ADAPBEEN - Adaptible Automata Framework

Reinaldo Felipe Soares Araujo; João Augusto Felberg Jacobsen; Willians Magalhães Primo;
Reginaldo Inojosa da Silva Filho.

***Resumo*— In this paper we present the ADAPBEEN, a Framework that implements the algebraic model of adaptive automata of first and second order. Our framework allows us to build applications that use adaptive technology, that is, programs that change their behavior depending on the received inputs. To this end, ADAPBEEN was built in a modular way. For better performance, it was developed in C ++ and in a way to allow the code to be portable.**

I. INTRODUÇÃO

Atualmente cada vez mais vêm tem sido utilizados dispositivos auto-modificantes em diferentes áreas tais como: Linguagens Naturais [1], Linguagens de Programação [2], Robótica [3], etc. E apesar da teoria adaptativa estar madura, geralmente as implementações estão restritas a projetos específicos [4], [5]. Algumas bibliotecas, ferramentas e frameworks tem sido construídos na tentativa de resolver esse problema, tais como: AdapTools[6], AdapLib[5], e a biblioteca para autômatos adaptativos[7].

Diante dessa demanda por dispositivos auto-modificantes, e por conta das questões relacionadas ao estudo da complexidade nos autômatos adaptativos, foi proposta uma nova formulação algébrica para o autômatos adaptativos. Esta formulação levou à descrição dos autômatos de primeira e segunda ordem aplicada à inferência indutiva de linguagens formais [8], porém a mesma ainda não havia sido implementada e experimentada. Este artigo apresenta o framework ADAPBEEN, que consiste na implementação, em código, dessa formulação algébrica.

O ADAPBEEN é um Framework de codigo aberto portátil, desenvolvido em C++ e que pode ser utilizado no Windows, Linux ou Mac. Ele utiliza os conceitos de camada para a estruturação de sua arquitetura, tornando-o mais didático, modularizado e fiel à teoria. O nosso objetivo em desenvolvê-lo consiste em prover uma biblioteca para o programador que pretenda utilizar a tecnologia adaptativa na construção de soluções computacionais.

O trabalho esta organizado como se segue: A seção seguinte apresenta uma fundamentação teórica sobre dispositivos adaptativos para melhor compreensão da implementação realizada. A seção III refere-se aos detalhes estruturais do ADAPBEEN com uma descrição de cada classe implementada e suas funcionalidades. E as seções IV e V além de uma sucinta explicação sobre os autômatos de primeira e segunda ordem as seções contem exemplos.

*Agradeço ao Grupo PET Fronteira e a Fábrica de Software pelo apoio.

II. FUNDAMENTAÇÃO TEÓRICA

Os autômatos adaptativos possuem a capacidade de modificar a própria estrutura em tempo de execução, com isso, estados e transições podem ser acrescentados ou removidos. Essas alterações estruturais dependem da entrada, do estado atual do autômato e das transições associadas ao mesmo que, caso forem adaptativas, podem alterar a topologia do autômato. Em função desse comportamento, o poder computacional o autômato adaptativo é equivalente ao de uma Máquina de Turing [9], capaz de reconhecer, inclusive, linguagens livres de contexto. Embora o autômato adaptativo seja dotado de grande poder computacional, o mesmo lida com o mesmo problema da Máquina de Turing em relação a incomputabilidade [10].

Durante os últimos anos, a tecnologia adaptativa tem se desenvolvido em uma série de aplicações. Os autômatos adaptativos, uma das facetas da tecnologia adaptativa, consiste em uma solução computacional para problemas complexos. Nosso objetivo neste trabalho é disponibilizar um framework, o ADAPBEEN, que sirva de base para os desenvolvedores criarem suas aplicações adaptativas.

