Inference of Safe Approximate Models for System Composition

Drd. Ing. Casandra HOLOTESCU

Una dintre problemele principale in ingineria software bazata pe componente este reprezentata de construirea riguroasa a unui sistem folosind mai multe componente software preexistente. Deoarece efectuarea manuala a acestui proces da de obicei nastere unui numar mare de erori, problema este de obicei rezolvata recurgandu-se la adaptarea si compunerea automata a sistemului respectiv. La data curenta, exista diverse metode de compunere a unui sistem pornind de la un set de componente preexistente, prin sintetizarea automata de adaptoare. Aceste adaptoare au scopul de a impune sistemului rezultat un comportament specificat. Totusi, acest tip de abordare necesita ca toate interfetele comportamentale ale tuturor componentelor din sistem sa fie complet specificate, ceea ce nu reprezinta o situatie prea des intalnita in industrie.

Pentru a adresa aceasta problema, a fost dezvoltata o tehnica de invatare a modelelor comportamentale pentru componente software al caror alfabet de evenimente contine atat evenimente controlabile, cat si necontrolabile. Un bun exemplu in acest sens il reprezinta componentele software asincrone: trimiterea la un moment dat a unui mesaj de catre o anumita componenta este un eveniment incontrolabil din exterior, deoarece decizia trimiterii mesajului este luata intern. Modelele comportamentale ale componentelor software se invata ca automate cu stari finite, automate ale caror tranzitii sunt determinate de producerea cate unui eveniment de tip trimitere/primire de mesaje.

Invatarea comportamentului unei componente sub forma unui automat cu stari finite este un proces de o complexitate ridicata, mai cu seama pentru componentele cu evenimente necontrolabile. Datorita acestor costuri ridicate, cel mai adesea modelele obtinute sunt doar aproximari, mai mult sau mai putin precise, ale comportamentului real al componentelor. Insa, atunci cand dorim sa folosim modelele invatate pentru compunerea unui sistem, dorim ca aceste modele sa fie sigure, chiar daca sunt aproximative. Adaptoarele generate pornind de la aceste modele nu au voie sa fie luate prin surprindere, la un moment dat, de un eveniment necontrolabil ne prevazut in model, dar care se manifesta in realitate. La fel, nu au voie nici sa incerce a controla componenta activand un eveniment controlabil prevazut de model, dar inexistent in practica. Metodele curente de invatare de modele, insa, considera doar componente black-box cu toate evenimentele controlabile si se axeaza mai degraba pe eficientizarea invatarii. Aplicarea unor astfel de metode in cazul componentelor cu evenimente necontrolabile poate rezulta in modele care nu aproximeaza in mod sigur comportamentul real.
De aceea, scopul abordarii este invatarea unor modele aproximative sigure. Astfel, comportamentul necontrolabil prevazut de un model aproximativ sigur reprezinta o supra-aproximare a comportamentului necontrolabil real al componentei, in timp ce comportamentul controlabil existent in model va subaproxima comportamentul controlabil real.

Asadar, metoda de invatare de modele comportamentale propusa reprezinta o metoda pentru invatarea de modele aproximative sigure pentru componente black-box, deci insuficient specificate. Invatarea se face in mod activ, la executie, printr-o explorare sistematica a spatiului de stari. Fiecare secventa de executie este retinuta intr-un arbore de secvente, folosit atat pentru ghidarea procesului de explorare a comportamentului, proces care se doreste a fi cat mai uniform, cat si, ulterior, pentru construirea modelului invatat. Daca numarul de executii posibile este limitat, invatarea se va termina intotdeauna cu un model aproximativ sigur, care poate fi folosit pentru a controla componenta reala. In schimb, daca nu avem stabilita o limita superioara a numarului de executii, procesul de invatare continua pana la obtinerea unui model precis, explorarea comportamentului componentei fiind in acest caz exhaustiva. Insa, datorita complexitatii exponentiale presupuse de explorarea exhaustiva, la executie, a comportamentului, invatarea de modele precise nu este fezabila in practica decat in cazul

componentelor cu un numar redus de stari si al caror alfabet contine, la randul sau, un numar relativ redus de evenimente.

