Towards the Initial Conceptual Database Model through the UML Metamodel Transformations

Drazen Brdjanin[*] and Slavko Maric[†]
[*]Faculty of Electrical Engineering, University of Banja Luka, Bosnia and Herzegovina, e-mail: bdrazen@etfbl.net
[†]Faculty of Electrical Engineering, University of Banja Luka, Bosnia and Herzegovina, e-mail: ms@etfbl.net

***Abstract*—This paper presents an ATL-based automatic generator of the initial conceptual database model. The implemented generator takes the detailed UML business activity diagram as the source business model and generates the UML class diagram representing the target initial conceptual database model.**

***Index Terms*—business activity diagram; class diagram; conceptual database model; ATL; UML.**

I. Introduction

The database development process mainly undergoes the following phases: requirements analysis, conceptual design, logical design and physical design. Each design phase ends in an appropriate model, which is characterized by the presented abstraction level on the one hand, and the reached level of the implementation details of the target database on the other. The main goal of conceptual design is to provide an overall view of information in the entire system. The corresponding model is usually called the *conceptual database model* and represents a semantic data model. The related literature is more significantly focused on conceptual design than on other phases, considering it to be the most important design phase.

This paper presents an approach to the automated generation of the initial conceptual database model based on the automatic extraction of business entities from the business model and the automatic generation of their relationships. The detailed UML business activity diagram, as a frequently used business process modeling notation, is taken as the source model for the business entities extraction. The generation of the target UML class diagram, which represents the initial conceptual database model, is driven by ATL transformation rules.

The paper is structured as follows. After the introduction, the second section gives the related work overview. The detailed business activity diagrams as the basis for automated initial conceptual design are described in the third section. The fourth section presents the transformation rules for the automatic generation of the class diagram. The experimental results of the generator's application in a concrete business system are given in the fifth section. Finally, we conclude the paper and highlight the directions for further research.

II. Related Work

Although the idea of conceptual database model design based on the business model is not very new, there are only a few papers that present the implemented automatic generator and provide experimental results, while the others just give the method overview.

Kamimura *et al.* in [1] propose a detailed algorithm for generating the E-R diagram using the *well-disciplined* IDEF0-based business model, but without an actual implementation.

Garcia Molina *et al.* in [2] propose an approach to the transition from business models to the initial conceptual model, which is based on detailed UML activity diagrams and the supplementary glossary. They propose the direct mapping of all information objects in the activity diagram to the respective classes in the target class diagram, but don't propose the creation of classes for business process participants. They base the creation of class associations on business rules informally specified in the glossary, which is not suitable for automatic generation. Suarez *et al.* in [3] follow the previous approach and propose an improvement in creating class associations. They propose creating associations for activities that have input and output objects by direct mapping of those activities to the respective associations between classes that correspond to input and output objects. This proposal can be used for the automated generation of associations, but has limitations related to the automated generation of association multiplicity, since they don't propose the explicit rules.

Besides the proposal for direct mapping of all business objects to the respective classes in the target class diagram, Brdjanin and Maric in [4] propose direct mapping for all business process participants as well, and define rules for the creation of associations between business participants and business objects based on activities performed on those objects. This approach is a basis for the *ADBdesign* - the automatic generator of the initial conceptual database model, presented in [5]. In this paper, we provide the ATL-based transformation rules for the automatic generation of the target class diagram, which follow the same approach.

III. Detailed Business Activity Diagrams

The UML activity diagram is a widely accepted business modeling notation [6]. In business modeling, the activity diagram can be used in two ways [7]. The first form is the so-called *macroactivity diagram*, which is used in developing the initial business model for the rough specification of activities which realize business use cases, i.e. for modeling business processes at a high level of abstraction. The second form is the so-called *detailed activity diagram* which is developed by populating the macroactivity diagram with all business process participants (business actors, workers, organizational units, etc.), used business objects and object flows.

Detailed business activity diagrams represent a reliable starting point for developing the initial conceptual database model, because business objects, object flows, activities on objects, and participants' responsibilities fully document the identified business use cases. It is these details (participants and objects) that are the basis for the initial conceptual model, because they are directly mapped into the classes of the target class diagram, while the identified activities are mapped into the class associations with the appropriate multiplicity, as given in [4] and [5].