Novas formulações para o modelo são continuamente apresentadas [10], [8]. O modelo escolhido para implementar o ADAPBEEN foi a formulação algébrica para o Autômato Finito Adaptativo de Segunda Ordem, que tem uma forte conexão com o aprendizado indutivo no limite. O formalismo dos autômatos finitos de segunda ordem é desenvolvido sobre o modelo dos autômatos finitos de primeira ordem, uma extensão natural dos autômatos adaptativos clássicos[8]. Essa nova formulação procura resolver os problemas de ambiguidades e imprecisões das formulações anteriores, além de aumentar a capacidade de expressão do modelo original. Neste contexto surgiram então as transformações adaptativas ocorridas em um autômato, sendo que elas foram conceituadas como ações básicas de inserção e remoção de transições no dispositivo subjacente. Utilizando-se esses operadores, foi definida uma variante da formulação dos autômatos adaptativos (modelo original), chamada autômatos adaptativos de primeira ordem. A escolha do nome "Autômatos Adaptativos de Primeira Ordem"foi proposital pelo autor, para levar a dedução de uma próxima ordem que tem como dispositivo subjacente ou de ordem anterior, ou seja, um autômato adaptativo de segunda ordem que insere e remove transições adaptativas de primeira ordem.

III. ADAPBEEN FRAMEWORK

Como mencionado inicialmente, o ADAPBEEN segue uma hierarquia de camadas que pode ser observada no dia-

grama da Figura 4. A primeira camada é formada pela classe *automaton* e a segunda camada, formada pela classes *DFA* e *NFA*, são responsáveis pelos autômatos finitos determinísticos e não-determinísticos que são capazes de reconhecer linguagens irregulares. A terceira camada é formada pela classe *AAFO* que é capaz de gerar autômatos adaptativos de primeira ordem, capazes de reconhecer linguagens livres de contexto. A última camada é formada pela classe *AASO*, capaz de gerar autômatos adaptativos de segunda ordem, que podem aprender famílias de linguagem.

É importante destacar a sequência com que cada camada é executada, as palavras reservadas para lidar com a adaptatividade e o fato de que as mudanças ocorrem em tempo de execução. A sequência de execução é da primeira camada até a última, ou seja, primeiro serão executadas as transições pertencentes ao automato finito não-determinístico, em seguida as sequências positivas e negativas de primeira ordem associadas a essa transição e por último as sequencias positivas e negativas de segunda ordem que também são associadas a transição. As palavras reservadas podem ser mudadas nas definições e são elas "qs"e -1"o primeiro significa que é um estado dinâmico e a segunda representa o símbolo vazio.

Segue abaixo uma descrição resumida de cada classe e suas funcionalidades:

A. Classe state

Esta classe é um dos elementos básicos de um autômato juntamente com as transições. Os estados são muito importantes em um autômato para se determinar um estágio e a determinada ação a entrada recebida. A classe *state* estrutura os estados de um autômato, os quais podem ser estados estáticos ou dinâmicos.
A classe *state* possui os atributos *_name*, *symbol*, *activated*, *stateOrigin*. Estes atributos são utilizados nos estados dinâmicos quando o autômato é de primeira ou segunda ordem. Os estados estáticos são os estados tradicionais/estáticos usados nos autômatos finitos determinísticos e não determinísticos, sendo necessário para o funcionamento do mesmo somente um nome. Exemplo:

state q1("q1);

Os estados dinâmicos são úteis quando se é necessário referenciar um estado pela função que ele exerce e que pode sequer existir. Eles são constituídos de três parâmetros, que são usados no método de busca *Sch* que está implementado na classe *automaton*.
Exemplo: Supondo que a cadeia de entrada foi “aabb” e que o autômato adaptativo criou um estado chamado *s4* e uma nova transição que liga o estado *s4* com o estado *q3* (o qual é estático) com o caractere ’a’. Ao declarar um estado dinâmico com essa forma: *state* desconhecido("q3","a",0), é feita uma solicitação para que procure um estado que tenha uma transição com *q3* com o símbolo ’a’ e que a transição esteja partindo dele (se o terceiro parâmetro fosse um então estaria chegando nele). Cabe salientar que caso a cadeia de entrada fosse diferente, por exemplo uma cadeia maior, então o estado *s4* continuaria existindo, mas poderia não ser o estado que conecta o estado *q3* ao autômato.