Pentru a creste eficienta procesului de invatare, algoritmului initial i-au fost aduse o serie de imbunatatiri ulterioare. Astfel, pentru a preveni invatarea unor subseturi de comportament care oricum nu se vor regasi in sistemul ce se doreste a fi compus, algoritmul nu exploreaza, in mod activ, decat comportament relevant pentru specificatia globala a sistemului. Presupunem, in acest caz, ca specificatia sistemului este data sub forma unui automat cu stari finite. Cum explorarea comportamentului componentei black box are loc in mod dinamic, la rulare, executia va fi in acest caz ghidata pas cu pas spre indeplinirea specificatiei, si abandonata imediat atunci cand specificatia globala este violata la executie.

Pe langa o varianta centralizata a algoritmului, care invata si coordoneaza simultan toate componentele sistemului, a fost dezvoltata si o solutie distribuita de invatare de modele, care exploreaza comportamentul fiecare componente black-box in parte, la nivel local, permitand astfel procesului de invatare sa ruleze in paralel pentru componentele din sistem. Avantajul acestei solutii de invatare locala, pe langa avantajele aduse de paralelizarea invatarii, este acela de a fi mai bine adaptata structurii naturale a sistemelor distribuite.

O alta optimizare este adusa de orientarea procesului de invatare spre invatarea acelor structuri ciclice din comportamentul componentelor care se regasesc in ciclurile specificatiei globale. Acest lucru a fost considerat necesar in conditiile in care majoritatea scenariilor de executie a unui sistem sunt, la randul lor, ciclice. Invatarea aproximativa a modelelor de comportament prezinta dezavantajul ca nu reuseste intotdeauna identificarea tuturor ciclurilor din comportamentului unei componente, tocmai datorita constrangerilor de cost sub care se desfasoara. Aceasta problema poate fi insa ameliorata prin orientarea explorarii la executie a comportamentului spre acele cicluri necesare sistemului pe care dorim sa-l compunem. Explorarea prioritara a acestor secvente de executie conduce nu numai la identificarea mai precisa si mai rapida a ciclurilor, ci si, dupa caz, la modele aproximative de dimensiuni mai mici si care descriu mai bine comportamentul real al componentelor invatate.

Tehnicile de invatare descrise au fost implementate in prototipul BASYL si apoi rulate pe cateva studii de caz. Pentru o mai buna viziune de ansamblu, studiile de caz pentru validarea abordarii au fost alese atat din literatura de invatare de componente, cat si din literatura de generare de adaptoare. Rezultatele experimentale obtinute demonstreaza ca modelele aproximative sigure astfel invatate pot fi folosite cu succes pentru generarea automata de adaptoare care sa poata controla comportamentul real al componentelor, iar sistemul dorit poate fi astefl compus in mod sigur.

Capitolul 1: Introducere

Primul capitol al tezei are rol introductiv si motiveaza necesitatea abordarii curente.

La momentul actual, ingineria software bazata pe componente castiga tot mai mult teren. Cum componentele provin deseori de la producatori diferiti, e nevoie de solutii riguroase pentru a asigura corectitudinea unui sistem alcatuit din astfel de module. Astfel de solutii folosesc diverse tipuri de metode formale pentru compunerea corecta, in mod automat, a unui sistem din mai multe parti componente, prin generarea automata de componente adaptor/coordonator. Aceste metode, insa, necesita specificatii formale ale comportamentului fiecarei componente in parte, alaturi de o specificatie a comportamentului dorit al sistemului.