The participants' areas of responsibility are modeled by swimlanes. All activities, objects and object flows related to a participant are shown in the same swimlane. Hence, there are as many swimlanes as there are participants involved in the process. The input of an object into an activity represents an input object flow directed from the object toward the activity, while the output of an object from an activity represents an output object flow directed from the activity toward the object.

In this paper we use the standard UML notation for the business activity diagram representation. In this way, business activities are modeled as the `OpaqueAction` nodes, while business objects are modeled as the `CentralBufferNode` nodes. Each swimlane is modeled as `ActivityPartition`. The UML metamodel excerpt from the UML superstructure [8] used for the detailed activity diagram representation is shown in Fig. 1.

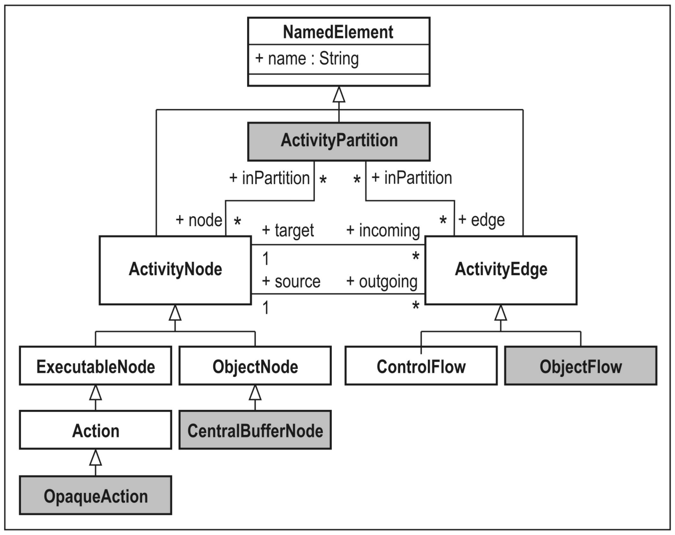

Fig. 1. UML metamodel excerpt used for activity diagram representation

IV. Initial Conceptual Database Model

The E-R (*Entity-Relationship*) notation [9], or one of its modifications, is traditionally used for conceptual database modeling, while the development of UML has seen an increasing use of the UML class diagram as well.

This section describes the class diagram as the initial conceptual database model and provides the transformation rules for automatic class diagram generation.

A. Class diagram as the initial conceptual database model

The target class diagram generator is to generate classes for all extracted business objects and business process participants. Each class is modeled by the `Class` UML metaclass.

Since the target class diagram is to represent the initial key-based conceptual model of a relational database, all generated classes will not contain operations, but only a set of `ownedAttributes`. Attributes may be class data members (e.g. attribute representing a primary key), as well as those based on associations with other classes. Each attribute is modeled by the `Property` metaclass.

Class associations are modeled by the `Association` metaclass. Each association has two `memberEnd` attributes (also modeled by the `Property` metaclass) representing the `source` and `target` association ends with the appropriate multiplicity. The related excerpt from the UML infrastructure [10] is shown in Fig. 2.

