Simulando a Execução de Políticas Públicas através de Jason e CArtAgO

Iverton Adão da Silva dos Santos, Antônio Carlos da Rocha Costa

Programa de Pós-Graduação em Modelagem Computacional – Centro de Ciências Computacionais – Universidade Federal do Rio Grande (FURG)
Av Itália, Km 8 – Campus Carreiros – 96.201-900 – Rio Grande – RS – Brazil

{iverton.santos,ac.rocha.costa}@gmail.com

Abstract. This paper introduces a set of concepts and pre-structured mechanisms (API, Classes and Operations) that contribute to the research on multiagent-based simulation of Public Policies. In particular, it makes use of the Jason and CArtAgO tools, which support high-level programming of cognitive agents and virtual environments, respectively.

Resumo. O presente trabalho apresenta um conjunto conceitos e de mecanismos pré-estruturados (na forma de uma API, Classes e Operações) que contribuem para a pesquisa em simulação Políticas Públicas através de sistemas multiagentes. Em particular, utiliza as ferramentas Jason e CArtAgO, as quais fornecem suporte em alto nível para a programação de agentes cognitivos e ambientes virtuais respectivamente.

1. Introdução

Este trabalho esta incluído nos esforços do projeto Modelagem e Simulação de Políticas Públicas (MSPP) [COSTA; DIMURO, 2010], o qual espera como principal resultado poder oferecer um conjunto de conceitos, princípios metodológicos e uma sistemática para a modelagem e simulação, baseadas em agentes, de políticas públicas destinadas a contribuir para a governança de contextos socioeconômicos onde organizações (públicas e/ou privadas) operem como um sistema de institucionais formais.

Nessa perspectiva, e de uma forma mais especifica, a contribuição deste trabalho está relacionada aos aspectos instrumentais do processo de simulação, procurando fornecer uma série de mecanismos pré-estruturados (na forma de uma API, Classes e Operações) para programação daqueles planos e normas, de modo que possam ser utilizados sistematicamente para programar e simular políticas públicas utilizando sistemas multiagentes por meio das ferramentas Jason e CArtAgO.

Desta forma, este artigo está estruturado da seguinte maneira. Na próxima seção (Seção 2) discorre-se resumidamente sobre Políticas Públicas, relacionando seus principais componentes e agentes envolvidos. Na seção (Seção 3), são apresentados resumidamente aspectos da plataforma de programação de agentes Jason; na sequência (Seção 4), discorre-se brevemente sobre o *framework* de implementação de ambientes virtuais CArtAgO. Na Seção 5, considerando o objetivo central deste artigo, é desenvolvida uma proposta de mecanismos pré-estruturados para apoio ao desenvolvimento de simulações de políticas públicas. Buscando contextualizar o trabalho, a Seção 6 apresenta os trabalhos com objetivos correlatos. Por fim, são expostas algumas considerações sobre os resultados alcançados e as oportunidades futuras.

2. Políticas Públicas

2.1 Componentes de uma Política Pública

Uma política é um conjunto de princípios que orientam e/ou condicionam decisões e ações dos agentes atuantes no contexto em questão, especialmente no que diz respeito às ações de utilização dos recursos disponíveis nesse contexto [EASTON, 1965]. Por sua vez, políticas públicas são políticas relativas à utilização de recursos públicos ou sociais [HILL, 2009], os princípios determinantes de uma política refletindo os valores que a inspiraram.

Tradicionalmente, o estudo das políticas públicas se faz com base em um conceito de *ciclo de política*, envolvendo as diversas etapas por que passa a criação e operação de uma política [HILL, 2009]:

- Identificação e formulação da questão a ser tratada, no contexto socioeconômico em foco;
- Formulação e análise de soluções alternativas, isto é, das possíveis políticas para o tratamento da questão;
- Escolha de uma solução para ser implementada, isto é, escolha da política a ser implementada;
- Implementação da política;
- Avaliação dos efeitos da política e consequente revisão/reformulação da mesma.

Este trabalho foca a etapa da implementação e, mais propriamente, a subetapa de execução de uma política já implementada ou em fase de implementação.

