Resolução Exercício 2 – Reúso de Software

Thiago Gottardi

18 de setembro de 2013

1 Diagrama de objetos

Os diagramas de estado fornecidos, ou seja, antes e após a operação de uma transformação endógena, são representados nas Figuras 1 e 2, respectivamente.

2 Código do Transformador em ATL

Nesta seção é fornecido um código ATL que permite realizar a operação de transformação exemplificada na Seção 1.

Antes de executar o código do transformador, foi requisitado no Item 4 da especificação do transformador que um modelo de entrada deveria ter uma restrição como pré-condição. Esta restrição se refere a proibir modelos que possuam duas ou mais transições que saiam de um mesmo estado possuindo um mesmo estímulo de entrada. Desta forma, pode ser executada a invariante OCL "*validacao()*" especificada na Figura 3.

Outro ponto importante é a definição de um outro metamodelo especificamente para definir parâmetros. Com isto é possível passar parâmetros para o transformador mantendo o paradigma dirigido por modelos. O metamodelo para definição dos parâmetros está representado na Figura 4.

Considerando que o modelo de entrada foi aprovado nas restrições (incluindo outras que forem necessárias apesar de não serem requisitadas neste exercício), pode-se executar o código especificado na Figura 5.

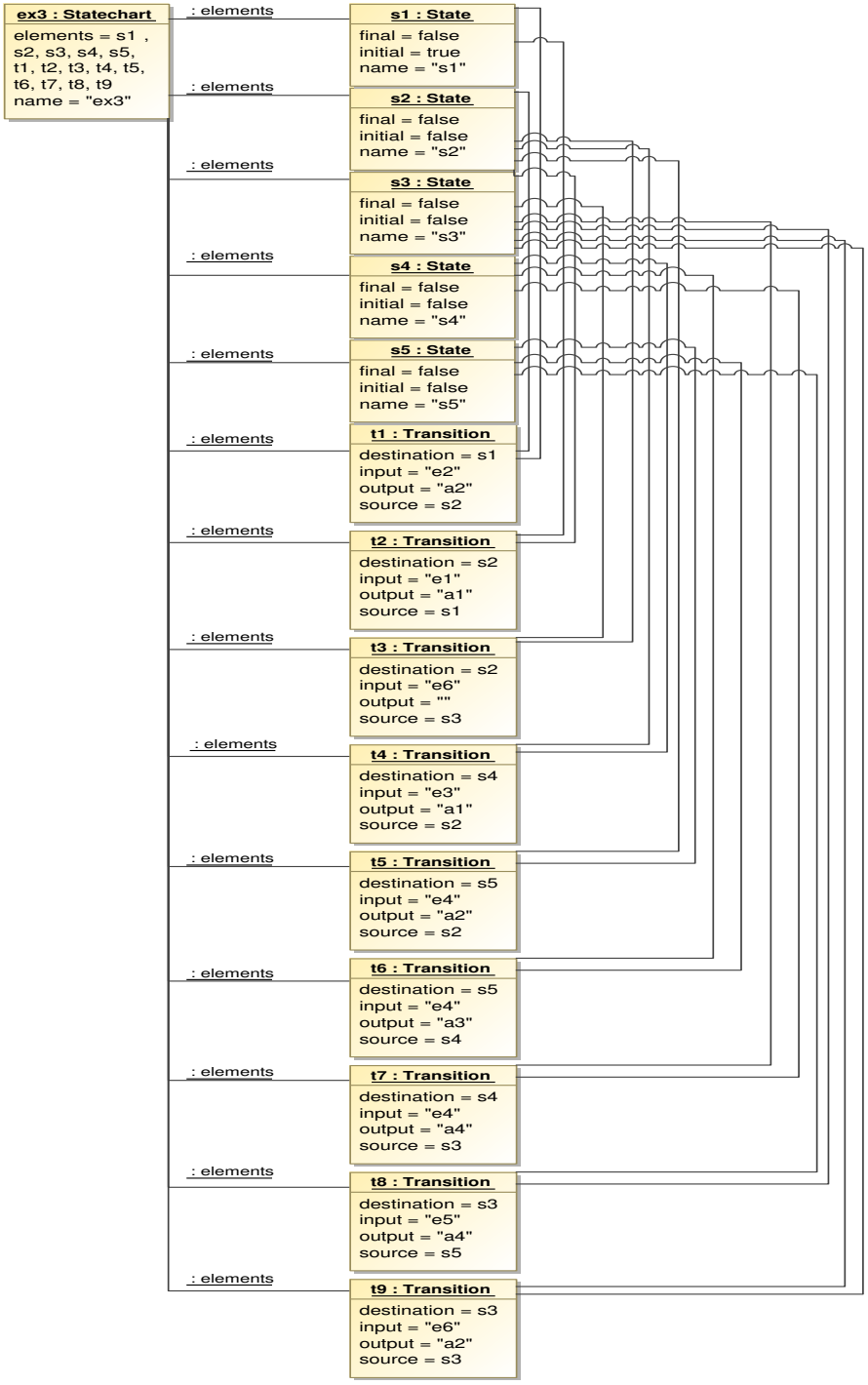

Figura 1: Diagrama de objetos para o diagrama de estados antes da transformação.