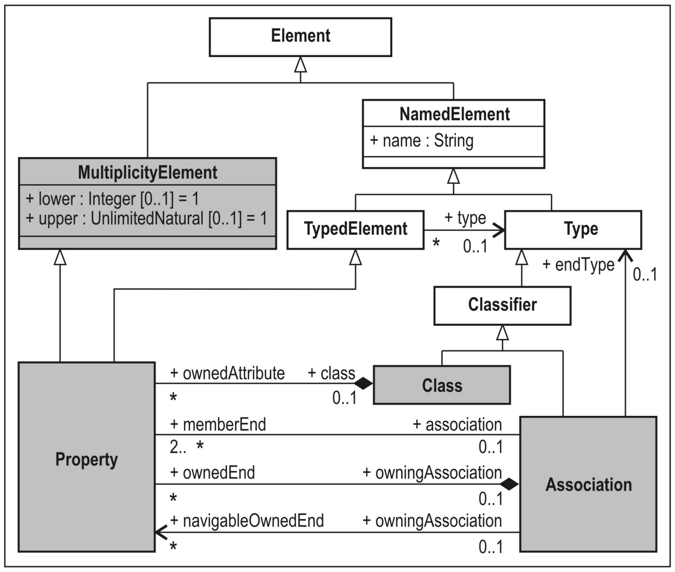

Fig. 2. UML metamodel excerpt for class diagram representation

B. ATL-based generator of the initial conceptual model

The ATL (*ATLAS Transformation Language*) is a hybrid language for specifying *model-to-model transformations* [11]. It follows the common MDE (*Model Driven Engineering*) transformation pattern, where the execution of a transformation program (*TP*) results in the automatic creation of a target model (*Mb*) from a source model (*Ma*). These three models, *Ma*, *TP* and *Mb*, conform to the respective metamodels (*MMa*, *MMt*, *MMb*) conforming to the same metametamodel *MMM*.

The automatic ATL-based generator of the initial conceptual database model is to take a detailed business activity diagram as the source model expressed in the XMI (*XML Metadata Interchange*) serialization format [12] and automatically generate the XMI representation of a class diagram as the target model. The source and target models both conform to the same metamodel (`"http://www.eclipse.org/uml2/3.0.0/UML"`), so we have endogenous transformations.

The ATL module, which implements the target generator, contains a number of attribute helpers and transformation rules. Some transformation rules are aimed at recognizing the activity diagram in the source model and generating the target model containing two packages. The first package contains necessary primitive types (in our case `pInteger` as primary key type). The second package (`cd`) will contain the XMI representation of the target class diagram. Due to space limitations, we don't provide all these rules, but only the main transformation rules whose execution result in populating the target class diagram with classes and their associations.

The target class diagram is populated with classes by applying two *matched transformation rules*. The first matched rule (Listing 1), according to [4] and [5], recognizes the business process participants represented by swimlanes (activity partitions) in the source activity diagram and generates the corresponding classes, i.e. each swimlane is mapped to the respective class of the same name in the target class diagram.

Listing 1. Matched rule for mapping business process participants

```
rule ActivityPartition
{
  from  s : uml!ActivityPartition
  to    d : uml!Class ( name <- s.name )
  do
  {
    thisModule.cd.packagedElement <- d;
    thisModule.addPrimaryKey(d);
  }
}
```

Additionally, a *called rule* named `addPrimaryKey` (Listing 2) adds an attribute named `id` representing the primary key in each generated class.

Listing 2. Called rule for adding primary key in generated class

```
rule addPrimaryKey (class : uml!Element)
{
  to
    pk : uml!Property
    (
      name <- 'id', type <- thisModule.pInteger
    )
  do
  {
    class.ownedAttribute <- pk;
  }
}
```

Similarly, the second matched rule `CentralBufferNode` (Listing 3), according to [2], [4] and [5], generates the respective classes for all recognized business objects, i.e. each business object represented as `CentralBufferNode` in the source activity diagram is mapped into respective class of the same name in the target class diagram.

Listing 3. Matched rule for mapping business objects

```
rule CentralBufferNode
{
  from  s : uml!CentralBufferNode
  to    d : uml!Class ( name <- s.name )
  do
  {
    thisModule.cd.packagedElement <- d;
    thisModule.addPrimaryKey(d);
  }
}
```

The relationships between business process participants and business objects are *agent-object* relationships [13] (the business actor buys a product, the worker issues a required document, etc.). As given in [4] and [5], multiplicity is most often that of type "1..*" (the business actor can file several applications, buy several products, etc.). Other types of multiplicity can often be classified under type "1..*" multiplicity (e.g. "1..1" is a special case of the "1..*" multiplicity).

Associations of the generated classes will be created through the execution of the third main matched rule (Listing 4), which exploits some attribute helpers decorating the source model before transformation execution. Since this transformation is more complex, we provide a more detailed explanation. The associations are to be generated based on object flows and activities. One of the object flow ends is always a business object. It is identified by the attribute helper `obj`. A class corresponding to that business object is the target association end. The source association end is always a class representing a swimlane. It is identified by the attribute helper named `ap`, based on the activity representing the other object flow end (attribute helper `act`). An association is to be generated with multiplicity "1" on the source end, and "*" on the target end. The association name corresponds to the activity performed on the object.

