****

**Universitatea „Politehnica” din Bucureşti**

**Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei**

**ATLAS**

**Transformation Language**

Studenti: Dobre Nicoleta

Diaconescu Oana

Grupa: 441A

Bucuresti, 2016

Cuprins

1. Introducere – Diaconescu Oana

2. Modele si Transformari de Model - Diaconescu Oana

3. Limbajul de Transformare ATLAS - Dobre Nicoleta

*3.1 Studiu de caz: Transformarea modelelor clasa in modele relationale*

*3.2 Structura generala a definitiilor transformarii*

*3.3 Ajutoarele*

*3.4 Reguli de transformare*

*3.5 Proprietati imperative ale ATL*

*3.6 Executarea definitiilor transformarilor*

4. Utilizarea ATL pentru Verificarea Modelelor - Diaconescu Oana

*4.1* *Reprezentarea ATL a Constrangerilor*

*4.2 Analiza solutiei*

5. Instrumente de dezvoltare ATL - Dobre Nicoleta

*5.1 Motorul*

*5.2 Editarea*

*5.3 Depanare*

6. Concluzii - Dobre Nicoleta

7. Bibliografie

**1.Introducere [1]**

Transformarea de model joaca un rol important in abordarea MDE (Model Driven Engineering). Definirea dezvoltarilor transformarilor de model se asteapta a deveni un task comun in dezvoltarea de software. Inginerii software ar trebui sustinuti in realizarea acestui task de instrumente si tehnici MDE mature in acelasi mod in care sunt in acest moment sustinuti de clasicele modele IDE, compilatoare si deuggere in munca lor de programare de zi cu zi.

O modalitate pentru a pune la dispozitie un astfel de suport este dezvoltarea unor limbaje specifice proiectate sa rezolve taskuri de transformare de model comune. Intr-adevar, aceasta este noua abordare a comunitatii de cercetare si a industriei software, avand ca rezultat propunerea unui numar de limbaje de transformari de model.

**ATL** (ATL Transformation Language) este un limbaj de tranformare a modelelor si set de instrumente dezvoltat si intretinut de OBEO si AtlanMod. Acesta este un limbaj hibrid, punand la dipozitie un mix de constructii declarative si imperative ce a fost initiat de echipa AtlanMod (ATLAS Group). In domeniul MDE (Model-Driven Engineering), ATL ofera modalitati de a produce un set de modele tinta dintr-un set de modele sursa.

Lansat sub termenii Eclipse Public Licence, ATL este o componenta M2M (Eclipse), in interiorul EMP (Eclipse Modelin Project).

O masina virtuala orientata pe transformari de model a fost definita si implementata pentru a oferi suport pentru ATL, mentinand un anumit nivel de flexibilitate. De fapt, ATL devine executabila pentru ca pur si simplu exista o transformare din meta-modelul sau la bytecode-ul masinii virtuale.

Extinderea ATL este, asadar, o problema de definire a semanticii facilitatilor de executare intr-un nou limbaj: actiuni simple pe modele (creari de elemente si asignari de proprietati).

Un program ATL (aici *T1.atl*) va avea nevoie de un **Ma.xmi** ca data de intrare si va produce un model **Mb.xmi** ca iesire. Ambele modele pot fi descrise in standardul OMG XMI. Modelul Ma corespunde metamodelului **Mma.km3,** iar modelul Mb corespunde metamodelului **Mmb.km3.** Notatia KM3 este un limbaj simplu si neutru de specificatie pentru metamodele.

Programul ATL in sine (T1.atl) este de asemenea un model, deci corespunde unui metalodel (metamodelul ATL) care nu este prezentat aici.

Un program ATL este format din header, un set de functii fara efecte secundare numite **“helpers”** si un set de **reguli**.

