Journal of Theoretical and Applied Information Technology
10th October 2014. Vol. 68 No.1

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

PROPOSE AN INTEGRATION BETWEEN UML STATIC AND DYNAMIC MODELS USING ENTITY-ATTRIBUTE-VALUE UNDER THE MDA CONTEXT

[1]AHMED MOHAMMED ELSAWI, [2]SHAMSUL SAHIBUDDIN, [3]ABDELHAMID ABDELHADI
[1]Software Engineering Department, UniversitiTeknologi Malaysia, Malaysia
[2]Prof., Software Engineering Department, UniversitiTeknologi Malaysia, Malaysia
[3]Asstt. Prof., Faculty of Computer Science and Information System, University of Khartoum, Khartoum
E-mail: [1]elsawi@gmail.com, [2]shamsul@utm.my, [3]abhamidhn@gmail.com

ABSTRACT

The Model Driven Architecture (MDA) is adopting models to improve the software productivity, reusability, maintainability and quality by focusing on models and metamodels in place of conventional code. The MDA separates the technical details from the business logic in two different models. The Platform Independent Model (PIM) is concerned with the business logic while the Platform Specific Model (PSM) is more focusing on the targeted platform. Normally, PIM and PSM models stand in different level of abstraction. Moving from one level of abstraction to another is achieved by Model transformation. Both PIM and PSM are modeled using UML diagrams. The UML supports a variety of diagrams that can be categorized into static and dynamic diagrams. The static diagrams are normally targeted the system's structures and it is commonly used to define the PIM and PSM models. On the other hand, the dynamic diagrams are targeting the system's behavior and its dynamic elements. To successfully develop a complete software using the MDA methodology, all structural and behavioral elements should be captured. Hence, different versions of the PIM and PSM models should be employed to cover the structural and behavioral elements of the system. Consequently, beside time, cost, and complexity issues a considerable number of model transformation iterations are required for each version separately. Into face of these issues, we propose this work to address the integration between UML behavioral and structural diagrams using the Entity-Attribute-Value (EAV) model. Also, we presented an example to show how this proposed concept not only allowing for an integration between UML static and behavioral models, but also shows the flexibility of integration models in different level of abstraction.

Keywords: *UML Models Integration, behavioral models, Static Models, MDA, EAV*

1. INTRODUCTION

Earlier, in the 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 levels of abstraction concepts limited the use of models for design and system documentation .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 [1].

The development Lifecycle in MDA divided into platform independent model (PIM) and platform specific models (PSM). Both models are working in different level of abstractions [2]. 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 activities in model driven software that normally serve in transforming 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 [3].

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

MDA Models can be expressed visually or textually[4].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 [5]. For the textual representation of models, [4, 5] 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[6].

UML does have a wide-ranging of behavior models. These behavior models permit the specification of a complete range of behaviors. These specifications are normally static. The class diagram can be one of these models that commonly used to describe the model's specifications. But on the other hand, the semantics of the behaviors are not included in the models as it is not included in the static model specifications. In this paper, we employ the knowledge representation capability of the Entity-Attribute-Value (EAV) concept [7] to integrate UML behavior and static models. The approach combined both, Static and dynamic models in a single EAV designed repository. The model constraints are managed by a structuring query, which based on our previous work in [8].

In Section 2 of this paper, we list out the related work concerning the integration between UML models. The Entity-Attribute-Value concept highlighted in Section 3. Section 4 presents Models and Metamodels representation. In Section 5 we show the integration between static and behavioral model concept. The results and discussion are in Section 6. Conclusions and future work are discussed in section 7.

2. RELATED WORKS

Integration of static and dynamic UML models has been addressed by [9]. By mapping the static and behavioral elements of the UML metamodels into Abstract State Machine (ASM), they convert the structural model elements into ASM vocabulary that representing a group of functions and domains. In this approach the focus was in representing the UML metamodel structure in ASM vocabulary semantics. They deviated from the standard UML diagrams by the ASM semantics. While our work is using the standard UML diagrams as it is, and integrated both behavioral and structural diagrams in a single EAV repository. On the other hand, their approach can be applied at a very high level of metamodels and abstractions while ours can go for different levels of abstractions (Metamodels/Models). This is beside the fact that their work is only taken into account model elements from a static view, dropping any relation to their actual, dynamic semantics. Consequently, their formal semantics are not enough to achieve an integration between UML structural and behavioral diagrams.