In practica insa, de multe ori, componentele nu sunt insotite de astfel de specificatii formale, ceea ce face dificila compunerea lor automata. In cazul unor astfel de componente insuficient specificate, pentru ca sistemul sa poata fi compus este necesara invatarea unui model comportamental pentru fiecare componenta in parte. De obicei, acest model comportamental este reprezentat de un automat cu stari finite. Invatarea unui astfel de automat se face la executie, prin testarea tuturor secventelor de executie posibile, si este un proces de o complexitate ridicata, care deseori produce doar modele de comportament aproximative.

Teza trateaza invatarea aproximativa sigura a modelelor comportamentale pentru componente al caror alfabet de evenimente contine atat evenimente controlabile, cat si necontrolabile. Astfel de componente sunt, de exemplu, componentele care comunica in mod asincron, prin trimitere si receptie de mesaje, deoarece in cazul acestora numai evenimentele de receptie de mesaje sunt controlabile din exterior, putandu-se trimite componentei unul sau altul din mesajele disponibile, in timp ce evenimentele de trimitere de mesaje se produc pe baza logicii interne a componentei.

Pentru componentele cu evenimente necontrolabile, este important ca modelele aproximative invatate sa contina intreg comportamentul necontrolabil real al componentelor, deoarece, astfel, adaptorul generat va fi pregatit sa reactioneze in orice moment, la orice eveniment necontrolabil fezabil. Totodata, este important si ca modelele invatate sa contina doar comportament controlabil valid, observat la executie, pentru ca in niciun moment adaptorul generat sa nu incerce evitarea unei cai eronate de executie prin activarea unui eveniment controlabil inexistent in realitate. Numim un model aproximativ care indeplineste cele doua cerinte de mai sus: supra-aproximarea comportamentului necontrolabil real, si subaproximarea comportamentului controlabil real, un model aproximativ sigur.

Abordarile curente din domeniul invatarii automate de modele comportamentale nu trateaza cazul componentelor cu evenimente necontrolabile, astfel ca modelele invatate de acestea nu reprezinta aproximari sigure ale comportamentului real al componentelor. Insa, atunci cand dorim sa realizam o compunere corecta a unui sistem continand componente cu evenimente necontrolabile, si cel putin o parte din aceste componente sunt incomplet specificate, comportamentul lor trebuie sa fie invatat sub forma unor modele de comportament care sa reprezinte modele aproximative sigure.

Invatarea de modele aproximative sigure ale comportamentului componentelor cu evenimente necontrolabile reprezinta principala contributie pe care teza si-o propune. In acest scop vor fi introduse mai multe metode de explorare a comportamentului componentelor insuficient specificate (in mod centralizat, distribuit, cu optimizare pentru explorarea prioritara a ciclurilor), o metoda de constructie a modelelor invatate pe baza informatiei adunate in procesul de explorare, si o implementare a abordarii teoretice realizata in tool-ul BASYL.

Capitolul 2: Notiuni preliminare

Capitolul introduce mai multe notiuni considerate necesare intelegerii tezei. La inceput este introdus domeniul ingineriei software bazate pe componente, urmarindu-sa dezvoltarea sa de la origini pana in prezent, alaturi de evolutia conceptului de componenta de la simplul modul de program la componente reutilizabile standardizate . Apoi, se prezinta mai in detaliu subdomeniul generarii automate de adaptoare prin intermediul unei game largi de tehnici din categoria metodelor formale. Cele mai multe dintre acestea utilizeaza diverse tipuri de automate cu stari finite pentru modelarea comportamentului componentelor si specificarea proprietatilor ce se doresc a fi impuse sistemului.

In continuare sunt expuse cateva aspecte teoretice esentiale. In primul rand, se prezinta in mod neexhaustiv si se definesc formal mai multe tipuri de automate, incepand cu automatele cu stari finite, continuand cu automate Mealy, cu automatele de interfata si cu automatele de intrare/iesire. Se insista pe legatura dintre un automat cu stari finite si limbajul regulat acceptat de acesta si pe conceptele de determinism, determinism la intrare, determinism la iesire, determinism comportamental, si non-determinism. Totodata sunt introduse si operatii intre automatele cu stari finite, cum ar fi produsul asincron si produsul sincron.