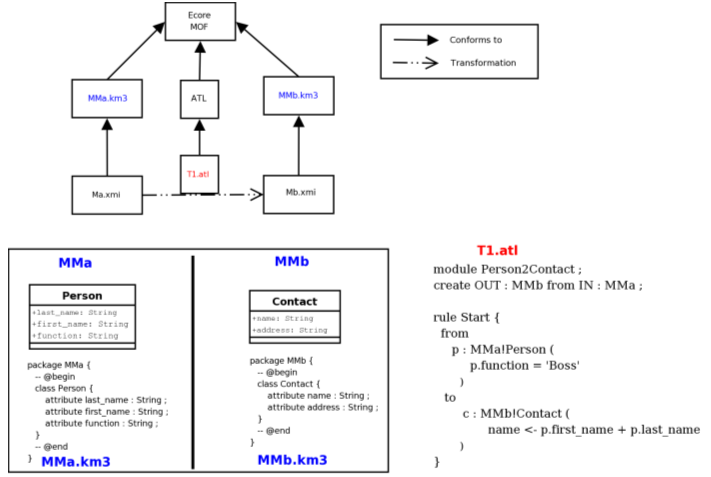


Figura 1

E disponibil si un plug-in open-source cu un set de tool-uri de dezvoltare ATL de la GMT Eclipse Modelling Project (EMP) ce implementeaza limbajul de transformare ATL care este inspirat din limbajul MOF QVT (Query/View/Transormation).

**2. Modele si Transformari de Model [2]**

In MDE, modelele sunt considerate ca fiind conceptul de unificare in ingineria IT. In mod traditional, modelele au fost folosite ca scheme initiale de proiectare menite sa comunice ideile printre dezvoltatori. MDE promoveaza modelele ca artefacte primare ce conduct intreg procesul de dezvoltare. Notiunea de model merge dincolo de viziunea limitata se diagrama semi-formala ce cere definiri mult mai precise so limbaje de modelare.

In comunitatea MDE se folosesc de ceva timp termenii de *model, metamodel* si *metametamodel.* Un model este reprezentarea unui sistem. Acesta surprinde niste caracterisitici ale sistemului si ofera informatii despre el. In MDE suntem interesati de modele exprimate in limbaje precise. Fundamentul conceptual al unui limbaj de modelare, cand este exprimat ca model, este numit *metamodel*. Adesea, metamodelele sunt considerate definitii ale sintaxei abstracte a limbajelor de modelare. Relatia dintre un model exprimat intr-un limbaj si metamodelul acestui limbaj este numit *conformsTo.* Metamodelele sunt la randul lor exprimate intr-un limbaj numit *metamodeling language.* Fundamentul sau conceptual este surprins intr-un model numit *metametamodel.* Modelele, metamodelele si metametamodelele formeaza o arhitectura tri-nivel cu nivelele denumite M1, M2 si respectiv M3.

Principiile MDE pot di implementate in cateva standarde. De exemplu, OMG propune un metamodel standard numit Meta Object Facility (MOF). Un exemplu de metamodel in contextul standardelor OMG este metamodelul UML,

Un proces de dezvoltare MDE este o serie de transformari de modele. In mod normal, transformarile sunt pasi de rafinare ale modelelor ce scad nivelul de abstractizare , dar si alte forme pot fi gasite in lanturile de transformari. Scopul este acela de a produce un model ce contine destule detalii pentru generarea automata de cod executabil. Alte scenarii in MDE sunt refactorizare, inginerie inversă, traducere de date între surse de date eterogene, etc.

In general, modelele de transformare pot fi implementate in diferite moduri, de exemplu, folosind un limbaj de programare de uz general.

Transformarile de model in MDE urmeaza in tipar comun, cunoscut ca *modelul de transformari de model* reprezentat in figura 2 . *Tab* este un proram de transformare din a carui executie rezulta creatia automata a modelului *Mb*  din *Ma.* Aceste trei entitati (*Tab, Ma* si *Mb*)sunt toate modele conforme metamodelelor *Mmt, Mmb,* respectiv *Mma. Mmt*  corespunde sintaxei abstracte a limbajului de transformare.

Cele trei metamodele corespund unui metametamodel numit *MMM.* In contextul standardului OMG, metametamodelul *MMM*  este MOF-ul.

