing an entity may also be represented within an element's open-tag as attribute-value pairs, each component of a pair being separated by an equal sign.

4. MODEL AND METAMODEL REPRESENTATION

In this part we are presenting how we represents models and metamodels in EAV format. Figure 1 show our instance model that we designed by a simple State Machine design language for an application in which Passengers buy tickets at the time they obtain reservations. At check-in time they obtain boarding cards if there are still seats available. Due to over booking of flights they may be rescheduled on later flights.

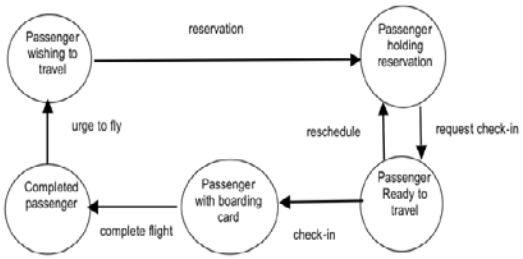

Figure 1: A State Machine Model For Airline Passenger

Some of the information in the Airline Passenger model is implicit. In this situation, we need to interpret the graphical objects in the diagram, which we do by consulting the documentation of the State Machine modeling language and its particular representation in this case.

Here, there are three types of object:

States, represented by ovals, each of which has a name, represented by the text contained in the oval.

Transitions, represented by arrows. A transition is from a source state (represented by the plain end of the arrow) to a target state (represented by the end of the arrow with an arrowhead).

Events, each of which is associated with a transition. An event is represented by a name near the arrow representing the associated transition. The diagram contains five instances of State: Passenger

A metamodel representing the concept in Figure 1 is shown in Figure 2, represented as a UML Classes diagram. Note that the instances in the diagram of Figure 1 do not appear in the metamodel of Figure 2. Note also the metaclass NamedElement, which is a superclass of the meta-classes State and Event. The states and events of Figure 1 are all named. The metaclass NamedElement supplies an attribute name to its subclasses.

Journal of Theoretical and Applied Information Technology
30th November 2013. Vol. 57 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

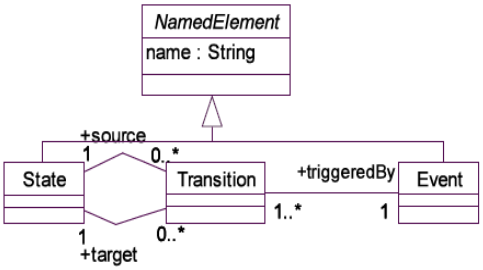

Figure 2: UML Class Diagram for State Machine

Metamodels are closely related to database schemas. Instances of the concepts specified in the model are stored in a database specified by the schemas developed from the metamodel. Figure 3 shows the instances in the class list of Figure 1 represented in a database whose schema is developed from the metamodel of Figure 2.

Notice that the population of the database in **Appendix C** consists entirely of tuples of literals. Each column of each table is relational attribute of a literal type. A column in a table is ultimately derived from a literal-valued attribute in the UML Classes model of **Appendix B**. We can think of the population of the database as a collection of literals organized according to the classes, associations and attributes in the Classes model.

In the same way, the instances in the Airline Passenger model of Figure 1 can be represented as a population of a database whose schema is developed from the metamodel of Figure 2, as shown in **Appendix A**. (Abbreviation used for more space). This database is the repository of a modelling tool supporting the simple State Machine design language. Here the columns are all derived from the name attribute of the class NamedElement in **Appendix A**. This conventional representation for the database tables State and Event, where they have only one column, name. The table Transition has three columns, all foreign keys. Two are derived from the name attribute of the class State and one from the name attribute of the class Event. Without the attribute name in NamedElement, it would be impossible to create a repository schema that would record the Airline Passenger model of **Appendix A**.

A further issue is that a relational schema requires that for each table certain attributes be declared to be the key for the table. That is, a row in the table can be identified by looking at the values of the key attributes. Knowing the values of the key attributes, we can look in the table to find the values of the other attributes in the row. Some metamodeling languages allow the specification of identifiers[9]. Entity-Relationship Modeling [10]and Object-Role Modeling [11] both support identifiers. UML, however, does not [2]. If UML is used as the metamodeling language, then additional information must be supplied to designate some attributes in the repository schema to be keys.