Subcapitolul urmator prezinta algoritmul L*, algoritmul de referinta in domeniul invatarii automate de modele. Acesta permite invatarea unui model comportamental la executie, presupunand cunoscute alfabetul si numarul maxim de stari ale componentei invatate; in plus se presupune ca aceasta poate fi oricand resetata la starea initiala. L* invata modelul comportamental al componentei utilizand doua tipuri de interogari: de apartenenta (interogheaza daca o secventa de evenimente apartine limbajului componentei) si de echivalenta (propune un model ipotetic si interogheaza daca acesta descrie exact comportamentul componentei). Invatarea avand loc la executie, interogarea de echivalenta este realizata prin intermediul unui set de interogari de apartenenta, testand secventele de evenimente care ar putea conduce la contradictii intre modelul ipotetic actual si comportamentul real.

Apoi, ultimul subcapitol prezinta o serie de elemente de teorie a controlului sistemelor cu evenimente discrete. Sunt introduse mai multe notiuni din domeniu (generator, limbaj marcat, calitatea unui generator de a fi blocant sau neblocant), se definesc evenimentele controlabile si necontrolabile. Apoi se discuta notiunea de controller sau supervisor al unui sistem, si se descrie un algoritm de generare automata a unui controller maxim permisiv care sa restrictioneze comportamentul unui sistem bine specificat la un subset comportamental permis de o proprietate de buna functionare a sistemului.

Capitolul 3: Stadiul actual al domeniului

Capitolul prezinta in mod succint mai multe abordari inrudite din domeniul invatarii automate de modele, generarii de controllere, testarii sistemelor asincrone si nu numai, verificarii sistemelor la executie, etc. Cel mai apropiat algoritm cunoscut este considerat algoritmul de verificare a componentelor insuficient specificate, care imbina verificarea formala cu invatarea de modele la executie. In acest caz, invatarea se face folosind algoritmul L* deja prezentat. Apoi, sunt prezentate ultimele variante si optimizari cunoscute ale algoritmului L* (abordand, de exemplu, cazul componentelor cu un alfabet de intrare infinit sau parametrizat, cu un numar infinit de stari, cu un anumit grad, redus, de nondeterminism, etc.). Urmatorul subcapitol continua prezentarea abordarilor ce fac uz de L*, introducand metode care imbina invatarea de modele cu generarea automata de controllere. In continuare sunt prezentate si cateva metode de invatare de modele neinrudite cu L*, bazate ori pe invatarea pasiva, pornind de la un set de secvente de executie colectate, ori pe extinderea unei suite de teste preexistente, ori pe considerarea unor tipuri speciale de componente, etc. Subcapitolul urmator introduce doua metode de testare a sistemelor asincrone, care iau in considerare dihotomia comportament controlabil versus comportament necontrolabil, alaturi de asumptia de necontrolabilitate a trimiterii de mesaje in generarea de teste pentru astfel de sisteme.

Capitolul 4: Invatarea de modele aproximative sigure

Capitolul 4 prezinta in detaliu solutia teoretica de invatare de modele aproximative sigure, alaturi de cateva variante relevante ale acesteia. Comportamentul componentelor va fi invatat sub forma unui automat cu stari finite ale carui tranzitii corespund fiecare unui eveniment de trimitere sau receptie de

mesaje. Se presupun cunoscute lungimea maxima a unei cai aciclice prin model, precum si numarul total maxim de vizite ale unei stari pana cand toate evenimentele necontrolabile din acea stare ajung sa se manifeste la executie, i.e. limita de fairness. Secventele de executie colectate pe parcursul procesului de explorare a comportamentului componentei vor fi retinute intr-un arbore de secvente, cate unul pentru fiecare componenta din sistem.

