UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE
ÁREA DE CIÊNCIA DA COMPUTAÇÃO

CLÁUDIO HENRIQUE CARNEIRO SAMPAIO

USO DE AGENTES NA DETECÇÃO DE FRAUDES EM IMPOSTO MUNICIPAL - ISS

SÃO LUIS
2007

CLÁUDIO HENRIQUE CARNEIRO SAMPAIO

USO DE AGENTES NA DETECÇÃO DE FRAUDES EM IMPOSTO MUNICIPAL - ISS

Dissertação apresentada como requisito parcial para obtenção de grau de Mestre em Engenharia da Eletricidade, com área de concentração em Ciência da Computação, pelo Programa de Pós-Graduação em Engenharia da Eletricidade, da Universidade Federal do Maranhão.

Orientador: Prof. Dr. Sofiane Labidi

SÃO LUÍS

2007

CLÁUDIO HENRIQUE CARNEIRO SAMPAIO

USO DE AGENTES NA DETECÇÃO DE FRAUDES EM IMPOSTO MUNICIPAL - ISS

Dissertação apresentada como requisito parcial para obtenção de grau de Mestre em Engenharia da Eletricidade, com área de concentração em Ciência da Computação, pelo Programa de Pós-Graduação em Engenharia da Eletricidade, da Universidade Federal do Maranhão.

Orientador: Prof. Dr. Sofiane Labidi

Aprovada em 22/06/2007

BANCA EXAMINADORA

_____

Prof. Dr. Sofiane Labidi (Orientador)

Universidade Federal do Maranhão (UFMA)

_____

_____

Ao meu saudoso e querido pai, Francisco Sampaio, que sempre me incentivou nos estudos e que sabia que o maior patrimônio que os pais podem deixar aos filhos é a educação.

AGRADECIMENTOS

A Deus, por ter me concedido a vida e por me guiar no meu caminho.

A meus pais, Francisco e Conceição, pelo amor, dedicação e por terem sempre acreditado em mim.

A minha irmã, Giselle, pelo carinho e pela ajuda ao longo da minha vida.

A minha querida esposa, Ana Lúcia, pelo amor, dedicação e compreensão durante as ausências no decorrer dos estudos.

Aos meus filhos, Mariana, Fabiana e Felipe, pelo carinho e por, às vezes, entenderem a minha ausência nas brincadeiras e nos passeios em família.

Ao Prof. Dr. Sofiane Labidi, pela confiança e orientação neste trabalho.

A todos os colegas e amigos que me ajudaram, de uma forma ou de outra, a concluir deste trabalho.

“Ensine a uma criança o caminho a seguir, e quando crescer ela não se desviará dele”

Provérbios 22:6

RESUMO

Este trabalho tem por objetivo principal a detecção de fraudes em impostos municipais, utilizando sistema multi-agentes. Para se atingir tal objetivo, apresentam-se algumas das fases de desenvolvimento do *software* e a implementação de alguns agentes. Propõe-se também a modelagem comportamental de contribuintes obtida a partir de dados históricos de suas ações e de funções de inferência de comportamento. Esse modelo, após validado com a utilização de dados da Prefeitura de São Luís, foi transformado em agente. Nas fases de análise e projeto do sistema, utilizaram-se as metodologias MAS-CommonKADS e a ontologia ONTOMADEM, esta baseada na MADEM (Multi-Agente Domain Engineering Methodology). Na fase de desenvolvimento, foram utilizados o ambiente JADE (Java Agent Development Framework) e os *plugins* do PROTÉGÉ para JADE, além do JESS, usado para implementar as ontologias e as regras.

Palavras-chave: Detecção de Fraudes. MAS-CommonKADS. Sistemas Multi-agentes.

ABSTRACT

This paper aims at detecting frauds in municipal taxes, using a multi-agent system. To attain such objective, it presents some of the stages of the software development and the implementation of some agents. The paper also proposes the behavior modeling of contributors, obtained from historical data from their actions and from behavior inference functions. That model, after being validated, with the use of data from São Luís County Hall, was transformed in agent. In the stages of analysis and design of the system, the methodologies MAS-Common KADS and the ontology ONTOMADEM (this one based in MADEM – Multi-Agent Domain Engineering Methodology), were used. In the stage of development, JADE (Java Agent Development Framework) ambience and the PROTÉGÉ for JADE plug-ins were used, besides JESS, used to implement the ontologies and the rules.