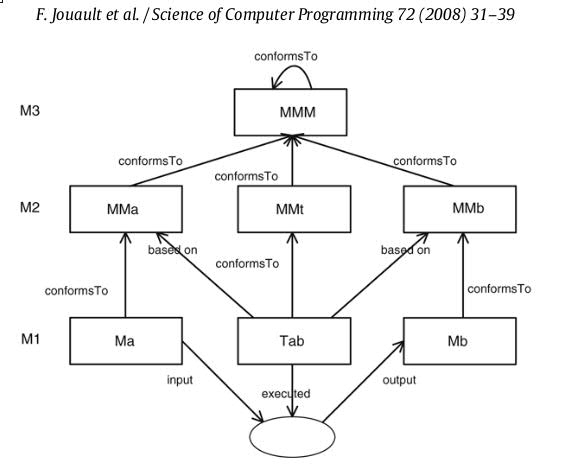


Figura 2 – Tiparul Transformarilor de Model

**3. Limbajul de Transformare ATLAS [4]**

ATL este aplicat in context cu tiparul transformatei aratat in Figura 3. In acest tipar un model sursa *Ma* este transformat intr-un model tinta *Mb* conform unei definitii de transformare mma2mmb.atl scrisa in limbaj ATL. Definita de transformare este un model in conformitate cu metamodelul ATL. Toate metodele sunt conforme cu MOF.

ATL este un limbaj de transformare hibrid. Contine un mix de constructii declarative si imperative. Stilul declarativ al specificatiilor transformatei prezinta un numar de avantaje. De regula se bazeaza pe specificarea relatiilor dintre tiparele sursa si tinta si, prin urmare, tinde a fi mai aproape de modul in care un programator percepe intuitiv tranformarea. Acest stil presupune codarea acestor relatii si ascunde detaliile legate de selectia elementelor sursa, aranjare, etc. Drept urmare poate ascunde algoritmi de transformare complecsi in spatele unei simple sintaxe.

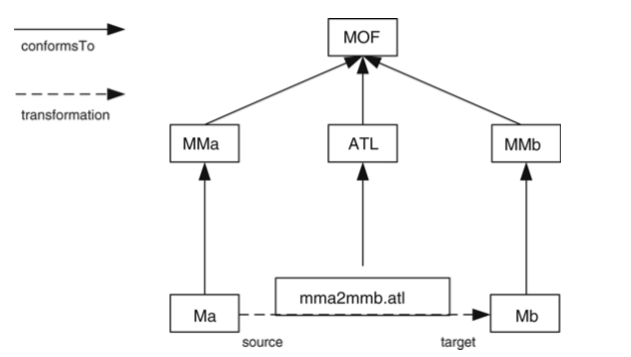


Figura 3

Transformarile ATL sunt unidirectionare, operand doar pe modele sursa read-only si modele tinta write-only. In timpul executiei unei transformari modelul sursa poate fi navigat, dar nu se pot produce modificari in cadrul acestuia. Modelul tinta nu poate fi navigat. O transformare bidirectionala este implementata drept o pereche de transformari, una pentru fiecare directie.

Modelele sursa si tinta pentru ATL pot fi exprimate in format de serializare XMI OMG. Metamodelele sursa si tinta pot fi de asemenea exprimate in notatii XMI sau mai convenabil in notatii KM3.

*3.1 Studiu de caz: Transformarea modelelor clasa in modele relationale*

Studiul de caz necesita programul de transformare ce transforma modele simple de transformari de clasa in modele relationale. Metamodelele sursa si tinta sunt reprezentate in Figura 4, respectiv Figura 5.

Intr-un model clasa, clasele au 0 sau mai multe atribute si pot specializa alte clase. Tipul de atribute este fie un tip de date primitive fie o clasa.

Un model relational contine un set de tabele. Fiecare tabel are 0 sau mai multe coloane. Cateva dintre coloane sunt chei. Fiecare coloana are un tip primitiv.