Se prezinta intai invatarea de modele prin explorarea centralizata a comportamentului acestora. Aceasta solutie presupune ca in sistem se introduce un adaptor proactiv care va intermedia, din acel moment, toate interactiunile din sistem. Acest adaptor proactiv preia controlul fiecarei executii si, in functie de secventele pe care doreste sa le incerce, dirijeaza executia curenta spre una sau alta dintre acestea. Controlul executiei curente se realizeaza prin interceptarea tuturor mesajelor trimise de componentele din sistem si retransmiterea selectiva a acestora catre componentele destinatie, retransmitere care se face in functie de ce evenimente controlabile se doresc activate. Intotdeauna, pentru activare se alege, pe baza informatiei stocate in arborele de secvente explorate (care retine si de cate ori a fost observata o anumita secventa de evenimente, de orice dimensiune), evenimentul cel mai putin explorat din punctul curent. Explorarea centralizata a comportamentului componentelor continua pana cand se atinge un numar maxim permis de executii, stabilit in prealabil, sau pana cand au fost epuizate toate executiile posibile, pentru toate componentele din sistem.

Ulterior explorarii la executie a comportamentului componentelor are loc construirea modelelor. Automatele cu stari finite care descriu comportamentul componentelor vor fi construite pornind de la arborele de secvente observate, dupa o prelucrare in prealabil a acestuia. In primul rand, toate evenimentele controlabile ramase neexplorate la executie vor fi marcate ca nefezabile in arborele de secvente, pentru a asigura proprietatea de subaproximare a comportamentului controlabil. Apoi, toate evenimentele necontrolabile din arbore, ramase neobservate la executie, vor fi marcate ca nefezabile daca nodul lor de origine din arbore a fost vizitat de mai multe ori decat limita de fairness, care ne asigura ca nu vor fi observate ulterior, si ca fezabile, dar cu comportament ulterior nedeterminat, daca nodul respectiv a fost vizitat de un numar de ori inferior limitei de fairness. Apoi, se partitioneaza nodurile arborelui pe baza relatiei de echivalenta dintre noduri reprezentata de bisimularea pe un adancime egala cu lungimea maxima a unei cai aciclice prin model, care in cazul nodurilor unui arbore este totuna cu a avea acelasi limbaj de secvente de lungimea maxima specificata. Fiecare dintre partitiile rezultate este asociata unei stari din automatul final, iar tranzitiile dintre noduri apartinand de partitii diferite devin tranzitii in automatul final. Un caz aparte il reprezinta aici evenimentele necontrolabile cu comportament ulterior nedeterminat, mentionate mai sus. Tranzitiile induse de aparitia acestora vor fi toate reprezentate ca ducand spre o stare speciala, care are tranzitii de iesire doar pe evenimente necontrolabile, iar aceste tranzitii de iesire, de fapt, duc inapoi spre ea insasi. Astfel este construit, pentru oricare dintre componentele din sistem, automatul invatat in urma procesului de explorare a comportamentului.

Invatarea de modele prin explorarea localizata a comportamentului este relativ similara invatarii centralizate, cu diferenta ca in loc de un singur adaptor proactiv central, avem cate un adaptor proactiv local pentru fiecare componenta in parte. Adaptoarele proactive locale comunica intre ele pentru a-si transmite mesajele interceptate de la componentele carora le corespund. Fiecare astfel de adaptor exploreaza in mod local comportamentul componentei insuficient specificate, activand anumite evenimente controlabile si dezactivandu-le pe celelalte in functie de necesitatile procesului de explorare. Explorarea comportamentului componentelor se face astfel in paralel, ceea ce creste eficienta procesului de invatare de modele. In plus, un astfel de proces de invatare este mai bine adaptat structurii naturale a sistemelor distribuite. Constructia modelului se face in mod similar explorarii centralizate, dupa ce explorarea la executie ia sfarsit.

