EB^{3}TG : Un outil de génération de transactions de base de données relationnelle pour EB^3

Panawé Batanado[1], Frédéric Gervais[1,2], Marc Frappier[1] et Régine Laleau[3]

[1]GRIL, Université de Sherbrooke,
2500, Boulevard de l'Université
Sherbrooke (Québec) J1K 2R1, Canada
{panawe.batanado,marc.frappier}@usherbrooke.ca

[2]CEDRIC, CNAM-IIE,
18 Allée Jean Rostand, 91025 Evry, France
frederic.gervais@usherbrooke.ca

[3]LACL, Université Paris 12,
IUT Fontainebleau, Département Informatique,
Route Forestière Hurtault, 77300 Fontainebleau, France
laleau@univ-paris12.fr

Résumé EB^3 est un langage formel créé pour la spécification des systèmes d'information. Les attributs des types d'entité et des associations du système sont évalués en EB^3 grâce à des fonctions récursives définies sur les traces valides du système. Nous présentons EB^{3}TG, un outil capable de générer automatiquement des programmes Java/SQL qui exécutent des transactions de base de données relationnelle qui correspondent aux définitions d'attributs EB^3.

Mots-clé Système d'information, EB^3, trace, transaction, Java, SQL

1 Introduction

Le langage EB^3 [FSD03] est un langage formel qui a été défini dans le but de spécifier des systèmes d'information (SI). Une spécification EB^3 comprend quatre parties : i) un diagramme entité-relation (ER) ; ii) une expression de processus ; iii) un ensemble de règles d'entrée-sortie ; iv) un ensemble de définitions d'attributs. Le diagramme ER représente les différents types d'entité [1] et associations du système. Le diagramme reprend les notations graphiques d'UML. Les séquences possibles des événements du SI, appelées traces valides, sont décrites par une expression de processus. Lorsqu'une action EB^3 requiert une sortie, une règle d'entrée-sortie associe cette action à une fonction qui permet d'évaluer la sortie en fonction de la trace courante du système. Enfin, les attributs des types d'entité et associations du SI sont spécifiés à l'aide de fonctions récursives sur les traces valides du système.

Le projet APIS [FFLR02] a pour objectif de générer des SI à partir de spécifications EB^3. La figure 1 représente les différentes composantes d'APIS. Un premier outil, appelé DCI-Web, permet de générer une interface Web [Ter05] à partir d'une spécification du GUI (*graphical user interface*). Pour mettre à jour ou pour interroger le système, l'utilisateur génère un événement à travers cette interface. L'événement est ensuite analysé par EB^{3}PAI [FF02], un interpréteur des expressions de processus EB^3. S'il est considéré comme valide, alors il est exécuté ; sinon,

[1]Terme utilisé en EB^3 pour désigner une classe. Une entité est l'instance d'un type d'entité.

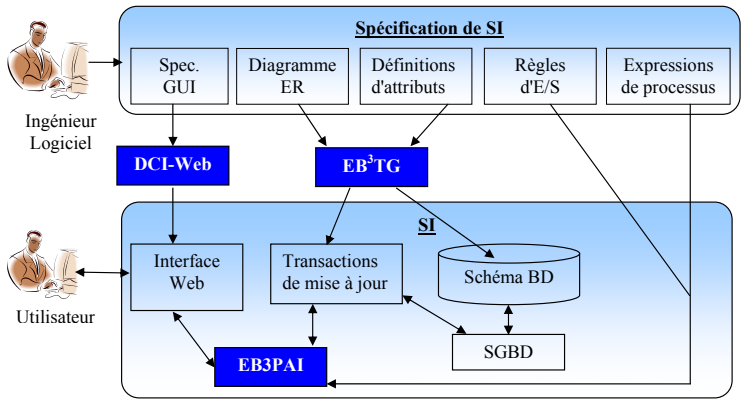

FIG. 1 – Composantes du projet APIS

un message d'erreur est envoyé à l'utilisateur. Nous présentons un outil complémentaire, appelé EB^{3}TG, qui permet de générer automatiquement des programmes Java qui exécutent des transactions de base de données (BD) relationnelle qui correspondent à la spécification des attributs du SI en EB^3. Les transactions obtenues peuvent ainsi être utilisées en combinaison avec l'outil EB^{3}PAI afin de mettre à jour ou d'interroger la BD lorsque les événements correspondants sont considérés comme valides par l'interprétation des expressions de processus EB^3. EB^{3}TG est compatible avec différents systèmes de gestion de BD (SGBD).