Ne vom concentra pe doua reguli de transformare descriind informal inm urmatorul fel:

* Pentru fiecare clasa din modelul sursa un tabel in modelul tinta este creat cu acelasi nume ca si numele clasei.
* Pentru fiecare valoare atribuita este creata o coloane in modelul tinta.

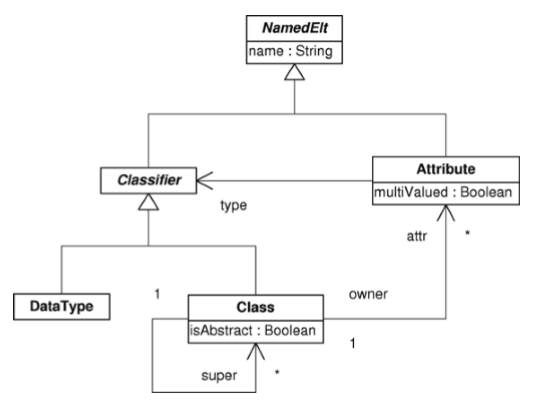


Figura 4

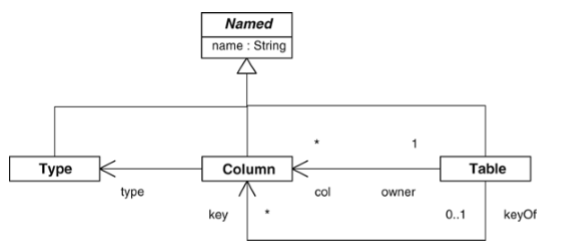


Figura 5

*3.2 Structura generala a definitiilor transformarii*

Definitiile transformarii in module ATL de tip formular. Un modul contine o sectiune de tip header obligatorie, o sectiune import si un numar ajutoare si reguli de tranformare. Sectiunile de tip header dau un nume modulelor de transformare si declara modelele sursa si tinta. Acesta este un exemplu de sectiune de tip header:



Sectiunea de tip header incepe cu cuvantul cheie *module* urmat de numele modulului. Apoi, modelele sursa si tinta sunt declarate ca variabile tiparite de metamodelele lor. Cuvantul cheie *create* indica modelele tinta. Cuvantul cheie *from* indica modelele sursa. In exemplul nostru modelul tinta este legat de cariabila OUT si este cread din modelul sursa IN. Modelele sursa si tinta sunt in conformitate cu metamodelele *Class* si *Relational*. In general,pot fi enumerate in sectiunea Header mai mult de un model sursa si tinta.

Ajutoarele si regulile de tranformare sunt constructii folosite pentru a specifica functionalitatea transformarii.

*3.3 Ajutoarele*

Termenul de ajutoare ( Helper) provine din Sectiunea 7.4.4, p11 din specificatiile OCL, ce definesc 2 tipuri de ajutoare, ajutoare operationale si ajutoare de atribute.

In ATL, un ajutoar poate specificat doar intr-o scriere OCP sau intr-o scriere a metamodelului sursa, datorita faptului ca modelele tinta nu sunt navigabile. Ajutoarele operationale definesc operatii in legatura cu elementul modelelor sau in legatura cu un modul. Ele pot avea parametri de intrare si pot folosi recursivitatea. Ajutoarele de atribute sunt folosite pentru a asocia valori read-only unor elemente ale modelelor sursa. Similat cu ajutoarele operationale acestea au un nume un context si o forma de scriere. Diferenta este ca nu au parametrii de intrare. Valorile lor trebuie specificate prin intermediul unor expresii OCL. Ele pot fi definite si cu ajutorul recursivitatii.

Ajutoarele de atribute pot fi vazute ca un mod de a decora modelele sursa inainte de executarea transformarii.



*3.4 Reguli de transformare*

Regulile de transformare sunt constructii obisnuite in ATL folosite pentru a exprima logica transformarii. Regulile ATL pot fi specificate fie intr-un stil imperativ, fie intr-unul imperativ.