Keywords: Detection of frauds. MAS-CommonKADS. Multi-agent Systems.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACL	- Agent Communication Language
AIDF	- Autorização de Impressos e Documentos Fiscais
API	- Application Program Interface
CNAE	- Classificação Nacional de Atividades Econômicas
CNPJ	- Cadastro Nacional de Pessoa Jurídica
CTMSL	- Código Tributário do Município de São Luís
CTN	- Código Tributário Nacional
DDEMAS	- Domain Design for Multi-Agent System
DMS	- Declaração Mensal de Serviços
DMSII	- Data Management System II
DTS	- Data Transformation Service
FIPA	- Foundation for Intelligent Physical Agent
FIPA-OS	- Foundation for Intelligent Physical Agent - Open Source
FTP	- File Transfer Protocol
GRAMO	- Generic Requirement Analysis Method based on Ontologies
ISS	- Imposto Sobre Serviços
ISSQN	- Imposto Sobre Serviços de Quaisquer Natureza
J2EE	- Java 2 Platform Enterprise Edition
J2ME	- Java 2 Micro Edition
J2SE	- Java 2 Runtime Environment Standard Edition
JADE	- Java Agent Development Environment
JESS	- Java Expert System Shell
JVM	- Java Virtual Machine

KQML	- Knowledge Query and Manipulation Language
KSE	- Knowledge Sharing Effort
LHS	- Left Hand Side
MADEM	- Multi-Agent Domain Engineering Methodology
OWL	- Web Ontology Language
P2P	- Peer to Peer
PROLOG	- Programming in Logic
RDF	- Resource Description Framework
RDFS	- Resource Description Framework Schema
RHS	- Right Hand Side
SMA	- Sistema Multi-Agente
SQL	- Structure Query Language
TILAB	- Telecom Italy Lab
UML	- Unified Modeling Language
XML	- Extensible Markup Language

1 INTRODUÇÃO

A administração fazendária brasileira, nas esferas federal, estadual e municipal, vem estudando alternativas e a conseqüente utilização de ferramentas e processos automatizados e inteligentes para melhorar o planejamento tributário, objetivando, com isso, o aumento da arrecadação de tributos, sem, contudo, onerar as alíquotas. Nessa linha, esforços estão sendo concentrados para coibir a evasão de receita.

Os principais problemas enfrentados pelos diferentes órgãos fazendários dizem respeito a questões como planejamento da ação fiscal, inteligência fiscal e previsão, acompanhamento e análise de receita. Em razão da dificuldade em analisar e compreender grande volume de dados, é que se tem explorado e desenvolvido técnicas para extração automática de conhecimento. Contudo, um ponto a se considerar também, é quanto ao inevitável tempo transcorrido entre o cometimento de uma infração, a prestação das informações por parte do contribuinte, o processamento das informações internas, o recebimento das informações fiscais externas, o processamento conjunto de todos esses dados e a efetiva seleção e fiscalização do contribuinte infrator. O fato é que quanto maior esse tempo decorrido, maior o risco de embaraço contábil e cadastral à constituição e cobrança do crédito tributário na esfera administrativa, já que muitas vezes quando se vai notificar um contribuinte depois de decorrido um longo tempo do cometimento de uma determinada infração, este pode ter incorrido em fatos contábeis relevantes - falência, cisão, incorporação, fusão, ou ainda não ser localizado em seu domicílio fiscal. Assim, qualquer esforço no sentido de agilizar a ação do sistema de fiscalização é considerado positivo e relevante no contexto de órgãos tributários.

Então, visando a melhoria dos resultados e agilização da ação fiscal, é que este trabalho propõe o uso de agentes inteligentes e a análise comportamental na fiscalização de

contribuintes fazendários municipais, no que tange ao imposto ISS (Imposto Sobre Serviços) ou ISSQN (Imposto sobre Serviços de Quaisquer Natureza).

1.1 Justificativa

O desenvolvimento e aperfeiçoamento de técnicas de mineração de dados, análise comportamental de contribuintes e agentes específicos voltados para a fiscalização de contribuintes de impostos municipais certamente promoverá melhorias no tempo e na qualidade da fiscalização, no planejamento tributário e na recuperação de informação. Conseqüentemente, incrementará a arrecadação, permitindo ainda que as empresas possam concorrer em igualdade de condições, haja vista que a quantidade de impostos sonegados tendem a diminuir.