Para tanto, consideramos que toda política é objetivada e implementada na dupla forma de um *conjunto de normas* a serem aplicadas aos agentes socioeconômicos envolvidos na questão enfocada pela política, assim como um conjunto de *planos de funcionamento* a serem seguidos pelos agentes governamentais que tem potencial de intervenção naquele contexto [COSTA, DIMURO et al., 2010].

Formalmente, dizemos então que uma política é um par Pol=(Nrms,Plns), onde Pol é a política, Nrms é o conjunto de normas e Plns é o conjunto de planos.

De acordo com [SILVA; TROTTMAN; et al, 2011] a simulação por agentes, a partir da qual é possível simular o comportamento de agentes por meio de modelos computacionais, permite evidenciar a relação custo/benefício de uma política. Isto porque a partir dessa ferramenta podem ser inseridos dados em programas que simulem o estímulo que deve ser dado aos agentes para que apresentem o comportamento esperado. Em linhas gerais, possibilita averiguar se as indicações e determinações dadas pela política terão, ou não, os reflexos previstos no contexto a que a política se refere.

2.2 Agentes Envolvidos em uma Política Pública

Há, portanto, dois componentes básicos em uma política pública que se põe em execução:

- planos, para serem realizados pelos agentes governamentais envolvidos na execução da política;

- normas (essencialmente, obrigações e proibições), para serem seguidas tanto pelos agentes componentes da sociedade, quanto possivelmente pelos agentes governamentais.

Há, assim, três tipos básicos de agentes envolvidos na execução de uma política pública:

- agentes emissores de políticas, com capacidade de expressar formalmente as políticas para os agentes envolvidos na execução da mesma (neste trabalho, consideramos um único

agente emissor de política, chamado agente *governo*);

- agentes sociais, aos quais a política se destina, com o objetivo de influir sobre seus funcionamentos, no contexto da questão enfocada pela política;

- agentes governamentais, que desempenham funções complementares à execução da política pelos agentes sociais (um tipo particular de agente governamental é o agente *fiscal*, responsável pela fiscalização do seguimento das normas pelos agentes sociais). Outro, é o agente *executor* o qual realiza um plano que pode alterar o ambiente ou promover interações como contratos de agentes sociais.

A definição dos mecanismos pré-estruturados para apoio à simulação de políticas pressupõe, então, que esses três tipos de agentes estejam presentes na simulação, conforme mostrado na Fig. 1.