Listing 4. Transformation rule for generation of associations

```
helper context uml!ObjectFlow def:
  obj : uml!CentralBufferNode =
        uml!CentralBufferNode.allInstances()->
        select( n | self.source = n or
                    self.target = n )->first();
helper context uml!ObjectFlow def:
  act : uml!OpaqueAction =
        uml!OpaqueAction.allInstances()->
        select( a | self.source = a or
                    self.target = a )->first();
helper context uml!ObjectFlow def:
  ap : uml!ActivityPartition =
       self.act.inPartition->first();
rule ObjectFlow
{
  from s : uml!ObjectFlow ( not s.act.oclIsUndefined())
  to
    d : uml!Association
    (
      name <- s.act.name,
      ownedEnd <- Sequence {osrc, odst},
      navigableOwnedEnd <- odst
    ),
    osrc : uml!Property
    (
      name <- 'source',
      type <- thisModule.resolveTemp(s.ap,'d'),
      upperValue <- uvs, lowerValue <- lvs
    ),
    odst : uml!Property
    (
      name <- 'target',
      type <- thisModule.resolveTemp(s.obj,'d'),
      upperValue <- uvt, lowerValue <- lvt
    ),
    uvs : uml!LiteralUnlimitedNatural ( value <- 1 ),
    lvs : uml!LiteralInteger ( value <- 1 ),
    uvt : uml!LiteralUnlimitedNatural
            ( value <- '-1'.toInteger() ),
    lvt : uml!LiteralInteger
  do
  {
    thisModule.cd.packagedElement <- d;
  }
}
```

V. "Real-world" example

The implemented generator is applied to the business model of the visa issuance system. An activity diagram documenting the realization of the selected business use case named `VisaIssuance` (also used in [5]) is shown in Fig. 3 (up).

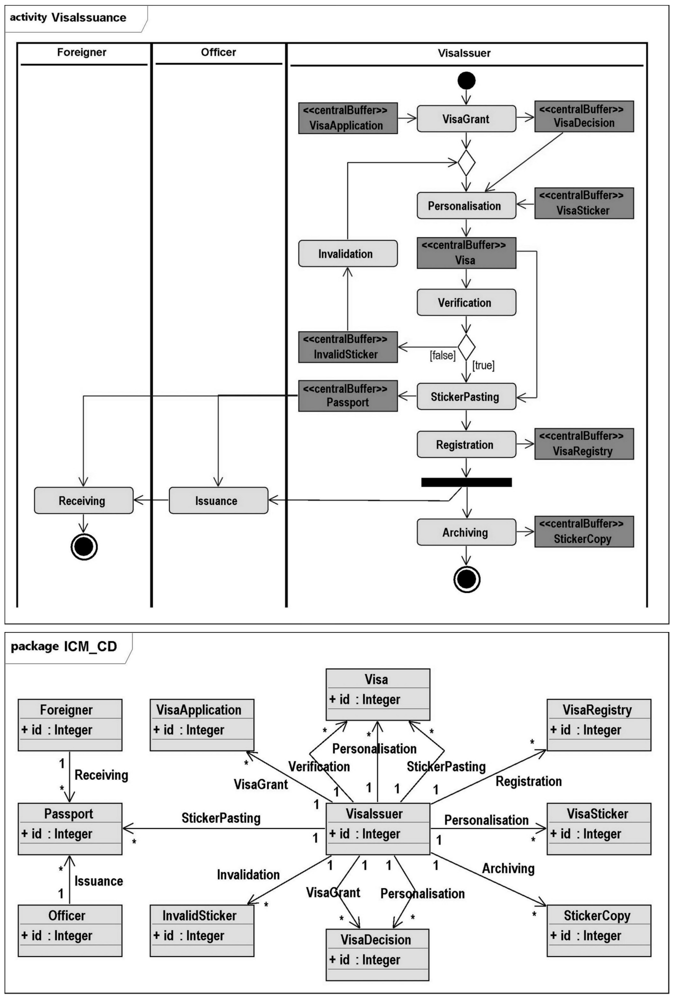

Fig. 3. Activity diagram for `VisaIssuance` business use case *(up)* and the corresponding automatically generated initial conceptual model *(down)*

The result of the generator's application (execution of the implemented ATL module) to the *.uml* file containing the XMI representation of the taken activity diagram is the *.uml* file containing the XMI representation of the target class diagram. Due to its size, we only provide the visualization result (Fig. 3 (down)) in the Topcased environment [14]. It shows that the implemented ATL module is able to generate:

(i) classes for all business process participants and business objects, as well (it is obvious that classes `Foreigner`, `Officer` and `VisaIssuer` are generated directly from the swimlanes, while the other classes are generated directly from the used business objects), and

(ii) associations of the created classes based on the activities performed by the respective participants on the respective objects. Besides the single class associations, the generator is also able to generate multiple associations, such as three associations of the `Visa` and `VisaIssuer` classes.