**Regulile pereche**

Regulile ATL declarative se numesc reguli pereche . O regula pereche este compusa dintr-un tipar sursa si unul tinta. Regula tiparului sursa specifica un set de tipuiri sursa si o garda ( o expresie de tip OCL Boolean).

Tiparul tinta este compus dintr-un set de elemente. Fiecare element specifica un tip de tinta si un set de legaturi. Prin legatura intelegem o proprietate a tipului de tinta. Si specifica o initializare a expresiei pentru valorea proprietatii.

**Executia semanticii** pentru reguli pereche. Regulile pereche sunt executate peste perechile din tiparul sursei lor. Pentru orice pereche sunt create in modelul tinta, elemente tinta ale tipurilor specificate, iar proprietatile lor sunt instantiate folosind legaturile lor.

Executand o regula pentru o pereche se creaza in stucturile interne ale motorului de transformare un link aditional pentru urmarire. Aceste linkuri au 3 componente: regula, perechea si nou createle elemente tinta.

Initializarea proprietatii foloseste un algoritm de rezolvare numit Algoritm de rezolvare ATL. Algoritmul se foloseste pe valorile expresiilor de legatura. Daca tipul valorii este primitiv valoarea este atribuita proprietatii corespunzatoare. Daca acest tip este de forma metamodel atunci exista doua posibilitati:

* Daca valoarea este un element tinta acesta este atribuit proprietatii.
* Daca valoarea este un element sursa aceasta este transformata in element tinta folosind linkurile interne de urmarire, ulterior atribuindu-se valoarea proprietatii.

Tipuri de reguli pereche:

- Reguli standard – aplicare odata pentru fiecare pereche. Pot fi gasite in modelele sursa.

- Reguli lenese – sunt actionate de alte reguli. Ele sunt aplicate pentru o singura pereche de cate ori se face referire catre aceasta de alte reguli, de fiecare data producand un alt set de elemente tinta. Ele sunt indicate de cuvantul cheie lazy.

- Reguli lenese unice- Sunt actionate de alte reguli. Ele sunt aplicate o singura data pentru o pereche. Daca o regula lenesa unica este actionata mai tarzie pentru aceasi pereche sunt folosite aceleasi elemente tinta create anterior. Sunt indicate de cuvintele cheie unique lazy

**Mostenirea regulilor**

In ATL mostenirea regulilor poate fi folosita ca un mecanism de refolosire al codurilor sau ca un mecanism pentru specificarea regulilor polimorfe.

O regula( numita subregula) poate mosteni de la o alta regula (numita regula parinte). Tipurile tiparelor sursei in regula parinte poate fi inlocuita cu subtipuri din tiparul sursei subregulii. Garda unei subreguli formeaza o conjunctie cu garda regulii parinte.

Un tipar tinta al subregulei extinde tiparul tinta parinte folosind o combinatie intre urmatoarele modificari: adaugare legaturi, inlocuire legaturi , adaugare de noi elemente tinta si scrierea de tipuri tinta.

*3.5 Proprietati imperative ale ATL*

Cateodata pot fi necesari algoritmi complecsi de transformare si poate fi dificil sa specificam o solutie pur declarativa pentru acestia. ATL are o parte imperativa bazata pe doua mari constructii:

* Reguli apelate. O regula apelata este in principiu o procedura: este invocata de nume si primeste argumente
* Bloc de actiune- Un bloc de actiune este o secventa de declaratii imperative ce poate fi folositin loc sau in combinatie cu tiparul tinta in reguli pereche sau apelate.

*3.6 Executarea definitiilor transformarilor*

Executia incepe prin invocarea unei reguli optionale apelate numita punct de intrare. Aceasta regula poate invoca si alte reguli. Algoritmul executa apoi alte reguli pereche standard. Imperechearea regulilor si aplicatia lor sunt impartite in doua faze. In prima faza toate tiparele regulilor sunt imperecheate cu modelele sursei corespunzatoare. Pentru fiecare pereche sunt create noi elemente tinta, creindu-se si link-urile de urmarire corespunzatoare. In ce-a de a doua faza suntexecutate toate legaturile pentru elementele tinta create anterior.