2 Présentation de l'outil EB^{3}TG

Notre outil a été implanté en Java. Le code source comprend 50 classes et 625 méthodes, pour un total de 20000 lignes de code. Les fonctionnalités de l'outil sont présentées dans la figure 2. Les algorithmes et les choix d'implantation de l'outil sont détaillés dans [GBFL06].

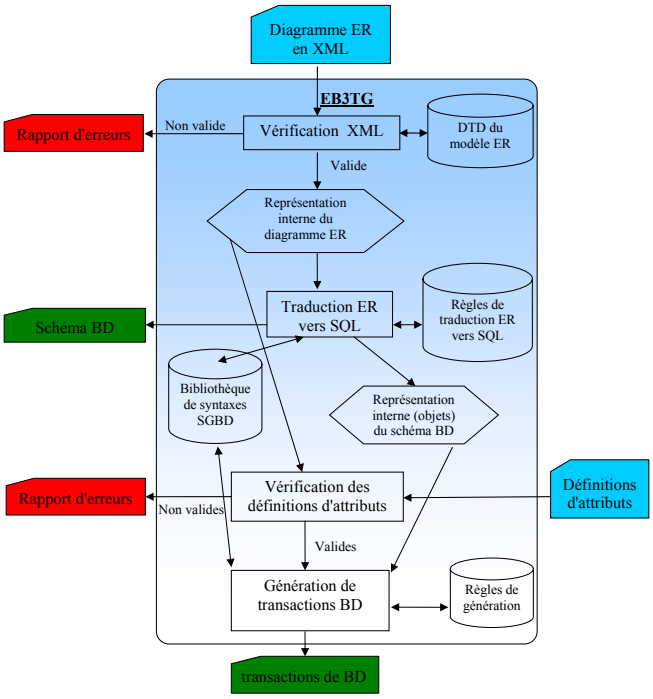

FIG. 2 – Architecture fonctionnelle de l'outil EB^{3}TG

Pour illustrer l'application de notre outil, nous utilisons un exemple de gestion de bibliothèque. Le système doit gérer des emprunts de livres par des membres.

À partir d'une représentation XML du diagramme ER, EB^{3}TG fait une vérification par rapport à la DTD (*Document Type Definition*) du modèle ER et retourne des messages d'erreur en cas de problèmes. La figure 3 montre par exemple la représentation XML du diagramme ER d'une bibliothèque, avec deux types d'entités, *book* et *member*, et une association, *loan*.

```
<entity name="book"  type="strong"  label="books entity type">
 <attribute name="bookKey" type="numeric" size="5" scale="2" key="true"/>
 <attribute name="title" type="varchar" size="20"/>
</entity>
<entity name="member" type="strong" label="members entity type" >
 <attribute name="memberKey" type="numeric" size="5"  key="true"/>
 <attribute name="nbLoans" type="numeric" size="5" null="false"/>
 <attribute name="loanDuration" type="numeric" size="3" null="false"/>
</entity>
<relation name="loan" label="Loan Relation" >
 <attribute name="dueDate" type="date"/>
 <participant refEntity="member" participation="partial"cardinality="1"
              role="borrower"/>
 <participant refEntity="book" participation="partial" cardinality="N"/>
</relation>
```

Fig. 3 – Exemple de représentation XML de diagramme ER

Lorsque la représentation XML est conforme à la DTD, l'outil génère un schéma de BD. L'algorithme de traduction d'un diagramme ER en un schéma de BD relationnelle est présenté dans [GFLB05]. Les énoncés SQL de création des tables et des contraintes d'intégrité sont générés en fonction du SGBD choisi par l'utilisateur. L'outil supporte actuellement Oracle, PostgreSQL et MySQL. Une bibliothèque de syntaxes permet de déterminer les mots réservés et les syntaxes propres à chacun de ces SGBD.