B. Classe transition

A classe *transition* possui os atributos *origin*, *destiny*, *symbol* e *shape* que são os componentes básicos de uma transição. Um autômato, geralmente, é composto em sua estrutura por estados e transições, sendo essa classe responsável pelas transições do autômato.

As transições são elementos básicos de um autômato, que informa para qual estado o autômato deve ir caso esteja em um determinado estado e receba um determinado símbolo como entrada. Nesse modelo as transições podem ser criadas através da classe *transition* que possui os atributos *origin*, *destiny*, *symbol* e *shape*. Além dos *getters* se *setters* usuais do paradigma orientado a objeto, a classe traz também os métodos *Start* e *Fin* que são utilizados no modelo algébrico.

C. Classe automaton

Cada autômato implementado no ADAPBEEN tem em sua base a classe *automaton*, que possui os elementos básicos dos autômatos, sendo eles o alfabeto, estados finais, estados iniciais, transições e o nome do autômato. Na classe *automaton* estão implementados também os métodos padrões da programação objeto como *getters* e *setters*, métodos do modelo teórico, e métodos para visualização. Os métodos do modelo teórico são os *Sch*, *rem* e *ins* que possuem a função de busca, remoção e inserção extremamente úteis para a adaptatividade. E para a visualização os métodos *printStates*, *printInitialStates*, *printFinalStates*, *printTransitions*, *printAlphabet* que possuem a função de imprimir todos os estados do autômato, imprimir os estados definidos como iniciais, imprimir os estados definidos como finais, imprimir as transições pertencentes ao autômato e imprimir os símbolos que o autômato reconhece respectivamente.

D. Classe DFA

A classe *DFA* abreviação de "Deterministic Finit Automata"(Autômato Finito Determinístico), possui o método *output* implementado, que permite saber se uma palavra pertence ou não ao autômato. Se o retorno do método *output* for verdadeiro então significa que a palavra parametrizada pertence ao autômato, se for falsa significa que não pertence ao autômato.

E. Classe NFA

A classe *NFA* abreviação para "Nondeterministic Finite Automaton"(Autômatos Finitos Não-determinísticos) é usada como base para os autômatos adaptativos de primeira e segunda ordem. Esta classe contem os métodos *output*, *toAFNDG* e *toDFA*. O método *output* é utilizado para saber se a palavra parametrizada pertence ou não ao autômato, ou seja, se o retorno for verdadeiro a palavra então pertence ao autômato e se for falso então a palavra não pertence ao autômato. O método *toAFNDG* transforma o autômato em um autômato generalizado que é um autômato que possui apenas um estado final. O método *toDFA* retorna um autômato finito determinístico equivalente.

F. Classe AAFO

Um automato adaptativo de primeira ordem tem como seu dispositivo subjacente um autômato finito não-determinístico nesse modelo. A classe *AAFO* abreviação para "Adaptatible Automata First Order"(Automato Adaptativo de Primeira Ordem) é capaz de lidar com autômatos adaptativos que podem ser reconhecedores de linguagens livres de contexto. A classe tem um novo atributo chamado *setoftests* que é o conjunto de comportamentos do automato de primeira ordem. Os elementos do conjunto de comportamentos é um par que possui um ID e um *pairFO* (pair First Order) que é um par com uma sequência positiva e negativa do que deve ser acrescentado ou removido do dispositivo subjacente. As novos métodos implementados na classe são *insFO*, *remFO* e os *getters* e *setters* relativos ao *setoftests*. Os métodos *insFO* e *remFO* são para remover pares adaptativos de primeira ordem do conjunto de comportamento (*setoftests*). Uma importante observação é que o ID do par adaptativo faz referência a qual transição do dispositivo subjacente ele está relacionado, ou seja, qual será a transição que o ativará. Logo o ID do par adaptativo e da transição necessitam ser correspondentes.