VI. Conclusion

This paper presents the ATL-based automatic generator of the initial conceptual key-based model of relational database based on the automatic extraction of business entities from the business model and the automatic generation of their relationships. The detailed UML business activity diagram is taken as the source model, while the target initial conceptual model is represented by the UML class diagram. The automatic generation of the target class diagram is driven by ATL transformation rules defined on the basis of UML metamodel which is common to both the source and target model.

The described approach and implemented generator imply that UML business activity diagrams, as a frequently used business process modeling notation, have (some) semantic capacity for automated initial conceptual database model design.

Given the fact that the shortcomings and limitations of the implemented generator mainly result from the insufficiently explored semantic potential of activity diagrams, further research will focus on the full identification and additional enhancement of the semantic capacity of detailed activity diagrams for automated database design, the complete definition of the transformation rules and the approach formalization.

References

[1] M. Kamimura, K. Inoue, A. Hasegawa, R. Kawabata, S. Kumagai, K. Itoh, "Integrated Diagrammatic Representations For Data Design In Collaborative Processes", *Journal of Integrated Design & Process Science*, vol. 7(4), pp. 35–49, 2003.

[2] J. Garcia Molina, M. Jose Ortin, B. Moros, J. Nicolas, and A. Toval, "Towards use case and conceptual models through business modeling", in *ER 2000*, LNCS, vol. 1920, A.H.F. Laender, S.W. Liddle, and V.C. Storey, Eds. Berlin Heidelberg: Springer, 2000, pp. 281–294.

[3] E. Suarez, M. Delgado, and E. Vidal, "Transformation of a process business model to domain model", *Proc. of WCE 2008 - World Congress on Engineering*, vol. 1, pp. 165–169, 2008.

[4] D. Brdjanin and S. Maric, "An example of use-case-driven conceptual design of relational database", *Proc. of EUROCON 2007*, pp. 538–545, 2007.

[5] D. Brdjanin, S. Maric, and D. Gunjic, "ADBdesign: An approach to automated initial conceptual database design based on business activity diagrams", in *ADBIS 2010*, LNCS, vol. 6295, B. Catania, M. Ivanovic, and B. Thalheim, Eds. Berlin Heidelberg: Springer, 2010, pp. 117–131.

[6] R. Ko, S. Lee, and E. Lee, "Business process management (BPM) standards: A survey", *Business Process Management Journal*, vol. 15 (5), pp. 744–791, 2009.

[7] D. Brdjanin and S. Maric, "UML-business profile-based business modeling in iterative-incremental software development", *Proc. of EUROCON 2005*, pp. 1263–1266, 2005.

[8] Object Management Group (OMG), *Unified Modeling Language: Superstructure*, v2.2. OMG, 2009.

[9] P. Chen, "The Entity-Relationship Model: Toward a Unified View of Data", *ACM ToDS*, vol. 1 (1), pp. 9–36, 1976.

[10] Object Management Group (OMG), *Unified Modeling Language: Infrastructure*, v2.2. OMG, 2009.

[11] F. Jouault, F. Allilaire, J. Bezivin, and I. Kurtev, "ATL: A model transformation tool", *Science of Computer Programming*, vol. 72 (1-2), pp. 31–39, 2008.

[12] Object Management Group (OMG), *MOF2.0/XMI Mapping*, v2.1.1. OMG, 2007.

[13] V. Storey, "Understanding semantic relationships", *VLDB Journal*, vol. 2 (4), pp. 455–488, 1993.

[14] TOPCASED Project, *Toolkit in OPen-source for Critical Application & SystEms Development*, v3.2.0. http://www.topcased.org.