From model's representation prospective, the work on [1] is closely related to ours. Were they adopted the database model to represent models. However, both static and dynamic models presented separately using a conventional database model. On the other hand, the focus wasn't targeting the integration between static and behavioral models. While this work combines the model's structure and behavior in the EAV representation repository, benefiting from its open structure flexibility. Therefore, there is no need to redesign the schema upon a change in the model's structure or behavior. Also the self-describing data and the simple physical data format of EAV makes 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 their conventional structure.

In the next section we brief about EAV concept and show its strength in knowledge representation for models and metamodels.

3. THE 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 is 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[10], 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.

Journal of Theoretical and Applied Information Technology
10th October 2014. Vol. 68 No.1

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

Extensible Markup Language (XML) [3] 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

Normally, metamodels sitting in a level higher than it is instance model. In this part we are showing how we represent models and metamodels using EAV concept. Figure 1 shows our instance model that we designed with 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 overbooking of flights they may be rescheduled on later flights.

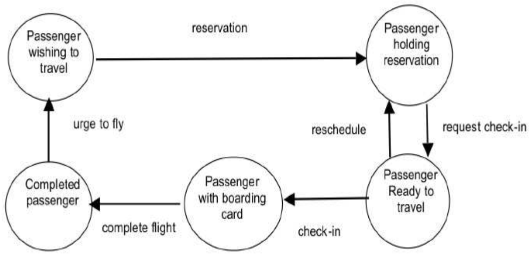

Figure 1: A State Machine Model For Airline Passenger

The State Machine diagram is a UML diagram that shows systems behavior and dynamic characteristics. Some of the information in this diagram are is implicit. In this situation, we need to interpret the graphical objects in the above 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

The static UML class diagram metamodel in Figure 2 is representing the concept shown in Figure 1. 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.

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. Appendix A shows the instances in the class list of Figure 1 represented in a database whose schema is developed from the metamodel of Figure 2.

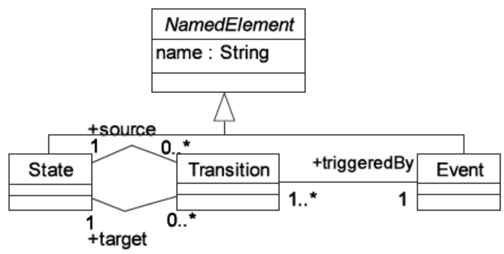

Figure 2: UML Class Diagram For State Machine

Notice that the population of the database in Appendix A 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 Class 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 Class 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. (An abbreviation used for more space). This database is the repository of a modelling tool supporting the simple State Machine design language. The columns here are all derived from the name attribute of the class NamedElement in Appendix A. This conventional representation of the database tables State and Event, where they have only one column, name. The table Transition has three columns, all are 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

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

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 are 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 at the table to find the values of the other attributes in the row. Some metamodeling languages allow the specification of identifiers [11]. Entity-Relationship Modeling [12] and Object-Role Modeling [13] both support identifiers. UML, however, does not [1]. 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 triggeredBy.

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 be reflected in 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 necessarily the one who is doing the development. On the other hand, most of the modelling tools do not allow any changes on their main metamodel on which they developed based on it. Beside the fact that the change in the diagram structure can't propagate easily to other behavioral diagrams attached to it. To overcome this limitation a dynamic structure employed to replace the conventional schema in Appendix A by an EAV structure in Appendix B. The open structure of EAV treats 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(
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 structure. 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 structured query whose result is violations of this constraint is

```
SELECT * FROM EAV A WHERE
A.ENTITY = 'TRANSITION' AND
NOT EXIST(
SELECT * FROM EAV B WHERE B.ENTITY =
```

Journal of Theoretical and Applied Information Technology
10th October 2014. Vol. 68 No.1

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

```
'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 ='TRIGGEREDBY')
)
```

Additional constraints 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 static model, but it may not tell a designer how to concretely represent a design. Instead, a separate dynamic version of the static model may employ to address this issue. For example, the static class diagram in Figure 2 does not tell the designer enough to be able to represent the behavioral characteristics of the Airline Passenger state model presented in Figure 1. To do this, the static conceptual model must be augmented by some rendering conventions. However, we are implementing this by joining both, the behavioral model and structural static metamodel presented in Figure 1 and Figure 2 respectively in a single EAV structure, shown in the next part.