G. Classe pairFO

Um autômato adaptativo de primeira ordem tem como dispositivo subjacente um automato finito não-determinístico nesse Framework, sendo assim sequência positivas e negativas vão adicionar e remover transições e estados do *afnd* subjacente. A classe *pairFO* implementa o par ordenado adaptativo de transições próprias e estrangeiras que devem ser removidas e acrescentadas do automato. A definição de transições próprias e estrangeiras está disponível no modelo teórico na qual foi baseado o Framework.

H. Classe negativeSequence

Sequências negativas são transições estrangeiras que devem ser acrescentadas ao automato. Aqui representada pela classe *negativeSequence*.

I. Classe positiveSequence

Sequências positivas são transições próprias que devem ser removidas do autômato aqui representada pela classe *positiveSequence*.

J. Classe AASO

Um autômato adaptativo de segunda ordem tem como dispositivo subjacente um autômato adaptativo de primeira ordem, ou seja, um autômato adaptativo de segunda ordem adiciona e remove pares adaptativos de primeira ordem. Uma vez que um autômato adaptativo de primeira ordem pode ser um reconhecedor para uma linguagem livre de contexto ao modificar o automato de primeira ordem, a linguagem do reconhecedor é alterado podendo reconhecer uma nova linguagem. A classe *AASO* é a camada responsável pela adaptatividade de segunda ordem com capacidade de inferir um autômato de primeira ordem aprendendo assim uma linguagem. *AASO* tem três novos atributos chamados de *setoftestsSO*, *inferedAutomata* e *flag*. O atributo *setoftestsSO*, semelhante ao *setoftests* da classe *AAFO*, é o conjunto de comportamentos do *AASO* sendo seus elementos pares de inteiros e pares adaptativos de segunda ordem, isto é, o inteiro é o ID dando referência a transição que ativa as adaptações que devem ser feitas e os pares adaptativos de segunda ordem são os *pairSO* que são sequencias positivas e negativas de pares adaptativos de primeira ordem que devem ser incluídos e removidos do *AASO*. Além disso o atributo *inferedAutomata* é atualizado após um automato ser inferido pelo método *inferAutomata*, sendo assim capaz de agir como um reconhecedor para a linguagem inferida. Os novos métodos implementados na classe são o *inferAutomata*, *insSO* e *remSO*. O método *inferAutomata* cada vez que é utilizado transformações são feitas no dispositivo subjacente (*inferedAutomata*) que após se estabilizar atualiza o atributo *flag* para o valor *true* significando que ouve uma convergência e foi aprendida uma linguagem.

K. Classe pairSO

Um par adaptativo de segunda ordem é um par que possui uma sequência positiva e uma sequência negativa que são os pares adaptativos de primeira ordem, que devem ser removidos e acrescentados ao autômato de primeira ordem. A classe *pairSO* implementa o par ordenado de segunda ordem que possuem pares ordenados de primeira ordem, que devem ser removidos ou acrescentados ao autômato modificando assim o mesmo.

L. Classe negativeSequenceSO

Sequência negativa de segunda ordem são os pares adaptativos de primeira ordem que devem ser acrescentados ao autômato. Neste Framework a sequência negativa de segunda ordem é implementada como classe *negativeSequenceSO*.

M. Classe positiveSequenceSO

Sequência positiva de segunda ordem são os pares adaptativos de primeira ordem que devem ser removidos do autômato. Aqui representada pela classe *positiveSequenceSO*.

IV. O Autômato Adaptativo de Primeira Ordem

Um autômato adaptativo de primeira ordem que é capaz de reconhecer linguagens livres de contexto, e como já fora dito na seção III, a terceira camada representada pela classe AAFO (seção III-F) é capaz de criar autômatos adaptativos de primeira ordem. Segue abaixo (Figura VI) um exemplo de implementação de um autômato de primeira ordem capaz de reconhece a linguagem $L1 = a^n b^n \mid n \geq 1$ ilustrado na Figura V.