In the STM repository of **Appendix A**, the tables State and Event both have the attribute name as key, while the Transition table has a key composed of the three attributes source, target and TRIGEREDBY.

Once we have a schema and a population for an application, we can use the query language associated with the database system to make queries about the population. Queries are typically about the semantics of the application. Nevertheless, any change on the metamodel in Figure 2 should by reflected on its instance model in Figure 1 and consequently in the database in **Appendix A**. However, because of the conventional database structure a Data Definition Language (DDL) statements should be used. For example to add new attribute to the table Event or State an Alter table statement should be employed. Which normally done by the model designer who's not necessary the one who is doing the development. On the other hand, most of the modelling tools does not allowing any changes on their main metamodel on which they developed based on it. To overcome this limitation a dynamic structure employed to replace the conventional schema in **Appendix A** by EAV structure in **Appendix B**. The open structure of EAV treat all the tables in the conventional schema as a tuple entry in a single EAV table. The thing that gives more control in managing models dynamicity, upgrade and maintenance.

Structure-oriented queries are important in Modelling tool applications. For example, a state machine can have an initial state (a state with no transitions in) or a final state (a state with no transitions out). These states can be identified respectively by the following two views

```
CREATE VIEW InitialState(StateName) AS(
SELECT A.Value_ FROM EAV A
Where A.ENTITY = 'STATE' AND A.ATTRIBUTE =
'NAME'
AND
A.Value_ NOT IN (
SELECT B.Value_ FROM EAV B
WHERE
       B.ENTITY = 'TRANSITION'
        AND
        B.ATTRIBUTE = 'TARGET'))

CREATE VIEW FinalState(StateName) AS(
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
SELECT A.Value_ FROM EAV A
Where A.ENTITY = 'STATE' AND A.ATTRIBUTE =
'NAME'
AND
A.Value_ NOT IN (
SELECT B.Value_ FROM EAV B
WHERE
       B.ENTITY = 'TRANSITION'
        AND
   B.ATTRIBUTE = SOURCE))
```

The above two views return no data because the state machine in Figure 1 is cyclic. Hence, we interested to validate whether our state machine design is entirely cyclic, with neither initial nor final states.

```
CREATE VIEW CyclicModel(Cyclic) AS
SELECT "Cyclic" FROM State WHERE
       NOT EXISTS SELECT * FROM
InitialState
       AND
       NOT EXISTS SELECT * FROM FinalState
```

In particular, Modelling Tool repositories are intended to store designs, which are often expressed in graphical languages (like UML). The two-dimensional nature of graphical languages makes it relatively easy to have a design language where the design concepts are expressed as a complex structures. These complex structures generally have formation rules (Constrains), which can be checked by structural queries. Structural queries therefore are more important for modelling tools than for general database applications.

An example of a design language (metamodel) with complex structures having constrains is our simple State Machine language of Figure 2. An instance of Transition is necessarily linked to two instances of State and one instance of Event. A structural query whose result is violations of this constrain is

```
   SELECT * FROM EAV A WHERE
   A.ENTITY = 'TRANSITION' AND
   NOT EXIST(
   SELECT * FROM EAV B WHERE B.ENTITY =
'STATE'
   AND B.ATTRIBUTE = 'NAME'
   AND B.VALUE_ IN
   (SELECT B1.VALUE_ FROM EAV B1 WHERE
   B1.ENTITY = 'TRANSITION'AND
   B1.ATTRIBUTE ='SOURCE')
   AND
   SELECT * FROM EAV C WHERE C.ENTITY =
'STATE'
   AND C.ATTRIBUTE = 'NAME'
   AND C.VALUE_ IN
   (SELECT C1.VALUE_ FROM EAV C1 WHERE
   C1.ENTITY = 'TRANSITION'AND
   C1.ATTRIBUTE ='TARGET')
     AND
   SELECT * FROM EAV D WHERE D.ENTITY =
'EVENT'
   AND D.ATTRIBUTE = 'NAME'
   AND D.VALUE_ IN
         (SELECT D1.VALUE_ FROM EAV
   D1 WHERE D1.ENTITY = 'TRANSITION'AND
   D1.ATTRIBUTE ='TRIGEREDBY')
   )
```

Additional constrains can be added in to a given design, for example that there be exactly one initial state and exactly one final state, or that there be no isolated states.