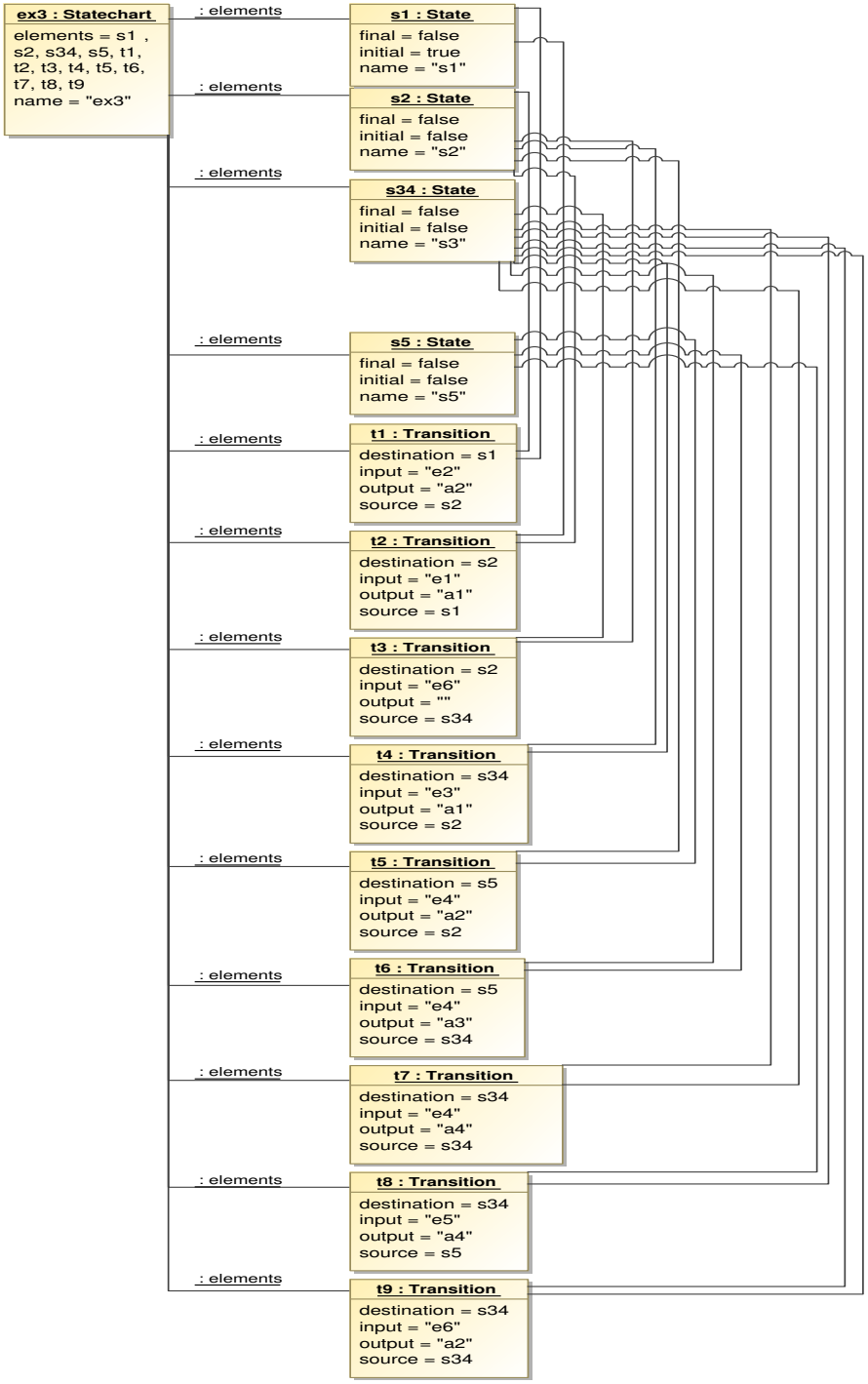

Figura 2: Diagrama de objetos para o diagrama de estados após a transformação.

context Transition **def:** preValidacao() : Boolean =
Transition.allInstances()->select(t1 | **not** (t1.source = **self**.source **implies** t1 <> **self implies** t1.input <> **self**.input)).isEmpty();

context Statechart **inv:** validacao() : Boolean =
Transition.allInstances()− >select(t | **not** t.preValidacao()).isEmpty();

Figura 3: Restrição de pré-condição.

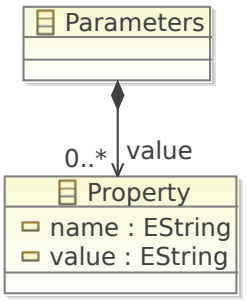

Figura 4: Metamodelo para definição de parametros.

```
-- @path state=/State/model/state.ecore
-- @path parameters=/Parameters/model/parameters.ecore

module uniao;
create OUT: state from IN: state, PAR: parameters;

helper context state!Statechart def: isStatechartDefined(): Boolean =
    not (self.oclIsUndefined());

helper context state!State def: isStateDefined(): Boolean =
    not (self.oclIsUndefined());

helper context state!Transition def: isTransitionDefined(): Boolean =
    not (self.oclIsUndefined());

--validação pedida no item 4, não deveria fazer parte do transformador.
helper context state!Transition def: preValidacao() : Boolean =
    state!Transition.allInstances()->select( t1 | not (t1.source = self.source implies t1 <> self implies t1.input <> self.input)).isEmpty();

helper def: validacao() : Boolean =
    state!Transition.allInstances()->select(t | not t.preValidacao()).isEmpty();

--verifica se o estado do contexto deve ser unido
helper context state!State def: isEstadoSelecionado(): Boolean =
    parameters!Property.allInstances()->select( p | p.value = self.name).notEmpty();

--gera a raiz e cria um estado unido vazio com valores padrão.
helper def: unido : state!State = state!State.newInstance();
rule Statechart2Statechart {
    from
        se: state!Statechart (