V. O Autômato Adaptativo de Segunda Ordem

A quarta camada do ADAPBEEN é capaz de inferir autômatos adaptativos de primeira ordem e está implementada na classe AASO descrita na seção III-J. Que foi proposto no modelo teórico[8] ele deve ser capaz de aprender uma linguagem, uma discussão e experimento para um próximo trabalho. Segue abaixo na Figura V um exemplo de um Autômato Finito Adaptativo de Segunda Ordem que pode

inferir um reconhecedor para as linguagens $L1 = a^n b^n c$ e $L2 = ab^n c^n$ e consequentemente aprender um linguagem. A Figura V exemplifica um automato de primeira ordem inferido pelo AASO(Figura V) e capaz de reconhecer a linguagem $L1 = a^n b^n c$.

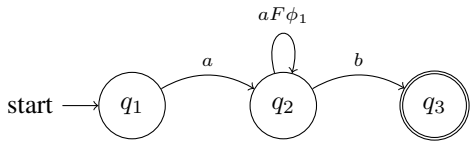

Fig. 1. Automato que reconhece a $L1 = a^n b^n \mid n \geq 1$

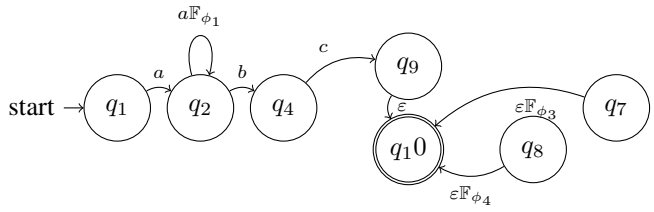

Fig. 2. Exemplo de autômato inferido pelo autômato da Figura V que reconhece a linguagem $L2 = ab^n c^n \mid n \geq 1$

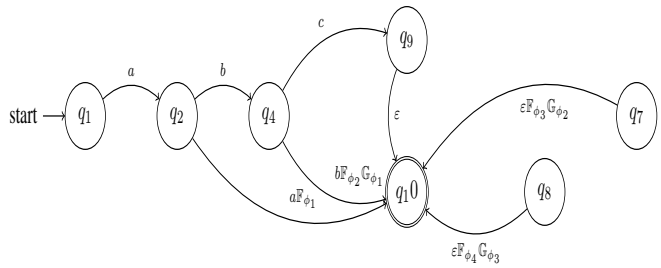

Fig. 3. Exemplo de Autômato Finito de Segunda Ordem capaz de aprender as linguagens $L1 = a^n b^n c$ e $L2 = ab^n c^n \mid n \geq 1$

VI. Conclusões

A implementação do ADAPBEEN é uma iniciativa que visa divulgação e promoção da tecnologia adaptativa. A partir deste trabalho, será possível aos programadores o desenvolvimento de soluções adaptativas de maneira rápida e robusta. O framework apresentado aqui é acessivel a todos que desejam utilizá-lo e estará disponível no BitBucket.

O desenvolvimento desse framework foi um trabalho realizado no âmbito do grupo de pesquisa BioCompTech. O desenvolvimento seguiu naturalmente o modelo algébrico e permite, a quem o utilizar, o acesso à adaptatividade de primeira e segunda ordem.

Os teste realizados com o framework mostraram que o ADAPBEEN encontra-se completamente operacional. Como atividades futuras, planeja-se a criação de aplicativos adaptativos nas áreas de processamento de linguagens e jogos.

Referências

[1] D. Padovani and J. J. Neto, "Adaptive automata applied to natural language processing," *Procedia Computer Science*, vol. 109, pp. 1152–1157, 2017.

[2] P. Cereda and J. J. Neto, "Um arcabouço para extensibilidade em linguagens de programação," in *Memórias do WTA 2015 IX Workshop de Tecnologia Adaptativa*, p. 18.