Le schéma de BD généré pour l'exemple de la bibliothèque est présenté dans la figure 4. Une table est créée pour chaque type d'entité et pour l'association. Des contraintes référentielles sont également générées automatiquement pour éviter des problèmes de référencement mutuel entre tables. Le SGBD choisi dans ce cas est Oracle.

```
CREATE TABLE book (
bookKey     numeric(5,2),
title       varchar(20),
CONSTRAINT PKbook PRIMARY KEY(bookKey));

CREATE TABLE member (
memberKey         numeric(5),
nbLoans           numeric(5)   NOT NULL,
loanDuration      numeric(3)   NOT NULL,
CONSTRAINT PKmember PRIMARY KEY(memberKey));

CREATE TABLE loan (
borrower    numeric(5),
bookKey     numeric(5,2),
dueDate     date,
CONSTRAINT PKloan PRIMARY KEY(bookKey));

ALTER TABLE loan ADD CONSTRAINT FKloan_member FOREIGN KEY
(borrower) REFERENCES member (memberKey) INITIALLY DEFERRED;

ALTER TABLE loan ADD CONSTRAINT FKloan_book FOREIGN KEY
(bookKey) REFERENCES book (bookKey) INITIALLY DEFERRED;
```

Fig. 4 – Exemple de schéma de BD généré

La figure 5 présente quelques exemples de définitions d'attributs. Il s'agit en fait des attributs affectés par l'exécution de l'action *Lend*, qui consiste à emprunter un livre. L'attribut

book.borrower représente l'emprunteur d'un livre, *member.nbLoans*, le nombre de prêts d'un membre et *loan.dueDate*, la date de retour d'un prêt.

```
member.nbLoans(s,mId)==
  match last(s) with
  NULL -> NULL,
  ...
  Lend(_,mId) -> member.nbLoans(front(s),mId)+1,
  ...
  _  -> member.nbLoans(front(s),mId);

loan.loan(s)==
  match last(s) with
  NULL -> {},
  ...
  Lend(bId,_) -> loan.loan(front(s))\/{bId},
  ...
  _  -> loan.loan(front(s));
```

```
loan.dueDate(s,bId)==
  match last(s) with
  NULL -> NULL,
  ...
  Lend(bId,mId) -> CURRENTDATE +
      member.loanDuration(front(s), mId),
  ...
  _ ->  loan.dueDate(front(s),bId);

book.borrower(s,bId)==
  match last(s) with
  NULL -> NULL,
  ...
  Lend(bId, mId) ->  mId,
  ...
  _ -> book.borrower(front(s),bId);
```

FIG. 5 – Exemple de définitions d'attributs EB^3

L'outil EB^{3}TG vérifie que les définitions d'attributs sont cohérentes par rapport au diagramme ER. En cas d'erreur, l'outil génère un message. Par exemple, la figure 6 montre deux messages d'erreur de syntaxe dans les définitions d'attributs. Le premier concerne l'absence du mot clé *match* à la colonne 9, ligne 40, du fichier *bookStore.txt* contenant les définitions d'attributs de cet exemple. Le deuxième message d'erreur signale que le nombre de paramètres dans l'appel récursif de *member.loanDuration* ne correspond pas au nombre de paramètres prévu dans sa définition. Le message permet également de situer cette erreur dans les définitions d'attributs : il s'agit de l'effet de l'action *Lend* dans la définition de l'attribut *loan.dueDate*.

```
1 bookStore.txt:40:9: expecting "with", found 'NULL'
2
3 >>Error in :
4   Attribute definition : loan.dueDate
5   Action : Lend(_,mId)
6   Cause : Invalid number of parameters in attribute recursive call
7   Clues : The attribute recursive call 'member.loanDuration'
8           must have exactly 2 parameters
```

FIG. 6 – Exemple de messages d'erreur

EB^{3}TG génère ensuite les programmes Java qui exécutent des transactions de BD relationnelle qui correspondent aux définitions d'attributs, en fonction du schéma de BD généré. Les règles de synthèse des transactions sont détaillées dans [GFLB05]. Par exemple, la méthode Java générée pour l'action *Lend* est présentée dans la figure 7. On y retrouve la structure typique d'une transaction : une série d'opérations est suivie d'un *commit()*, et en cas d'erreur, un *rollback()* est exécuté.