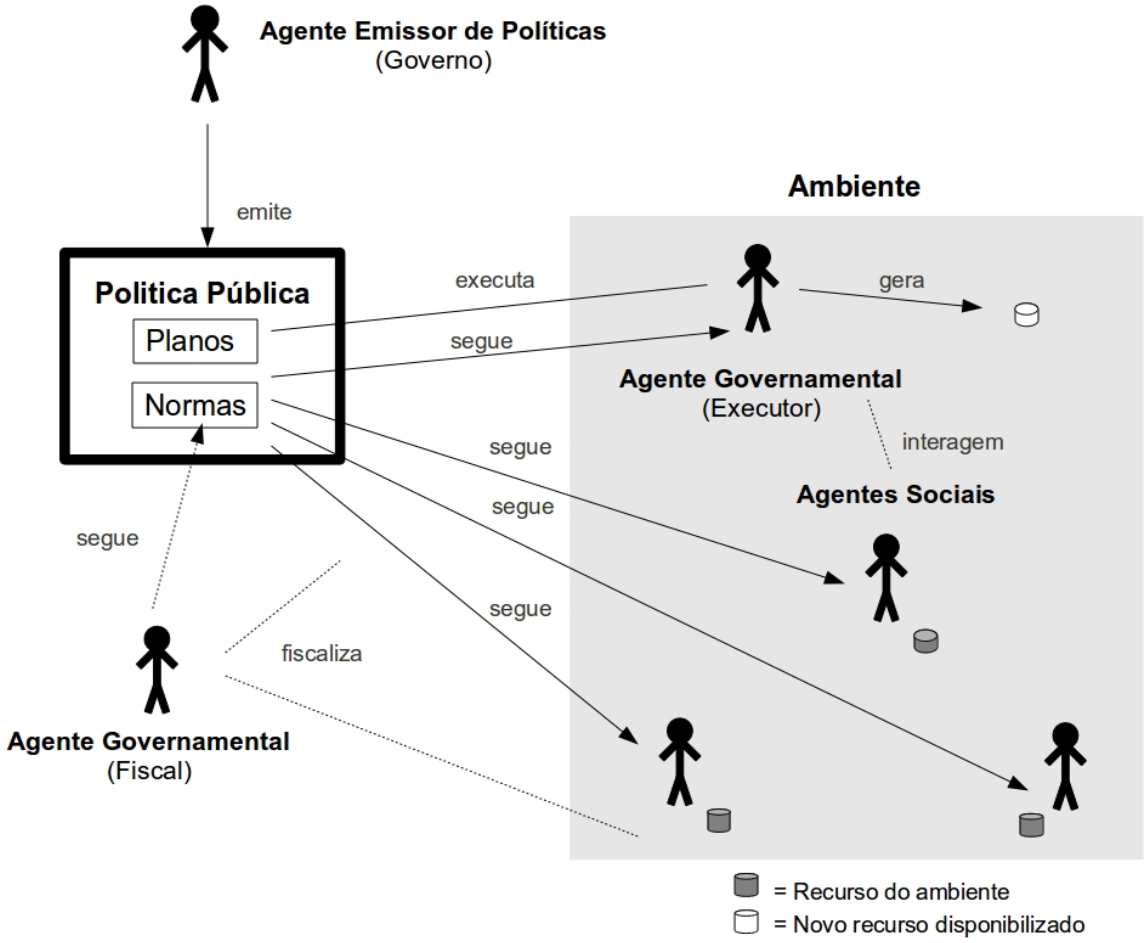

Figura 1: Fluxo dos elementos de uma política pública.

3. Jason

Jason (*A Java-based AgentSpeak Interpreter Used with Saci For Multi-Agent Distribution Over the Net*) [BORDINI; HUBNER; WOOLDRIDGE, 2007] permite a implementação de agentes cognitivos baseados na arquitetura BDI (B*elief, Desires and Intentions*) [RAO; GEORGEFF, 1992], sendo a programação realizada na linguagem AgentSpeak(L) [BORDINI; VIEIRA, 2003].

Esse ambiente inclui comunicação entre agentes baseada na teoria de atos de fala [BORDINI; VIEIRA, 2003].

Outra característica da plataforma é possibilitar que um sistema multiagente possa ser executado de maneira distribuída em uma rede com a utilização do SACI (*Simple Agent Communication Infrastructure*) [BORDINI; HUBNER; et al, 2004] ou da plataforma JADE [KRZYSZTOF; MACIEJ, 2005]. Além disso, Jason é multi-plataforma (característica herdada de sua implementação em Java) e está disponível sob licença GNU LGPL.

4. CArtAgO

CArtAgO (*Common ARTifact infrastructure for AGents Open environments*) [RICCI; SANTI; PIUNTI; et al, 2010] é um *framework* que possibilita a implementação de ambientes virtuais para sistemas de agentes. Este *framework* permite a implementação do ambiente como uma camada computacional que encapsula as funcionalidades e os serviços de *artefatos* (objetos não-autônomos) que os agentes podem explorar em tempo de execução.

Conceitualmente, o **CArtAgO** é baseado no meta-modelo *Agents & Artefacts* (A&A) [OMICINI; RICCI; VIROLI, 2008] para modelar sistemas multiagentes. Este modelo introduz uma metáfora de alto nível retirada da ideia de que humanos trabalham de forma cooperativa com o seu ambiente: *agentes* são como entidades computacionais que realizam algum tipo de tarefa orientada para alcançar um objetivo (em analogia aos trabalhadores humanos), e *artefatos* são como recursos e ferramentas dinamicamente construídas, manipuladas e compartilhadas pelos agentes para dar suporte e realizarem suas atividades individuais e coletivas (como artefatos no contexto humano). Assim, programadores de agentes podem desenvolver artefatos os quais são instanciados no ambiente e podem prover serviços para os agentes, podendo inclusive realizar uma comunicação com serviços externos do tipo *web-services*.