[3] L. C. Barros and A. R. Hirakawa, "An approach by straight line segment adaptive techniques in robot navigation," *IEEE Latin America Transactions*, vol. 12, no. 7, pp. 1292–1297, 2014.

[4] H. Pistori, J. J. Neto, and E. Costa, "A free software for the development of adaptive automata," in *Proceedings of the IV Workshop on Free Software-WSL (IV International Forum on Free Software)*, 2003.

[5] F. L. Siqueira, "The adaplib library for execution of adaptive devices: Architecture and use," *IEEE Latin America Transactions*, vol. 9, no. 2, pp. 213–218, 2011.

[6] L. Jesus, D. Santos, A. Castro Jr, and H. Pistori, "Adaptools 2.0: Aspectos de implementação e utilização," *Revista IEEE América Latina. Vol*, vol. 5, pp. 1548–0992, 2007.

[7] N. L. Werneck, "Biblioteca para autômatos adaptativos com regras de transição armazenadas em objetos feita em c+," in *Segundo Workshop de Tecnologia Adaptativa*, 2008, pp. 22–26.

[8] R. I. d. Silva Filho, "Uma nova formulação algébrica para o autômato finito adaptativo de segunda ordem aplicada a um modelo de inferência indutiva." Ph.D. dissertation, Universidade de São Paulo, 2011.

[9] R. L. A. Rocha and J. J. Neto, "Autômato adaptativo, limites e complexidade em comparação com máquina de turing," in *Proceedings of the second Congress of Logic Applied to Technology–LAPTEC*, 2000, pp. 33–48.

[10] D. Queiroz, "Uma definição simplificada para o estudo das propriedades dos autômatos finitos adaptativos," in *Quarto Workshop de Tecnologia Adaptativa*, 210.