3 Conclusion

Nous avons présenté l'outil EB^{3}TG qui permet de générer automatiquement des programmes en Java, qui exécutent des transactions de BD relationnelle, à partir de définitions d'attributs EB^3. La motivation du projet APIS, et de l'outil EB^{3}TG en particulier, est de libérer le programmeur des détails d'implantation des transactions, pour qu'il se concentre plutôt sur les phases d'analyse et de spécification. L'objectif est maintenant d'intégrer l'outil EB^{3}TG aux autres composantes d'APIS.

```
1 public static void Lend(int bId,int mId){
2   try {
3    ResultSet rset0 = connection.createStatement().executeQuery(
4     "SELECT A.nbLoans "+
5     "FROM member A "+
7     "WHERE A.memberKey = "+ mId+ " ");
8    String var0 = ((rset0.next())?rset0.getDouble(1)+"":"null");
9
10   ResultSet rset1 = connection.createStatement().executeQuery(
11    "SELECT A.loanDuration "+
12    "FROM member A "+
13    "WHERE A.memberKey = "+ mId+ " ");
14   String var1 = ((rset1.next())?rset1.getDouble(1)+"":"null");
15
16   connection.createStatement().executeUpdate("UPDATE loan SET "+
17    "borrower = "+mId+" "+
18    "WHERE bookKey = "+ bId +" ");
19
20   int var2 = connection.createStatement().executeUpdate(
21    "UPDATE loan SET "+
22    "dueDate = SYSDATE+"+var1+" "+
23    "WHERE bookKey = "+ bId +" ");
24
25   if( var2==0 ){
26     connection.createStatement().executeUpdate(
27     "INSERT INTO loan ( bookKey,dueDate) "+
28     " VALUES ( "+ bId +",SYSDATE+"+var1+")");
29   }
30
31   connection.createStatement().executeUpdate("UPDATE member SET "+
32     "nbLoans = "+var0+"+1 "+
33     "WHERE memberKey = "+ mId +" ");
34
35   connection.commit();
36  } catch ( Exception e ) {
37    try{
38       connection.rollback();
39    } catch (SQLException s){
40       System.err.println(s.getMessage());
41      }
42    System.err.println(e.getMessage());
43   }
44 }
```

Fig. 7 – Exemple de méthode implantant une transaction

Références

[FF02] B. Fraikin et M. Frappier. EB3PAI : An interpreter for the EB^3 specification language. In *15th Intern. Conf. on Software and Systems Engineering and their Applications (ICSSEA 2002)*, Paris, France, 3-5 Décembre 2002. CMSL.

[FFLR02] M. Frappier, B. Fraikin, R. Laleau, et M. Richard. APIS - Automatic production of information systems. In *AAAI Spring Symposium*, pages 17–24, Stanford, USA, 25-27 Mars 2002. AAAI Press.

[FSD03] M. Frappier et R. St-Denis. EB^3 : An entity-based black-box specification method for information systems. *Software and Systems Modeling*, 2(2) :134–149, Juillet 2003.

[GBFL06] F. Gervais, P. Batanado, M. Frappier, et R. Laleau. Génération automatique de transactions de base de données relationnelle à partir de définitions d'attributs EB^3. In *Atelier Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL 2006)*, Paris, France, 15-17 Mars 2006.

[GFLB05] F. Gervais, M. Frappier, R. Laleau, et P. Batanado. *EB^3 attribute definitions : Formal language and application.* Rapport technique n. 700, CEDRIC, Paris, France, Février 2005.

[Ter05] J.-G. Terrillon. Description comportementale d'interfaces Web. Mémoire de maîtrise, Département d'informatique, Université de Sherbrooke, 2005.