Pentru reducerea numarului de executii efectuate pe parcursul explorarii comportamentului si eliminarea executiilor inutile, s-a luat in considerare solutia de optimizare urmatoare: in cazul in care sistemul ce se doreste a fi compus este caracterizat de o anumita proprietate care i-ar limita limbajul initial la un sublimbaj, nu e necesar a se explora le executie decat acele secvente de evenimente care sunt acceptate de proprietatea sistemului. Restul executiilor, indiferent daca sunt fezabile sau nu, vor fi oricum restrictionate la compunerea sistemului, de adaptorul ce va fi generat automat cu scopul de a impune proprietatea dorita.

Alta solutie de imbunatatire a algoritmului de invatare de modele o reprezinta optimizarea invatarii prin explorarea cu prioritate a ciclurilor. Aceasta porneste de la observatia ca in majoritatea sistemelor, cele mai multe scenarii de utilizare sunt repetitive, deci ciclurile sunt importante si, chiar si intr-un model aproximativ, trebuiesc identificate prioritar. Pentru a obtine acest lucru, procesul de explorare optimizat va acorda o prioritate crescuta acelor scenarii de executie si acelor secvente care corespund unor cicluri din specificatia proprietatii dorite pentru sistemul ce va fi compus. Acest lucru duce la o focalizare a invatarii pe scenariile ciclice, ceea ce, pentru acelasi numar de executii, duce la o identificare mai rapida si mai precisa a ciclurilor fata de abordarea neoptimizata.

In final, odata ce modelele aproximative sigure au fost construite pentru toate componentele din sistem, sistemul dorit este compus prin intermediul sintezei unui adaptor. Acesta este obtinut dintr-un controller generat in mod automat pentru sistemul respectiv cu ajutorul algoritmului clasic de sinteza de controllere, din teoria controlului sistemelor cu evenimente discrete, algoritm prezentat in partea de notiuni preliminare. Adaptorul se obtine din controller astfel: fiecare tranzitie pe un eveniment de trimitere al unui mesaj se transforma intr-o tranzitie pe evenimentul de receptie al aceluiasi mesaj.

In incheierea capitolului de teorie sunt prezentate o serie de demonstratii formale ale unor proprietati atat ale modelelor aproximative sigure invatate, cat si ale adaptorului generat: in primul rand, faptul ca adaptorul respectiv este sigur, apoi ca modelele invatate accepta toate secventele de executie observate, etc. Demonstratiile sunt urmate de o analiza de complexitate a algoritmului, si o comparatie intre solutia elaborata si o posibila adaptare a algoritmului L* pentru invatarea de modele ale componentelor cu evenimente necontrolabile. Complexitatea ridicata a ambelor abordari reprezinta o caracteristica a domeniului, si o consecinta a luarii in considerare a evenimentelor necontrolabile. Fata de posibila adaptare a lui L*, abordarea elaborata aduce in plus costul construirii de modele, faza care la randul ei are o complexitate ridicata. Insa, tocmai aceasta faza asigura ca modelele invatate sunt aproximari sigure ale comportamentului real al componentei, care isi pastreaza proprietatea aceasta in orice numar de executii ar fi invatate, spre deosebire de L* care nu poate garanta invatarea unor modele aproximative sigure in limita unui numar maxim prestabilit de executii permise.

Capitolul 5: Rezultate experimentale

Abordarea descrisa a fost implementata in tool-ul BASYL. Acesta a fost scris in intregime in Java, si contine o componenta de monitorizare a evenimentelor ce se produc la executie, o componenta de decizie, care, la fiecare pas, alege evenimentul controlabil ce va fi activat in continuare, componente de coordonare a invatarii pentru metodele de explorare centralizata, respectiv distribuita, implementari ale structurilor abstracte de date descrise (arbori, automate cu stari finite), etc.