Some modeling languages allow constraints to be represented by annotations on the model, but it may not tell a designer how to concretely represent a design. For example, the UML model of Figure 2 does not tell the designer enough to be able to represent the design of the Airline Passenger state model so that it looks like Figure 1. To do this, the conceptual model must be augmented by some rendering conventions. However, we are implementing this by joining both model and metamodel presented in Figure 1 and Figure 2 respectively in a single EAV structure, shown in the next part.

5. THE OPEN SOURCE METAMODEL

In this part we combining a Metamodel in Figure 2 with its Instance Model in Figure 1. Since the documentation one of the modelling purpose, we add some basic information about the model. **Appendix C** presented the state machine in Figure 1 combined with its metamodel in Figure 2.

The Entity column in the above EAV structure can include several attributes separated by "." to address different areas in the representation of the models and metamodels. To realize this the Entity "Metmodel.Elmnt.NameElmnt.EVNT" and "Metmodel.Elmnt.NameElmnt.State" can be queried to list the correspondence data that inherited from the NameElmnt at the metamodel level as well as the model level as per below query.

```
    SELECT * FROM EAVRepository
    WHERE
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
ENTITY
LIKE('Metamodel.Elmnt.NameElmnt%')
```

Different scenarios can be implemented where the instant model can be addressed without the metamodel or vice versa. That's why it is advisable to create different views for each area of interest.

```
1:     </EAV>
2:     <EAV>
3:     <Row
4:
5:  ENTITY="Metmodel.Elmnt.NameElmn.EVNT
    "
6:           ATTRIBUTE="NAME"
7:           VALUE_="complete"
8:         />
9:     </EAV>
```

Figure 3: A Fragment Of A Combined Metamodel With Its Instance Model In Single EAV XMI Repository

The MDA tools along with other modelling tools support the XML/XMI format to support the interoperability, model interchange and code generation. The SQL/XML standard is ISO/IEC 9075–14:2005(E), Information technology – Database languages – SQL – Part 14: XML-Related Specifications (SQL/XML). As part of the SQL standard, it is aligned with SQL:2003 [3]. Figure 3 show apart from an XML representation to the EAVRepository table.

6. RESULTS AND DISCUSSION

Metamodels are combining a set of concepts and corresponding mechanisms that allow to "model" formally different contexts (e.g. #business processes/activities) with the same point of view. Similar to the open source software concepts, having suchcapability of representing metamodels and its instance models in an XML/XMI format in a single repository enables for instance, to manage models formalizing each one subset of an overall operational context, keeping it consistent with the others.

Therefore, consistency between models presenting the same type of point of view is one of the key interest of having one metamodel and models in the same repository. This EAV structure is capable to handle several metamodels in a single repository as well. Of course, this makes sense on condition that each metamodel address a point of view different from the points of views of the other metamodels. Having this capability, one can represent different points of views of the same context.

Normally, it is hard to validate the correctness of the models before development. So, the communication between the artifact designer and the developer is very crustal. Hence, it is hard to keep the models and development artifacts in synchronization during the development and maintenance phases. The open source metamodels concepts gives better control and quality on metamodels and its generated models: when defining and changing the metamodel it possible to immediately check how it influences to the models. This gives immediate feedback, testability and incremental metamodel definition. This is in sharp contrast to the ways how metamodels are defined in some standardization organizations where metamodels are not executed or tested with models (but stay as a document).

This is beside the great support to the model evolution: with proper mechanisms in place there is a flexibility to ensure that models will work, open in editors, produce the code etc. with the newer metamodel too (e.g. updates automatically the models to the new metamodel).

There are also other advantages like faster metamodel/language development, easier management, possibility to couple various generators based on the metamodel together, etc. The think that support software product line productivity.

Under the MDA context the static models (Class diagram) has a capability of 1 to 1 mapping to implementation (source code) potentially. However, the behavioral models (State Machine) arenormally lack of capability for entire code generation. Considering code generation from behavioral diagram, it is possible to generate the skeleton of method invocations, however, it is impossible to generate the content codes of methods(functions/operation). Otherwise, it is necessary to specify same description like the source codes. The proposed approach demonstrated the capability of integration between UML behavioral models (State Machine Diagrams) with Static model (Class Diagrams). Consequently, more controls are provided concerning the transformation to code.