Os principais elementos utilizados no CArtAgO são as *propriedades observáveis*, as *operações* e os *sinais*. As propriedades observáveis servem para representar atributos de recursos do ambiente (por exemplo, cor, altura, largura), as operações são implementadas de forma a prover uma interface de utilização de um recurso (por exemplo, sendo um motor um recurso do ambiente, poder-se-ia ter operações ligar, acelerar e desligar providas aos agentes). Por fim, os *sinais* possibilitam programar que os recursos avisem aos agentes sobre uma situação, os quais podem utilizar tais sinais para tomada de decisão (por exemplo, enviar um sinal caso a temperatura ultrapasse um limite).

O **CArtAgO**, realizado em uma tecnologia *Open Source*, está disponível em http://cartago.sourgefourge.net, e se mostra ao programador do Jason através de uma API baseada na linguagem Java.

5. Mecanismos Pré-estruturados para a Simulação de Políticas Públicas

A proposta de fornecer mecanismos pré-estruturados tem por objetivo facilitar a programação de simulações de políticas públicas nas ferramentas Jason e CArtAgO, no sentido de prover interfaces que visam abstrair e encapsular uma série de elementos que são básicos para aquela finalidade.

Mais especificamente, estes mecanismos se concretizam na forma de uma API com classes, funções, artefatos, operações e planos, os quais são construídos com base nos recursos das linguagens/plataformas AgentSpeak e Java.

Nas próximas seções são apresentados detalhadamente os passos já realizados em relação a esta proposta.

5.1 Norma de Obrigação com manutenção contextual/condicional

Nesta abordagem, teve-se por finalidade viabilizar ao agente emissor de políticas, a criação de um tipo de norma de obrigação, na qual este pode estabelecer que um agente (*social* ou *executor*) tem de verificar de acordo com uma periodicidade se uma condição contextual existe, e caso exista, o mesmo terá obrigação de realizar um objetivo específico. Assim, os parâmetros de criação de uma norma de obrigação são: Id (identificador único), Tipo (neste caso obrigação), Ação, Condição e Periodicidade. Também é possível modificar valores de parâmetros estabelecidos em norma, bem como remover uma norma em vigor.

O parâmetro periodicidade também pode ser utilizado como elemento de um mecanismo de fiscalização. Assim, por exemplo, um agente com papel de fiscal poderia ter um objetivo no qual verificaria dada a passagem de tal período se a obrigação foi cumprida pelos agentes.

Em termos de implementação, a norma se dá por meio do artefato **NormaObrig** com a propriedade observável norma, no seguinte formato `defineObsProperty("norma", Id, Tipo, Acao, Condicao, Period)` e com as operações `criaNorma`, `removeNorma` e `modificaNorma`, conforme mostrado na Fig. 2.

```
public class NormaObrig extends Artifact {

@OPERATION public void criaNorma(Object Id, Object Tipo, Object Acao,
String Condicao, Object Period){
      defineObsProperty("norma", Id, Tipo, Acao, Condicao, Period);
}
@OPERATION public void removeNorma(Object Id, Object Tipo, Object Acao,
Object Condicao, Object Period){
      removeObsPropertyByTemplate("norma", Id, Tipo, Acao, Condicao,
Period);
}
@OPERATION void modificaNorma(Object Id, Object Tipo, Object Acao, Object
Condicao, Object Period){
      ObsProperty prop = getObsProperty("norma");
      prop.updateValues(Id, Tipo, Acao, Condicao, Period);
      }
}
```

Figura 2: Artefato NormaObrig.

No exemplo abaixo, estabeleceu-se que numa periodicidade de 1000 unidades de tempo, um agente tem obrigação de pagar imposto de 10%, caso sua renda esteja num intervalo entre os valores 1000,00 e 1500,00:

```
criaNorma(Id, obrigatorio, imposto(10), "renda(R) & R > 1000 &
R < 1500", 1000)[artifact_id(Id)];
```