Por conta do exposto acima, algumas empresas privadas têm procurado desenvolver, para prefeituras, soluções que atendam a essa expectativa. Porém, as soluções apresentadas não usam o paradigma de agentes e não buscam a modelagem comportamental dos contribuintes. O uso deste dois itens permitem:

- A utilização de sistemas multiagentes pode auxiliar na eficácia dos métodos a serem utilizados, visto que agentes inteligentes constituem um novo paradigma de desenvolvimento de *software* com mecanismos apropriados para abordar a complexidade da solução a ser implementada. Com o uso de agentes cognitivos será possível a tomada de decisão baseada em uma base de conhecimento obtida a partir informações coletadas e aprendidas por especialistas, além de que permitem a análise contínua das fontes de informação, onde agentes independentes agem em uma sociedade auxiliando na obtenção de resultados de forma mais rápida;

- Com a modelagem comportamental de contribuintes será possível a predição de ações futuras baseados em dados de ações passadas, possibilitando um isto, apontar possibilidade de fraudes na verificação de desvios comportamentais.

A modelagem do sistema e a criação de uma ferramenta baseada em agentes que permita a modelagem comportamental de usuários e a detecção de fraudes no ambiente proposto permitirão que outros pesquisadores aprimorem a técnica e as metodologias aplicadas.

1.2 Objetivos do trabalho

1.2.1 Objetivo geral

O trabalho aqui proposto procura constituir uma comunidade de agentes que, cooperando entre si, consigam detectar fraudes em impostos municipais. Para que esse objetivo geral seja atingido, três objetivos específicos devem ser alcançados.

1.2.2 Objetivos específicos

São objetivos específicos deste estudo:

- Consolidar dados dos contribuintes, que consiste em extrair informações de diversas bases de dados;

- Construir perfis-padrão de contribuintes, que consiste em classificar e construir padrões de contribuintes, permitindo inferir dados de ações futuras a partir de dados de ações passadas;

- Analisar informações dos contribuintes, através da análise das informações armazenadas e, a partir de algoritmos e soluções pré-definidas, da construção da base de conhecimento do sistema e da determinação das ações e recomendações a serem tomadas.

Para que esses objetivos sejam alcançados, devem-se implementar e testar um sistema multiagentes tomando como base as informações dos contribuintes da Prefeitura de São Luís-MA.

1.3 Estrutura da dissertação

No Capítulo 2, faz-se uma revisão bibliográfica, fornecendo um embasamento teórico para o trabalho. Nesse capítulo, delimitam-se alguns assuntos como agentes inteligentes, mineração de dados, modelagens de usuários e as metodologias de desenvolvimento de *software* para sistemas multiagentes usadas.

No Capítulo 3, são aplicadas as etapas das fases de desenvolvimento das metodologias usadas para o estudo de caso. É proposta, ainda, uma arquitetura para a sociedade de agentes.

No Capitulo 4, apresenta-se a implementação do agente analisador e determinador, mostrando-se os resultados obtidos ao se aplicar o modelo proposto na Prefeitura de São Luís-MA, com o Imposto sobre Serviços de Qualquer Natureza (ISSQN ou simplesmente ISS).

Finalizando esta dissertação, o Capítulo 5 apresenta as conclusões a respeito do que foi exposto e se discute a continuidade do trabalho.

2 REVISÃO BIBLIOGRÁFICA

2.1 Conceitos de Sistemas Multiagentes

Segundo Coser [1999], um agente não pode ser facilmente definido. Por esse motivo, muitos autores concordam ao afirmar que, analisando-se as definições já produzidas, observa-se que são específicas aos modelos desenvolvidos e/ou implementados por esses mesmos autores. Assim, os agentes apresentam características e propriedades que são particulares ao trabalho de cada autor, além de outros que são comuns a todos os autores.

Usando o conceito apresentado por Girardi [2004, p.1], pode-se afirmar que "Um agente é uma entidade autônoma que percebe seu ambiente através de sensores e age sobre o mesmo utilizando-se dos executores".

Um agente, segundo Faria [2004], é qualquer entidade que possui autonomia para chegar a uma conclusão, tomar decisões e executar tarefas sem a intervenção humana ou de outros sistemas. Girardi [2004] expressa que a prática do desenvolvimento de *softwares* baseados em agentes está se popularizando devido às facilidades que esse paradigma oferece para abordar a complexidade do *software* e à disponibilidade de metodologias para a construção de aplicações segundo esse paradigma.

Nesse contexto, observa-se que a abstração de um componente de *software* não é mais um simples objeto passivo – uma entidade com uma memória e um comportamento próprio; agora é um agente – uma entidade (física ou abstrata) com autonomia, habilidade social e capacidade de aprender. Assim, o agente se constitui numa abstração adequada para se compreender, desenvolver e usar efetivamente os novos sistemas de informação, caracterizados pela sua complexidade [SODRÉ, 2002].