The limitation of the Open Source Metamodelis inherited from EAV representation drawbacks. Where a considerable up-front programming is needed to do many tasks that a conventional architecture would do automatically. Moreover,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

such programming needs to be doneonly once, and availability of generic EAV toolscould remove this limitation. Also, for bulk retrieval EAV design is considered less efficient than a conventionalstructure. Consequently, performing complex attribute-centricqueries, which are based on values of attributes, and returning a set of objects is both significantly less efficient as well as technically more difficult.

7. CONCLUSIONAND AND FUTURE WORK

In this paper we have presented a new concept of Open Source Metamodel where we represented a metamodel combined with its instance models inspired by the Entity-Attribute-Value concept. Both the metamodel and its model represented in a single repository. Having is repository in XML/XMI format make it exchangeable and accessible to most of CASE tools in general and MDA transformation tools in specific.

The paper focused on the representation of models and metamodels under the MDA context. However, in the near future we plan to bring the Domain Specific Language (DSL) on board in order to standardize and simplify the repository update and population.

Also our intention to use this approach for platform representation in to support of a transformation to a particular platform executable code.

ACKNOWLEDGEMENT

The authors would like to express their deepest gratitude to UniversitiTeknologi Malaysia (UTM) for their financial support under Research University Grant Scheme.

REFRENCES:

[1] H. Kern, et al., "Towards a comparative analysis of meta-metamodels," in Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11, 2011, pp. 7-12.

[2] R. M. Colomb, "Metamodelling and Model-Driven Architecture," Faculty of Computer Science and Information Systems University of Technology Malaysia, vol. 1, 2009 2009.

[3] A. Eisenberg, et al., "SQL: 2003 has been published," ACM SIGMoD Record, vol. 33, pp. 119-126, 2004.

[4] T. Kiefer and M. M. Nicola, "Generating structured query language/extensible markup language (SQL/XML) statements," ed: Google Patents, 2012.

[5] M. Peltier, et al., "MTRANS: A general framework, based on XSLT, for model transformations," in Workshop on Transformations in UML (WTUML), Genova, Italy, 2001.

[6] M. Peltier, et al., "On levels of model transformation," in XML Europe, 2000, pp. 1-17.

[7] V. Dinu and P. Nadkarni, "Guidelines for the effective use of entity-attribute-value modeling for biomedical databases," International journal of medical informatics, vol. 76, p. 769, 2007.

[8] W. Kent, "A simple guide to five normal forms in relational database theory," Communications of the ACM, vol. 26, pp. 120-125, 1983.

[9] D. Varró and A. Balogh, "The model transformation language of the VIATRA2 framework," Science of Computer Programming, vol. 68, pp. 214-234, 2007.

[10] I.-Y. Song and K. Froehlich, "Entity-relationship modeling," Potentials, IEEE, vol. 13, pp. 29-34, 1994.

[11] T.Halpin, "Object-role .modeling (ORM/NIAM)" in Handbook on Architectures of Information Systems, ed: Springer, 2006, pp. 81-103.

Journal of Theoretical and Applied Information Technology
30th November 2013. Vol. 57 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

APPENDICES

APPENDIX A

Airline Passenger state model of Figure 1 represented as a conventional database population

State
Name
WishTravel
Completed
HoldRes
ReadyTravel
WBoardCard

Event
Name
reservation
reschedual
reqCheckIn
checkIn
complete
urgeFly

Transition		
Source	Target	triggeredby
WishTravel	HoldRes	reservation
HoldRes	ReadyTravel	reqCheckIn
ReadyTravel	HoldRes	reschedual
ReadyTravel	WBoardCard	checkIn
WBoardCard	Completed	complete
Completed	WishTravel	urgeFly