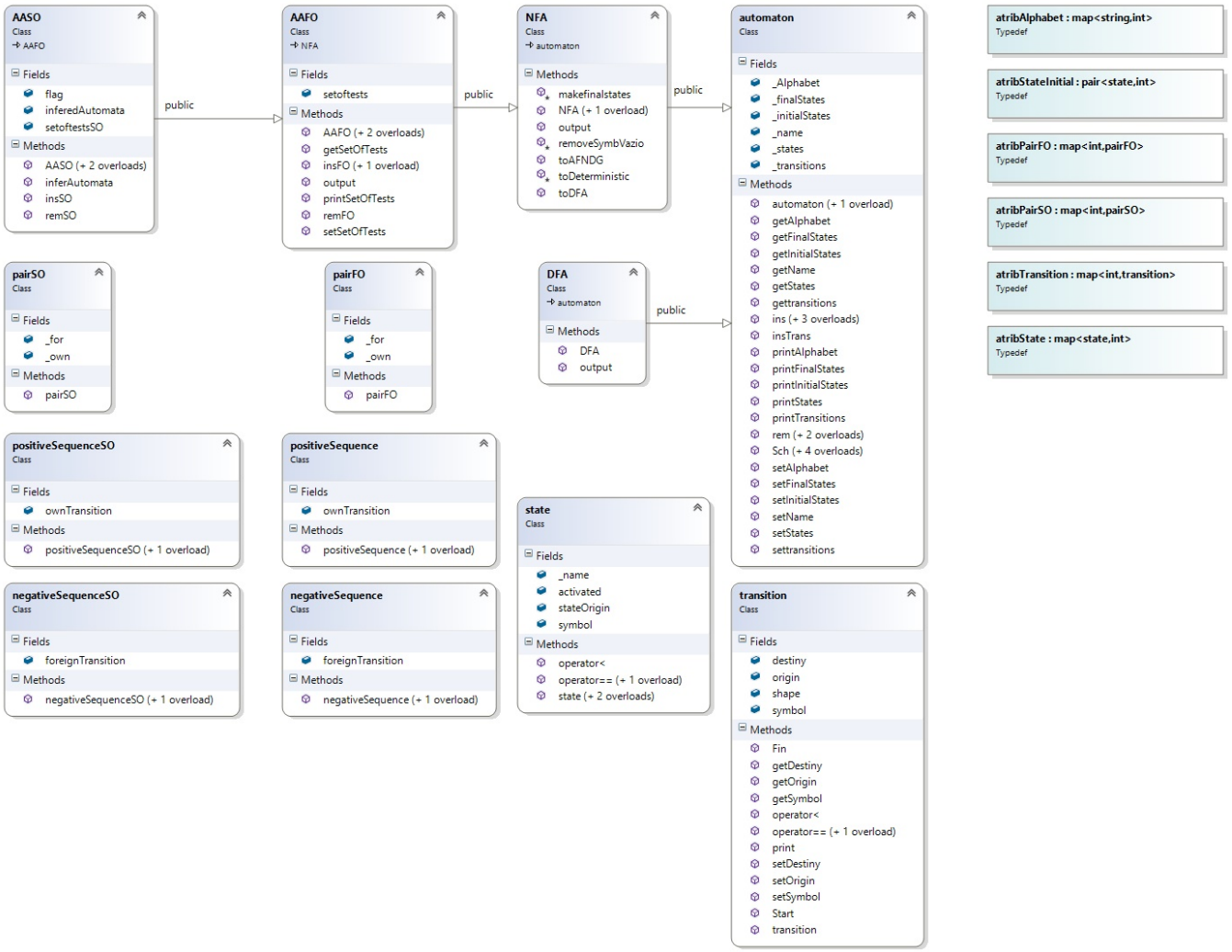

Fig. 4. Estrutura do framework.

```
#include <string>
#include <iostream>
#include <map>
#include "AASO.h"

using namespace std;

int main() {
        atribAlphabet Alfabeto;
        atribState Estados;
        atribTransition mapTransicoes;
        atribState estadosIniciais;
        atribState estadoFinais;
        Alfabeto.insert(make_pair("a", 0));
        Alfabeto.insert(make_pair("b", 1));
        Estados.insert(make_pair(state("q1"), 0));
        Estados.insert(make_pair(state("q2"), 1));
        Estados.insert(make_pair(state("q3"), 2));
        mapTransicoes.insert(make_pair(0, transition(state("q1"), "a", state("q2"))));
        mapTransicoes.insert(make_pair(1, transition(state("q2"), "a", state("q2"))));
        mapTransicoes.insert(make_pair(2, transition(state("q2"), "b", state("q3"))));
        estadosIniciais.insert(make_pair(state("q1"), 0));
        estadoFinais.insert(make_pair(state("q3"), 2));

        //map <int, transition>
        atribTransition _pro;
        atribTransition _for;

        positiveSequence transPro2;
        negativeSequence transFor2;

        atribPairFO conjuntodeComportamentosPO;
        atribPairFO auxPO;

        _pro.insert(make_pair(_pro.size(), transition(state("q3", "b", 0), "b", state("q3"))));
        _for.insert(make_pair(_for.size(), transition(state("q3", "b", 0), "b", state("qs"))));
        _for.insert(make_pair(_for.size(), transition(state("qs"), "b", state("q3"))));

        transPro2 = positiveSequence(_pro);
        transFor2 = negativeSequence(_for);

        pairFO novoPO(transPro2, transFor2);
        //pairFO novoPO;
        pair < int, pairFO > F1(1, novoPO);

        novoPO = pairFO(transPro2, transFor2);
        conjuntodeComportamentosPO.insert(F1);
        AAFO teste("AAFO", Alfabeto, Estados, estadosIniciais, estadoFinais, mapTransicoes, conjuntodeComportamentosPO);

        string entrada;
        while (cin >> entrada) {
                if (teste.output(entrada)) {
                        cout << "Aceita" << endl;
                }
                else {
                        cout << "Nao aceita" << endl;
                }
        }
}
```

Fig. 5. Exemplo de implementação de um Autômato Finito Adaptativo de Primeira Ordem que reconhece a $L1 = a^n b^n \mid n \geq 1$