Experimentele s-au concentrat pe 4 studii de caz, dintre care unul a fost ales din literatura de verificare de servicii (Protocolul Single Sign On), altul din cea de compunere automata a componentelor in

sisteme prin generarea de adaptoare (studiul de caz care descrie un Sistem de gestionare al datelor), precum si doua studii de caz din literatura de invatare de modele (studiile de caz Domotics si Protocolul SIP). Pe primele doua studii de caz, care vizau sisteme cu mai multe componente, au fost testate metodele de explorare centralizata si distribuita a comportamentului, in timp ce modelele pentru entitatile din Domotics si din Protocolul SIP au fost invatate in izolare. Rezultatele invatarii, pentru mai multe variante ale limitei superioare a numarului de executii, au fost supuse unei analize calitative, respectiv au fost discutate asemanarea modelului aproximativ cu modelul original, numarul de cicluri din modelul original care au fost identificate, si precizia identificarii lor, numarul de tranzitii necontrolabile din model care duc spre starea simbolizand comportamentul viitor nedeterminat, etc. Pe toate studiile de caz vizate s-a putut compara, prin prisma modelelor obtinute, metoda de invatare neoptimizata cu cea optimizata pentru explorarea prioritara a ciclurilor: modelele obtinute cu ajutorul metodei optimizate au avut in general dimensiuni sensibil mai mici decat celelalte, si au prezentat o identificare mai timpuri si mai precisa a ciclurilor din specificatie. In plus, pentru toata studiile de caz cu exceptia lui Domotics, s-au generat de controllere pentru modelele obtinute si proprietatea specificata, pentru generarea controllerelor folosindu-se tool-ul existent Supremica. Controllerele generate au fost apoi verificate impreuna cu modelele originale ale componentelor invatate, dovedindu-se sigure. Astfel, abordarea a fost validata.

Capitolul 6: Concluzii

Finalmente, capitolul de concluzii prezinta un rezumat al contributiilor proprii, descrie si discuta limitarile curente ale abordarii elaborate in aceasta teza, precum si, pornind de la aceste limitari, perspectivele de dezvoltare ulterioara a acesteia. Ca si contributie principala a tezei este numita introducerea unei solutii de invatare de modele aproximative sigure pentru componentele software cu evenimente necontrolabile. Aceasta solutie a fost realizata prin intermediul urmatoarelor contributii secundare: elaborarea celor doua metode, centralizata si distribuita, de explorare la executie a comportamentului componentelor dintr-un sistem, introducerea unei optimizari orientate pe cicluri a celor doua metode de explorare, o metoda de constructie a modelelor invatate pe baza observatiilor adunate pe parcursul fazei de explorare, metoda care asigura caracterul aproximare sigura a comportamentului real al componentei pentru modelul invatat intr-un numar limitat de executii, si, in cele din urma, tool-ul BASYL, in care au fost implementate toate tehnicile enuntate mai sus si care a fost folosit pentru validarea experimentala a abordarii pe un set de 4 studii de caz. Principala limitare a abordarii este, cum am mai spus, complexitatea ridicata a invatarii, specifica domeniului. Pe langa aceasta, o alta limitare importanta o constituie faptul ca, deocamdata, BASYL lucreaza la un nivel de abstractie inalt si ca, pentru a putea fi utilizat pe componente reale, o mapare a alfabetului concret al componentelor reale la un alfabet abstract, mai simplu si redus ca dimensiune, este necesara. Perspectivele de dezvoltare ulterioara sunt orientate inspre reducerea si inlaturarea, pe cat posibil, a neajunsurilor actuale, astfel propunandu-se realizarea unui algoritm mai rapid de constructie de modele, bazat pe o partitionare mai eficienta a nodurilor arborelor, dezvoltarea unor tehnici automate de generare automata a unor abstractizari sigure, care sa permita invatarea de componente reale, cu un numat mare de evenimente, si, nu in cele din urma, adaptarea procesului de invatare unor situatii neprevazute de algoritmul curent, in care, de exemplu, primirea unui mesaj de catre o componenta nu este un eveniment observabil din exterior, sau in care invatarea porneste de la un model imperfect al componentei.