Desta forma, através da observação de um artefato do tipo **NormaObrig** e do esquema de código da Fig. 3, um agente pode perceber a criação ou modificação de uma norma, e, consequentemente, executar um plano que realizará a manutenção da mesma de acordo com os parâmetros estabelecidos.

```
+norma(Id, Tipo, Acao, Condicao, Freq)[artifact_id(Id)]: true
<- .drop_intention(manutencao(seguir_norma(Id, _, _, _, _)));
   !manutencao(seguir_norma(Id, Tipo, Acao, Condicao, Freq)).


+!manutencao(seguir_norma(Id, Tipo, Acao, Condicao, Freq)): true
       <- !seguir_norma(Id, Tipo, Acao, Condicao);
          .wait(Freq);
          !!manutencao(seguir_norma(Id, Tipo, Acao, Condicao, Freq)).
+!seguir_norma(Id, Tipo, Acao, Condicao): condicao(Condicao) &
          Tipo == "obrigatorio"
       <- .println("seguindo norma: ", Id, Tipo, Acao, Condicao);
          .term2string(ACTION,Acao);
          !ACTION.

+!seguir_norma(Id, Tipo, Acao, Condicao): not condicao(Condicao)
       <- .println("Esta norma não abrange minha condição! ", Condicao).

condicao(Condicao) :- .term2string(COND, Condicao) &
                      .eval(X, COND) &
                      X == true.
```

Figura 3: Esquema de plano de manutenção de uma norma de obrigação.

5.2 Normas Proibitivas com penalidades e fiscalização

As normas proibitivas com penalidades foram criadas para possibilitar que o agente emissor de políticas estabeleça ações que não devem ser executados sobre um determinado recurso do ambiente, sendo que a desobediência de um agente (agente *social* ou *executor*) a tal regra implica em uma penalidade (por exemplo, uma multa). A fiscalização, ou seja, aplicação da penalidade é realizada pelo agente que tem por papel cumprir tal tarefa (o agente *fiscal*).

Assim, os parâmetros de criação de uma norma de proibição são Id (identificador único), Tipo (neste caso, proibição), Ação, Penalidade e Parâmetro da Penalidade (no caso de uma multa, o parâmetro seria o valor da mesma). Também é possível modificar valores de parâmetros estabelecidos em norma, bem como remover uma norma de proibição em vigor.

O processo de implementação da norma de proibição, semelhantemente ao caso anterior, se dá por meio de um artefato **NormaProib** com a propriedade observável norma, no entanto segue o seguinte formato `defineObsProperty("norma", Id, Tipo, Acao, Penalidade, ParPenalidade)` e com as operações `criaNorma`, `removeNorma` e `modificaNorma`, conforme Fig. 4, abaixo.

```
public class NormaProib extends Artifact {

@OPERATION public void criaNorma(Object Id, Object Tipo, Object Acao,   String
Penalidade, Object ParPenalidade) {

defineObsProperty("norma", Id, Tipo, Acao, Penalidade,ParPenalidade);
       }

@OPERATION public void removeNorma(Object Id, Object Tipo, Object       Acao,
Object Penalidade, Object ParPenalidade){
removeObsPropertyByTemplate("norma", Id, Tipo, Acao, Penalidade,  ParPenalidade);
       }
```

```
@OPERATION void modificaNorma(Object Id, Object Tipo, Object Acao,      Object
Penalidade, Object ParPenalidade){
ObsProperty prop = getObsProperty("norma");
prop.updateValues(Id, Tipo, Acao, Penalidade, ParPenalidade);
       }
}
```

Figura 4: Artefato NormaProib.

No exemplo abaixo, estabeleceu-se que é proibido a ação queimar sobre o recurso horta, sendo que a penalidade a ser aplicada em caso de desobediência será multa, com valor de 10.

```
criaNorma(Id, proibido, queimar, multa, 10) [artifact_id(horta)];
```

Por padronização, qualquer recurso implementado deve gerar uma percepção quando uma ação for invocada, sendo esta percepção num formato `+acao(ACAO, ACTOR, PAR)`. Onde os parâmetros indicam respectivamente a ação executada, quem realizou, e um valor utilizado na execução da ação (por exemplo, se ação fosse plantar teríamos a quantidade plantada). Isto pode ser feito através do recurso de *signals* do CArtAgO, no caso deste exemplo específico, na seguinte forma, `signal("acao", "queimar", getOpUserName(), 10);`