                se.isStatechartDefined()
            )
    to
        ss: state!Statechart (
            elements <- se.elements.including(thisModule.unido),
            name <- se.name.concat('copiado')
        )
    do
    {
        thisModule.unido.name <- 'unido';
        thisModule.unido.final <- false;
        thisModule.unido.initial <- false;
    }
}


-- transforma estados que não devem ser unidos
rule State2State {
    from
        se: state!State (
            se.isStateDefined()
            and
            (not se.isEstadoSelecionado())

        )
    to
        ss : state!State
        (
            name <- se.name,
            final <- se.final,
            initial <- se.initial
        )
}


--adiciona estados unidos no estado unido
rule State2StateUnido {
    from
        se: state!State
        (
            se.isStateDefined()
            and
            (se.isEstadoSelecionado())

        )
    to
        drop
    do {
        thisModule.unido.name <- thisModule.unido.name.concat(se.name);
        thisModule.unido.final <- thisModule.unido.final or se.final;
        thisModule.unido.initial <- thisModule.unido.initial or se.initial;
    }
}

--copia transição, no caso de origem ou destino for unido, a referência é substituída.
rule Transition2Transition {
    from
        te: state!Transition (
            te.isTransitionDefined()
        )
    to
        tSaida: state!Transition (
            source <- te.source,
            destination <- te.destination,
            input <- te.input,
            output <- te.output
        )
    do
    {
        if (tSaida.source.isEstadoSelecionado())
            tSaida.source <- thisModule.unido;
        if (tSaida.destination.isEstadoSelecionado())
            tSaida.destination <- thisModule.unido;
    }
}

```

Figura 5: Código de transformação em ATL para a operação exemplificada.