5. INTEGRATION BETWEEN STATIC AND DYNAMIC MODELS

In this part we combine the static class diagram (Matamodel) in Figure 2 with its behavioral Instance Model in Figure 1. Since the documentation one of the modelling purpose, we have added some basic information about the model. Appendix A presented the dynamic state machine model in Figure 1 combined with its static structure metamodel in Figure 2.

The Entity column in Appendix C is EAV structure that can include several attributes separated by "." to address different areas in the representation of the dynamic models and static metamodels. To realize this the Entity "Metamodel.Element.NamedElement.EVENT" and "Metamodel.Element.NamedElement.State" can be queried to list the correspondence data that inherited from the NamedElement at the metamodel level as well as the model level as per below query.

```
SELECT * FROM EAVRepository
WHERE
ENTITY
LIKE('Metamodel.Element.NamedElement%')
```

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

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 [2]. The below show apart from an XML representation to the EAVRepository table

```
<EAV>
<Row

ENTITY="Metamodel.Element.NamedElement.EVEN
T"
        ATTRIBUTE="NAME"
        VALUE_="complete"
      />
</EAV>
```

6. RESULTS AND DISCUSSION

Normally, Metamodels represented in a static UML Models. They 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. The different in context can include the structural and behavioral characteristics beside the consideration of their different level of abstractions. Having such capability of representing metamodels

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

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, automation of change propagation can be achieved with less time and cost.

Therefore, integration and consistency between models presenting the same type of point of view is one of the key interest of having one static and dynamic 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. Opening the door to bring more structural and dynamic models in different level of abstraction integrated together to support the MDA vision of end to end development of software applications using modeling language.

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 adoption of this approach 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 thing ,ml 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) are normally 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 from models.

The limitation of this approach is 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, such programming needs to be done only once, and availability of generic EAV tools could remove this limitation. Also, for bulk retrieval EAV design is considered less efficient than a conventional structure. Consequently, performing complex attribute-centric queries, 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. CONCLUSION AND FUTURE WORK

In this paper we have presented a new concept of integration between UML static and dynamic models where we represented a static metamodel class diagram combined with its instance dynamic state machine models inspired by the Entity-Attribute-Value concept. Both static and its dynamic 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 and integration of structural and behavioral models 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 computer platform representation in to support of a transformation to a particular platform executable code.

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

REFRENCES:

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

[2] A. Eisenberg, J. Melton, K. Kulkarni, J.-E. Michels, and F. Zemke, "SQL: 2003 has been published," *ACM SIGMoD Record,* vol. 33, pp. 119-126, 2004.

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

[4] M. Peltier, J. Bézivin, and G. Guillaume, "MTRANS: A general framework, based on XSLT, for model transformations," in *Workshop on Transformations in UML (WTUML), Genova, Italy*, 2001.

[5] M. Peltier, F. Ziserman, and J. Bézivin, "On levels of model transformation," in *XML Europe*, 2000, pp. 1-17.

[6] H. Kern, A. Hummel, and S. Kühne, "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.

[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] A. M. ELSAWI, S. SAHIBULDIN, and A. ABDELHADI, "INTRODUCING THE OPEN SOURCE METAMODEL CONCEPT," *Journal of Theoretical & Applied Information Technology,* vol. 57, 2013.

[9] A. Cavarra, E. Riccobene, and P. Scandurra, "Integrating UML static and dynamic views and formalizing the interaction mechanism of UML state machines," in *Abstract State Machines 2003*, 2003, pp. 229-243.

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

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

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

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

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

APPENDICES

APPENDIX A:

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

Journal of Theoretical and Applied Information Technology
10th October 2014. Vol. 68 No.1

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

APPENDIX B:

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

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

APPENDIX C:

ENTITY	ATTRIBUTE	VALUE_
Metamodel	ID	1
Metamodel	Name	State Machine
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	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
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.TRANSITION	SOURCE	Completed
Metamodel.Element.TRANSITION	SOURCE	HoldRes
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

Appendix C: Fragment of EAV representation to models in Figure 1 and Figure 2