Desta forma, através da observação de um artefato do tipo **NormaProib** e do esquema de código da Fig. 5, um agente (*social* ou *executor*) pode perceber a criação, modificação ou remoção de uma norma de proibição, e, consequentemente, optar por restringir ou não seus comportamentos de acordo com tal norma. De mesma forma, um agente executando o papel de *fiscal* sobre o recurso disponibilizado, pode perceber quais são as proibições que devem ser verificadas afim de aplicar possíveis penalidades.

```
+norma(Id, Tipo, Acao, Penalidade, Pppenalidade) : Tipo == "proibido"
       <- .println("Governo criou norma: ", Tipo, ": ", Acao);
          .term2string(SAcao, Acao);
          .abolish(ok(SAcao));
          +proibido(SAcao);
          .perceive.

-norma(Id, Tipo, Acao, Penalidade, PPenalidade) : Tipo == "proibido"
       <- .println("Governo removeu norma: ", Tipo, ": ", Acao);
          .term2string(SAcao, Acao);
          .abolish(proibido(SAcao));
          +ok(SAcao);
          .perceive.
```

Figura 5: Esquema de plano para percepção de normas.

Em linhas gerais, o agente com papel de *fiscal* percebe uma ação executada no recurso e é capaz de inferir se a mesma é proibida e aplicar a penalidade, por meio do seguinte trecho de código da Fig. 6.

```
+acao(ACAO, ACTOR, PAR): e_proibido(ACAO)
<- .println("Acao: ", ACAO, "ACTOR: ", ACTOR, "PARAMETRO: ", PAR);
   ?penalidade(ACAO, PENALIDADE, PPENALIDADE);
   .println("Penalidade: ", PENALIDADE, PPENALIDADE);
   ?papel(PAPEL);
   .send(ACTOR,tell, penalidade(PENALIDADE, PPENALIDADE));
   .println("Penalidade submetida ao agente: ", ACTOR).

e_proibido(ACAO) :- .term2string(TACAO, ACAO) &
                    proibido(TACAO).
```

Figura 6: Esquema de plano para percepção de ações.

5.3 Delegação de planos para serem executados por agentes governamentais

Como visto na Seção 2, uma política estabelece obrigações e proibições para os agentes sociais e para os agentes governamentais (por meio de normas), assim como planos que devem ser executados pelos agentes governamentais. O intuito deste segundo mecanismo é, então, apoiar a simulação dos procedimentos pelos quais o agente governo delega planos de ações aos agentes governamentais.

Em termos de implementação, a sintaxe dos planos delegados aos agentes governamentais segue o formato AgentSpeak, ou seja,

```
+!evento : contexto -> corpo.
```

Assim, definiu-se um artefato denominado **Plano** com a propriedade observável `defineObsProperty("plano", Id, PlanEvent, PlanContext, PlanBody, Freq)` e as operações criaPlano e modificaPlano, conforme Fig. 7, abaixo:

```
public class Plano extends Artifact {

@OPERATION public void criaPlano(Object Id, String PlanEvent, String
PlanContext, String PlanBody, int Freq)   {
 defineObsProperty("plano", Id, PlanEvent, PlanContext, PlanBody, Freq);
}

@OPERATION void modificaPlano(Object Id, Object PlanEvent, Object
PlanContext, Object PlanBody, int Freq){
ObsProperty prop = getObsProperty("plano");
prop.updateValues(Id, PlanEvent, PlanContext, PlanBody, Freq);
       }
}
```

Figura 7: Artefato Plano.

Portanto, os parâmetros das operações representam respectivamente: identificador único do plano, evento de disparo, contexto de aplicação, corpo e uma periodicidade (caso o plano deva ser executado repetidas vezes).

A operação `modificaPlano` permite que o agente com papel de governo modifique o contexto ou o corpo de um plano já delegado, de forma que o agente responsável pela execução do mesmo passa a re-executá-lo de acordo com tais modificações.