Algoritmul nu presupune nici o ordine in imperecherea regulilor, creerea elementelor tinta pentru o pereche sau pentru instantierea elementelor tinta.

**4. Utilizarea ATL pentru Verificarea [5]**

*4.1* *Reprezentarea ATL a Constrangerilor*

Motorul ATL poate fi folosit numai pentru a verifica constrangeri daca acestea sunt exprimate exact asa cum a fost specificate. Algirtmul de creare a unui program dintr-un set de constrangeri exte urmatorul: Pentru fiecare constrangere, se creaza o regula de transformate ATL in asa fel incat:

* tipul de model sursa al regulii este dat de contextul constrangerii
* garda regulii contine negarea expresiei booleane asociata constrangerii;
* modelul tinta al regulii specifica un singur tip: *Problema,* care trebuie creata in pereche.

Va fi facuta o initializare folosind trei legaturi pentru: severitatea functiei invariante, o descriere a problemei si locatia acesteia. Din moment ce elementul tinta este initializat din expresii OCL ce navigeaza prin modelul sursa, implementarea unei descrieri ca un sir fiind directa.

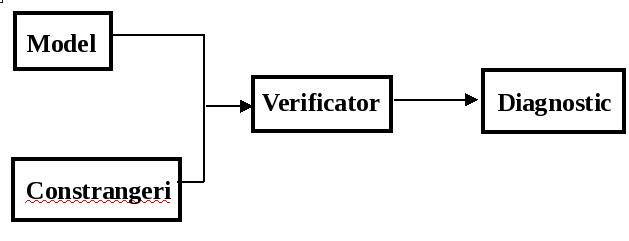
*4.2 Analiza solutiei*

***Figura 6***  prezinta o structura generala pentru verificarea de constrangeri. Un *Verificator* verifica *Constrangerile* pe un *Model*, ceea ce rezulta intr-un *Diagnostic.* Vom nota cu *V* operatia implementata de Verificator, cu *M Modelul*  ce trebuie verificat, cu *C* *Constrangerile* si cu *D Diagnosticul* rezultat.

Atunci vom putea scrie *D=V(M,C).*

Tocmai am prezentat o solutie integrand constrangerile si verificatorul in aceeasi entitate: un program ATL. Daca notam acest program cu *P,* aunci avem *D=P(M).* Comparand cele doua expresii ale lui *D*  observam ca *P* ii corespunde lui *V* in care *C* este deja evaluat. Aceasta este foarte asemanatoare cu operatia de prlucrare a limbajelor functionale. De fapt, o versiune prelucrata a lui *V* va fi notata *Vc(M)* precum *D=Vc(M).*

Este foarte probabil ca un set de constrangeri date *C* sa fie folosit pentru a verifica cateva modele conforme cu metamodelul corespunzator. Asta se intampla in practica atunci cand compilatorul ATL este folosit pentru a compila constrangeri.



*Figura 6 - Verificarea Constrangerilor*

**5. Instrumente de dezvoltare ATL [2]**

ATL este acompaniat de un set de instrumente create cu ajutorul platformei Eclipse. In Figura este reprezentata arhitectura in ansamblu a intrumentelor de dezvoltare ATL ( ATL Development Tools (ADT)). ADT este compus din motorul de transformare ATL si din Mediul de dezvoltare al ATL integrat.

*5.1 Motorul*

Motorul ATL este responsbil de sarcinile de baza ATL: compilarea si executarea. Transformarile ATL sunt compilate in programe prin bit-cod specializat. Bit-codul este executat in Masinaria Virtuala ATL ( VM). VM este specializat in manipularea modelelor si ofera seturi de instructiuni pentru manipularea acestora.

Arhitectura motorului de executie ATL poate fi observat in Figura 7.