APPENDIX B

Airline Passenger state model of Figure 1 represented in EAV database population

ENTITY	ATTRIBUTE	VALUE_
EVENT	NAME	checkIn
EVENT	NAME	Complete
EVENT	NAME	reqCheckIn
EVENT	NAME	Reschedule
EVENT	NAME	Reservation
EVENT	NAME	urgeFly
STATE	NAME	Completed
STATE	NAME	HoldRes
STATE	NAME	ReadyTravel
STATE	NAME	WBoardCard
STATE	NAME	WishTravel
TRANSITION	SOURCE	Completed
TRANSITION	SOURCE	HoldRes

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

TRANSITION	SOURCE	ReadyTravel
TRANSITION	SOURCE	ReadyTravel
TRANSITION	SOURCE	WBoardCard
TRANSITION	SOURCE	WishTravel
TRANSITION	TARGET	Completed
TRANSITION	TARGET	HoldRes
TRANSITION	TARGET	HoldRes
TRANSITION	TARGET	ReadyTravel
TRANSITION	TARGET	WBoardCard
TRANSITION	TARGET	WishTravel
TRANSITION	TRIGGEREDBY	checkIn
TRANSITION	TRIGGEREDBY	complete
TRANSITION	TRIGGEREDBY	reqCheckIn
TRANSITION	TRIGGEREDBY	reschedule
TRANSITION	TRIGGEREDBY	reservation
TRANSITION	TRIGGEREDBY	urgeFly

APPENDIX C

Combined Metamodel with its Instance Model in Single EAV Database Population

ENTITY	ATTRIBUTE	VALUE_
Metamodel	ID	1
Metamodel	Version	1.1
Metamodel	Name	State Machine
Metamodel	Date	25-Jun-13
Metamodel.Element	ID	1.1.1.1
Metamodel.Element	Name	NamedElement
Metamodel.Element.NamedElement	DataType	String
Metamodel.Element.NamedElement	Attribute	Name
Metamodel.Element.NamedElement.EVENT	NAME	checkIn
Metamodel.Element.NamedElement.EVENT	NAME	complete
Metamodel.Element.NamedElement.EVENT	NAME	reqCheckIn
Metamodel.Element.NamedElement.EVENT	NAME	reschedule
Metamodel.Element.NamedElement.EVENT	NAME	reservation
Metamodel.Element.NamedElement.EVENT	NAME	urgeFly
Metamodel.Element.NamedElement.EVENT	NAME	Completed
Metamodel.Element.NamedElement.EVENT	NAME	HoldRes
Metamodel.Element.NamedElement.EVENT	NAME	ReadyTravel
Metamodel.Element.NamedElement.EVENT	NAME	WBoardCard

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Metamodel.Element.NamedElement.EVENT	NAME	WishTravel
Metamodel.Element.NamedElement.NAME	NAME	Completed
Metamodel.Element.NamedElement.NAME	NAME	HoldRes
Metamodel.Element.NamedElement.NAME	NAME	ReadyTravel
Metamodel.Element.NamedElement.NAME	NAME	WBoardCard
Metamodel.Element.NamedElement.NAME	NAME	WishTravel
Metamodel.Element.TRANSITION	SOURCE	Completed
Metamodel.Element.TRANSITION	SOURCE	HoldRes
Metamodel.Element.TRANSITION	SOURCE	ReadyTravel
Metamodel.Element.TRANSITION	SOURCE	ReadyTravel
Metamodel.Element.TRANSITION	SOURCE	WBoardCard
Metamodel.Element.TRANSITION	SOURCE	WishTravel
Metamodel.Element.TRANSITION	TARGET	Completed
Metamodel.Element.TRANSITION	TARGET	HoldRes
Metamodel.Element.TRANSITION	TARGET	HoldRes
Metamodel.Element.TRANSITION	TARGET	ReadyTravel
Metamodel.Element.TRANSITION	TARGET	WBoardCard
Metamodel.Element.TRANSITION	TARGET	WishTravel
Metamodel.Element.TRANSITION	TRIGGEREDBY	checkIn
Metamodel.Element.TRANSITION	TRIGGEREDBY	complete
Metamodel.Element.TRANSITION	TRIGGEREDBY	reqCheckIn
Metamodel.Element.TRANSITION	TRIGGEREDBY	reschedule
Metamodel.Element.TRANSITION	TRIGGEREDBY	reservation
Metamodel.Element.TRANSITION	TRIGGEREDBY	urgeFly