Por fim, através da observação de um artefato do tipo Plano e do esquema de código da Fig. 8, um agente poderá perceber a criação ou modificação de um plano, e, posteriormente, executá-lo.

```
+plano(Id, PlanEvent, PlanContext, PlanBody, Freq)[artifact_id(Id)]:
existe_plano(PlanEvent)
<- .term2string(PPlanEvent,PlanEvent);
.drop_intention(executePlan(PPlanEvent,_));
.concat("+!", PlanEvent, PlanEvent1);
.relevant_plans(PlanEvent1, LP, LL);
.print("Lista de planos encontrados: ", LL);
.nth(0,LL,PlanPast);
.remove_plan(PlanPast);
.concat("+!", PlanEvent, ":", PlanContext, "<-" , PlanBody, PlanFull);
.add_plan(PlanFull);
.print("Plano ", PlanEvent , " ATUALIZADO");
!executePlan(PPlanEvent, Freq).

+plano(Id, PlanEvent, PlanContext, PlanBody, Freq)[artifact_id(Id)]:    not
existe_plano(PlanEvent)
```

```
<- .concat("+!", PlanEvent, " : ", PlanContext, " <- ", PlanBody,    PlanFull);
.add_plan(PlanFull);
.print("PlanFull: ", PlanFull);
.print("Plano ", PlanEvent ," CRIADO");
.term2string(PPlanEvent,PlanEvent);
!executePlan(PPlanEvent, Freq).

existe_plano(PlanEvent) :- .concat("+!", PlanEvent, PlanEvent1) &
.relevant_plans(PlanEvent1, LP) &
.length(LP,X) &
X > 0.

+!executePlan(PPlanEvent, Freq): true
<- .wait(Freq);
.println("Frequencia aguardada: ", Freq);
!PplanEvent;
!!executePlan(PPlanEvent, Freq).

-!FALHA: true <- println("Contexto não existente...").
```

Figura 8: Esquema para percepção de planos.

6. Trabalhos correlatos

É importante destacar que a busca por contribuir no âmbito de políticas públicas é alvo das mais diversas áreas como Administração, Economia, Ciências Sociais, Ciências Políticas etc. Contudo, o interesse aqui está em contribuir provendo recursos computacionais à mesma. Neste sentido, relacionamos alguns trabalhos na área de computação e destacamos como são suas tratativas para abordar tal tema. Nesta perspectiva, será possível contextualizar e comparar a contribuição deste trabalho de uma forma mais clara.

Um dos projetos que abrange este tema é o projeto e-POLICY [e-POLICY, 2012] o qual visa desenvolver um sistema de apoio à decisão para auxiliar os formuladores de políticas em seu processo de decisão. No mesmo, os impactos sociais são derivados a partir de dados recuperados por participação em redes sociais. Nele, tanto o decisor político quanto os cidadãos são auxiliados nas tomadas de decisões e nos processos de participação por meio de ferramentas avançadas de visualização. Além disso, o projeto e-POLICY propõe a avaliação dos impactos econômicos, sociais e ambientais da política tanto a nível global como individual.

O segundo trabalho nesta direção é o projeto denominado OCOPOMO (Open Collaboration for Policy Modeling) [OCOPOMO, 2012] o qual tem como objetivo central demonstrar que, com as técnicas adequadas, a integração da modelagem política formal, da geração de cenários e da colaboração aberta em larga escala não é apenas possível, mas essencial em todos os níveis de formação política seja local, regional, nacional ou global. Assim, busca fornecer aos governos e aos operadores de políticas públicas um ferramental para melhor dominar a complexidade no desenvolvimento de suas estratégicas políticas.

Em ambos projetos, e-POLICY e OCOPOMO, pretende-se usar um sistema multiagente para simular os efeitos das políticas analisadas sobre os sistemas sociais que elas pretendem regular.

Em [BROWN. L.; HARDING. A, 2002] apresenta-se um terceiro trabalho o qual se dá por um estudo sobre a modelagem social e políticas públicas, contudo no que se refere ao potencial e utilidade da técnica de microsimulação aplicada a políticas públicas.

Por fim, uma série de trabalhos focam em construir simuladores computacionais específicos para tratar problemas pontuais, como de políticas públicas para saúde, transito e meio-ambiente. Porém, está abordagem não é relacionada a prover recursos para suporte a políticas públicas num sentido instrumental geral.

O levantamento bibliográfico realizado mostra, portanto, que há projetos e trabalhos que procuram fornecer uma espécie de suporte computacional para o processo de elaboração de políticas públicas. No entanto, pode-se dizer que embora sendo possível relacionar estes projetos, até o momento não se encontrou na literatura (além do projeto MSPP) esforços direcionados ao desenvolvimento de padrões de projeto para simulações de políticas públicas baseadas em agentes. Igualmente, não se encontraram trabalhos voltados, especificamente, para a implementação desses mecanismos numa direção de padrões de projeto para as ferramentas Jason e Cartago, não sendo possível relacionar trabalhos com semelhanças mais diretas a esta abordagem.