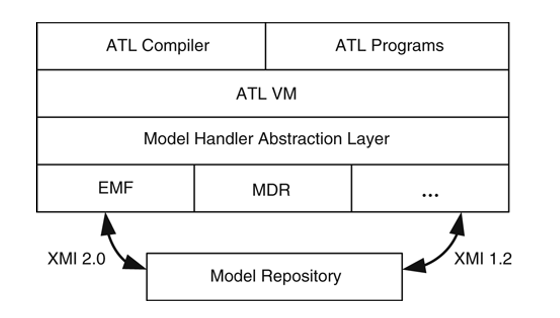


Figura 7

Pentru a izola VM de specificatiile sale este introdus un nivel intermedial numit strat de abstractizare a modelului manipulat. Acest strat traduce instructiunile de la CM pentru manipalarea modelelor in intructiuni al unui specific model manipulat.

*5.2 Editarea*

Editorul ATL suporta sublinierea sintaxelor, reportarea erorilor, si viziunea de tip contur. Partea din stanga jos a Figurii 8 arata cum exemplul Class2Relational.atl este reprezentat intr-un editor. Partea de jos dreapta a figurii arata conturul corespunzator.

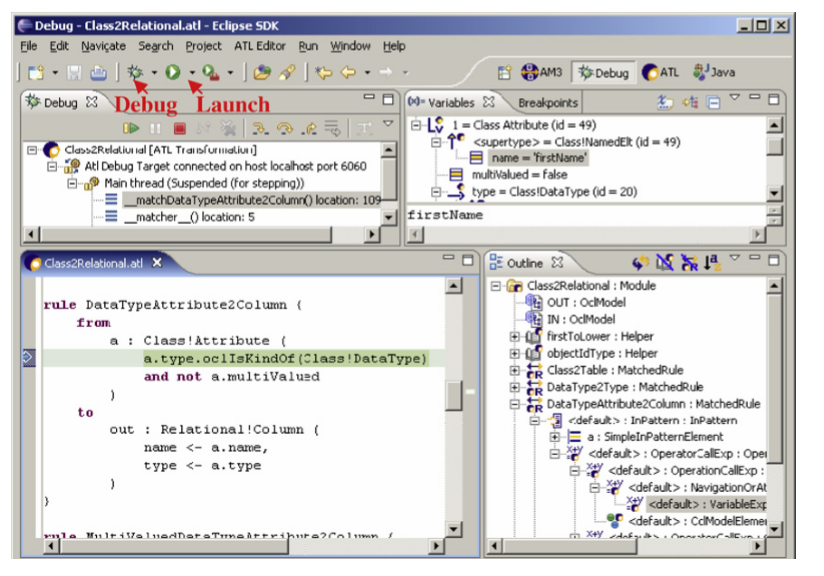


Figura 8

Compilatorul ATL este automat apelat la fiecare fisier ATL din fiecare proiect ATL in timpul unui procces de constructie Eclipse. Din oficiu acest proces este actionat cand un fisier este modificat.

Executarea unei transformari ATL necesita o sursa declarata si modele si metamodele tinta ce sunt legate catre modelele existente. Acest lucru este facut la lansarea ajutorului de configurare. Figura 10 ne da o imagine a acestui ajutor si ne arata corespondenta dintre interfata utilizatorului si contextul operational al transformarii aratate in Figura 9. Sagetile solide leaga modelele si metamodelele de declaratiile lor, iar sagetile intrerupte leaga declaratiile de fisierele corespunzatoare.

Motorul ATL deleaga modelele de citire si scriere catre Modelele manipulate care stau la baza.