Nos modelos reativos [BOUNABAT, 2001], utilizam-se agentes que não possuem um modelo de raciocínio e agem com base em respostas a estímulos externos. Já os agentes cognitivos ou deliberativos [WOOLDRIDGE, 2002] possuem um modelo simbólico de raciocínio e um plano a ser realizado e/ou negociado com outros agentes para alcançar seus objetivos.

Muitas vezes, porém, um só agente não é capaz de resolver a maioria dos problemas. Portanto, em sistemas baseados em agentes, comumente se faz necessária a utilização de diversos agentes atuando de forma integrada num ambiente.

O sistema multiagente (SMA) consiste em coordenar o comportamento inteligente de um conjunto de agentes autônomos cuja existência pode ser anterior ao surgimento de um

problema em particular. Nos SMAs, os agentes constituem a parte central que deve cooperar e trocar conhecimento para se obter a solução de problemas antes desconhecidos.

Para Girardi [2004], um SMA pode ser caracterizado como um grupo de agentes que atuam em conjunto, no sentido de resolver problemas que estão além das suas habilidades individuais. Os agentes realizam interações entre si, de modo cooperativo, com o intuito de atingir uma meta.

Os SMAs diferem de sistemas com um único agente pelo fato de serem constituídos por vários agentes, os quais modelam os objetivos e as ações dos demais. Nesse ambiente, em que vários agentes e serviços comuns são executados, deve-se padronizar a forma de comunicação.

Os SMAs são também conceituados como uma especificação formal de um modelo abstrato que contextualiza o mundo real. Com isso, possibilita a padronização de um vocabulário assegurado, no qual os termos escolhidos sejam suficientes para especificar e definir conceitos, além de permitir relacionamentos adequados a partir da escolha terminológica realizada, pois inclui a expressão exata do domínio específico do conhecimento.

2.2 Formas de Comunicação no Ambiente Multiagentes

A comunicação é, sem duvida, o aspecto básico de qualquer SMA. Num ambiente em que vários agentes e vários serviços são oferecidos à padronização, permite-se que os agentes troquem informações que servirão de base para coordenar suas ações e realizar cooperação. Para padronizar a forma de comunicação, é preciso compartilhar vocabulários de palavras e significados, ou seja, faz-se necessária uma ontologia.

Existem diversas maneiras de os agentes trocarem informações uns com os outros em sistemas multiagentes, dentre as quais:

- **Comunicação direta** – cada agente se comunica com qualquer outro agente sem intermediário;
- **Federado** – neste modelo, uma estrutura hierárquica de agentes é estabelecida e a troca de mensagens se dá através de mediadores ou facilitadores;
- ***Broadcast*** – é utilizada em situações em que a mensagem deve ser enviada para todos os agentes do ambiente ou quando o agente remetente não conhece o agente destinatário nem seu endereço. Em suma, todos os agentes recebem a mensagem enviada;
- ***Blackboard* ou quadro-negro** –bastante usada na Inteligência Artificial como modelo de memória compartilhada, nada mais é que um repositório no qual agentes escrevem mensagens a outros agentes e obtêm informações sobre o ambiente.

2.3 Protocolo de comunicação entre agentes

A Linguagem de Comunicação de Agente (*Agent Communication Language* – ACL) é o padrão de comunicação estabelecido pela FIPA (*Foundation for Intelligent Physical Agent*) [FIPA, 2005], sendo esta uma organização que busca estabelecer padrões para a interoperabilidade de agentes e de *softwares* heterogêneos.

2.3.1 KQML

A KQML (*Knowledge Query and Manipulation Language*) é uma linguagem na qual os formatos de mensagens são denominadas performativas, contendo um protocolo de comunicação de alto nível para troca de mensagem independente de conteúdo e da ontologia aplicável. Ela foi desenvolvida pelo KSE (*Knowledge Sharing Effort*).

A KQML é uma linguagem dividida em três camadas: a do conteúdo, a da mensagem e a da comunicação. O conteúdo se refere ao que está contido na mensagem, a qual

toda implementação KQML ignora, exceto para determinar as delimitações desse conteúdo [FININ et al.,1997].

As sintaxes do ACL e do KQML são muito parecidas. Embora a ACL seja o padrão estabelecido pela FIPA, o KQML tem se mostrado o padrão de fato para boa parte das implementações.