7. Considerações finais

É possível citar dois resultados principais alcançados por este trabalho. O primeiro é o levantamento e definição dos principais elementos de uma política pública, no que se refere a uma perspectiva de simulação de políticas públicas em sistemas multiagentes: normas, planos, agentes sociais, agentes governamentais.

O segundo está no nível de implementação. Neste sentido, os recursos aqui apresentados podem servir como instrumentos iniciais para programação, utilizando Jason e CArtAgO, de um sistema multiagente que envolva aqueles aspectos básicos constituintes em uma política pública, mais especificamente procedimentos (planos) e elementos normativos simples (como obrigações e proibições).

Atualmente, o trabalho encontra-se na fase da definição e implementação de um método para que agentes fiscais verifiquem se as normas de obrigação e os planos estabelecidos estão sendo cumpridos, assim como foi feito para as normas de proibição, que são fiscalizadas e tem penalidade aplicada caso os agentes as descumpram.

Como continuidade, pretende-se realizar um estudo de como fazer que os mesmos mecanismos propostos possam ser utilizados dentro dos diferentes níveis organizacionais, como grupos de agentes e instituições.

Por fim, espera-se que os resultados obtidos pela concretização da proposta deste trabalho, integrados com os esforços do projeto MSPP, possam prover contribuições metodológicas e ferramentais para a simulação baseada em agentes de políticas públicas.

8. Referências

BORDINI, R. H.; HUBNER, J. F.; WOOLDRIDGE, M. **Programming Multi-Agent Systems in AgentSpeak using Jason**. University of Liverpoll: Wiley, 2007.

BORDINI, R. H.; VIEIRA R. **Linguagens de Programação Orientada a Agentes: uma introdução baseada em AgentSpeak(L)**. Revista de Informática Teórica e Aplicada – UFRGS, , 2003, V 10, N. 1, 32 p.

BORDINI, R. H.; HUBNER, J. F. et al. **Jason: a java-based agentspeak interpreter used with SACI for multi-agent distribution over the net**. Manual, first release. [S.l.], 2004.

BROWN. L.; HARDING. A.; **Social Modelling and Public Policy: Application of Microsimulation Modelling in Australia** *Journal of Artificial Societies and Social Simulation* vol. 5, no. 4 Disponível por WWW em <http://jasss.soc.surrey.ac.uk/5/4/6.html> , 2002**.**

COSTA, A. C. R.; DIMURO, G. P.; et al.; **Modelagem e Simulação de Políticas Públicas** http://mspp.c3.furg.br/ acesso em 03/2010.

EASTON, D.; **A Framework for Political Analysis**. Prentice-Hall, Englewood Cliffs, 1965.

Engineering the policy making life cycle (ePolicy) Disponível por WWW http://cress.soc.surrey.ac.uk/web/projects/59-epolicy, 2012.

KRZYSZTOF, C.; MACIEJ, G.; PAWEL, K.; MICHAL, S.; MARCIN, P. **Efficiency of JADE Agent Platform**, Scientific Programming, 2005.

HILL, M.: **The Public Policy Process**. Pearson Longman, London, 2009. (5th ed.).

OMICINI, A.; RICCI, A.; VIROLI, M. **Artifacts in the A&A meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems**, 17(3):432–456, Dec. 2008.

Open Collaboration for Policy Modeling Disponível por WWW http://www.ocopomo.eu/, 2012.

RAO, A. S.; GEORGEFF, M. P. **An Abstract Architecture for Rational Agents**. In: IN-INTERNATIONAL CONFERENCE ON PRINCIPLES OF KNOWLEDGE REPRESENTATION AND REASONING (KR'92), 3., 1992. Proceedings... Morgan Kaufmann, 1992. p.439–449.

RICCI, A.; SANTI, A.; PIUNTI, M et al. CArtAgO **(*Common ARTifact infrastructure for AGents Open environments*).** Disponível por WWW em http://cartago.sourceforge.net/ acesso em 03/2010.

SILVA, V. C.; TROTTMAN, P.; COELHO, F. S.; SARTI, M. F.; **A Abordagem de Sistemas Complexos em Administração (Pública): conceitos, agenda de pesquisa e uma aplicação na subárea de Administração/Políticas Pública(s).** XIV SemeAd Seminários em Administração, 2011, ISSN 2177-3866