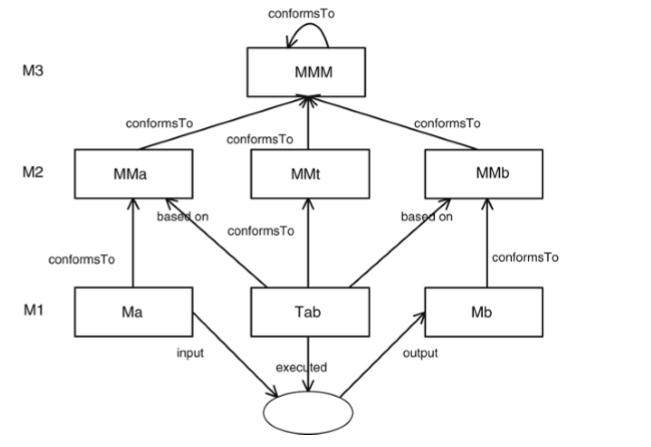


Figura 9

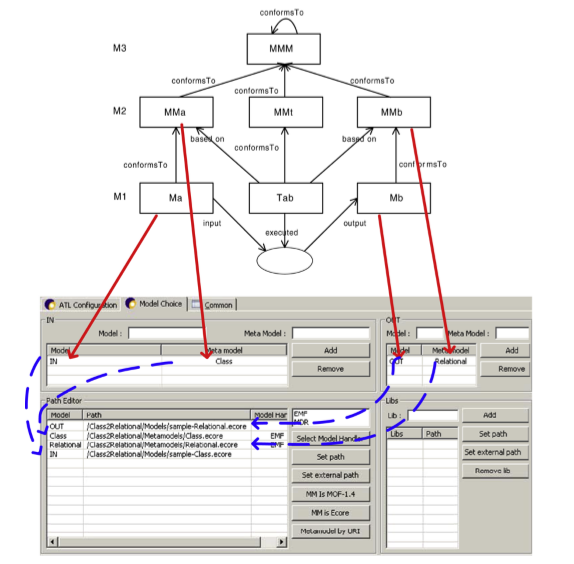


Figura 10

*5.3 Depanare*

Transformarile ATL pot fi depanate folosint aceleasi configuratii de lansare folosite la lansarea programului. Singura diferenta este ca acum se foloseste butonul de depanare in loc de butonul de lansare, asa cum se poate observa in Figura. Transformarile pot fi executate pas cu pas sau lasate sa functioneze normal. In acest caz executarea se opleste in momentul in care se identifica o eroare sau in momentul in care se ajunge la sfarsit. Contextul curent poate fi analizat folosind viziunea de tip cariabila din partea de sus dreapta a Figurii . Aceasta permite o navigare simpla la modelele sursa si tinta din contextul curent.

**6. Concluzii**

In aceasta lucrare am prezentat ATL: un limbaj de transformari de model hibrid dezvoltat ca o parte din ATLAS Model Management Architecture. ATL este sustinut se un set de instrumente de dezvoltare construite pe baza mediului Eclipse: un compilator, o masina virtuala, un editor si un debugger. ATL permite ca ambele abordari, imperativa si declarativa, sa fie folosite in definirea transformarilor, depinzand de problema respectiva.

ATL este curent folosit sau evaluat pe mai mult de 100 de site-uri, academice sau industriale. Exista si o librarie de transformari ATL disponibile open-source din proiectul M2M Eclipse. Statusul curent al instrumentelor ATL deja permite rezolvarea de probleme non-triviale. Acest lucru este demonstrat prin numarul de exemple implementate care este in continua crestere si de interesul pe care il arata ascensiune a rapida a comunitatii de utilizatori ATL ce ofera un feed-back semnificant. Mai mult de 80 de scenarii diferite ce asigura peste 160 de transformari individuale sunt disponibile pe website-ul ATL M2M.

**7. Bibliografie**

**[1] https://en.wikipedia.org/wiki/ATLAS\_Transformation\_Language**

**[2]http://universite.jejemaes.net/MA\_2/SysDis%20MA/Articles\_de\_r%C3%88f%C3%88rence/Bezivin\_\_\_\_ATL-\_A\_model\_transformation\_tool.pdf**

**[3] http://ac.els-cdn.com/S1571066106001393/1-s2.0-S1571066106001393-main.pdf?\_tid=8446659a-be85-11e5-be93-00000aab0f26&acdnat=1453191816\_5ccf4390c33bb3ca3ee0d38f31bf2688**

**[4] “Science of Computer Programming” - J.A. Bergstra**

**[5] “Electronic Notes in Theoretical Computer Science” - M.W. Mislove**