2.3.2 FIPA-ACL

A FIPA-ACL é uma linguagem baseada em ações de fala, como o KQML. A sua sintaxe é bastante semelhante à do KQML, porém o conjunto de performativas (atos comunicativos) é diferente. As especificações FIPA sobre comunicação de agentes possuem três partes, que são:

1. *FIPA Interaction Protocols* (IPS): tratam de protocolos de troca de mensagem pré-estabelecidos para mensagens ACL;

2. *FIPA Communicative Act* (CA): especificações que tratam das diferentes expressões (*utterances*) para mensagens ACL;

3. *FIPA Content Language* (CL): especificações que tratam das diferentes representações dos conteúdos das mensagens ACL. A especificação completa do padrão de comunicação está disponível em http://www.fipa.org/specs/aclpecs.tar.gz [FIPA, 2005].

2.4 Plataforma JADE (*Java Agent Development Environment*)

Atualmente, existem várias plataformas para o desenvolvimento de agentes. Neste trabalho, porém, será usada apenas a plataforma JADE, pois esta fornece um *framework* e ferramentas que ajudam na criação de agentes, juntamente com uma estrutura que facilita sua implementação.

O JADE é um *middleware* desenvolvido pelo TILAB (Telecom Italia Lab) para o desenvolvimento de aplicações baseadas em multiagentes distribuídos, usando arquitetura de comunicação ponto-a-ponto (P2P).

Conforme Poggi, Rimassa e Turci [2000], JADE é uma arquitetura de *software* implementada para tornar fácil o desenvolvimento de aplicações de agentes de acordo com as especificações da FIPA-OS (*Foundation for Intelligent Physical Agents – Open Source*) para interoperabilidade de sistemas multiagentes inteligentes. JADE utiliza um modelo de agentes e é implementado em JAVA, o que oferece uma boa eficiência e propicia a reutilização de *software*.

Poggi, Rimassa e Tomaiuolo [2001] acrescentam que JADE suporta a maior parte da infra-estrutura relatada nas especificações da FIPA-OS, como protocolos de transporte, codificação de mensagens e páginas brancas e amarelas de agentes. Possui, além disso, várias ferramentas para fácil gerenciamento e *debugging* de agentes.

O JADE é baseado em alguns princípios básicos:

- Interoperabilidade: segue os padrões da FIPA, permitindo interação com outros agentes não desenvolvidos em JADE;

- Uniformidade e portabilidade: JADE provê um conjunto de APIs que são independentes da camada de rede e versão JAVA (J2EE, J2SE e J2ME);

- Facilidade de uso.

2.4.1 O Modelo Arquitetural

O JADE possui as classes do JAVA necessárias para o desenvolvimento de aplicações agentes e o ambiente de execução que provê os serviços básicos e que deve estar ativo no dispositivo antes de o agente ser executado. Cada instância do JADE executor[1] é chamada de *container*. O conjunto de *containers* é chamado de plataforma e provê uma

[1] Tradução livre para *JADE run-time*.

camada homogênea que esconde do agente a complexidade e a diversidade das camadas inferiores (*hardware*, sistema operacional, tipo de rede, JVM). A Figura 1 mostra a arquitetura JADE.

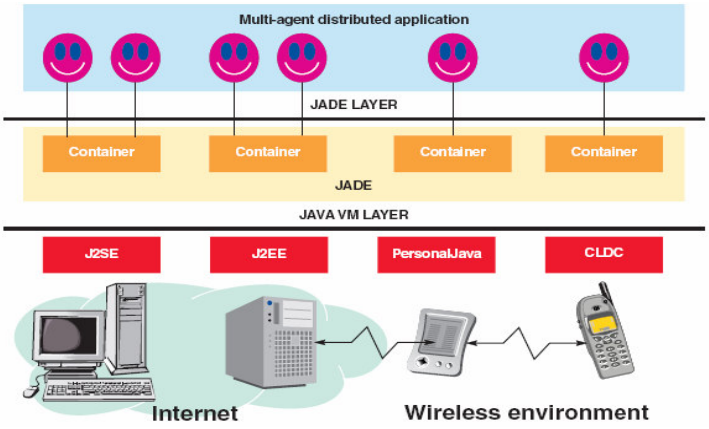


Figura 1. Arquitetura JADE.

O JADE já está sendo testado em ambiente de comunicação via rádio de diversos fabricantes, dentre os quais: Nokia, Motorola, Siemens, Palm, Compaq, Psion, HP.

2.4.2 Modelo funcional

JADE permite a cada agente descobrir dinamicamente outros agentes e se comunicar com eles de acordo com o paradigma ponto-a-ponto. Do ponto de vista da aplicação, cada agente é identificado por um único nome e provê um conjunto de serviços. Ele pode registrar e modificar esses serviços e/ou pesquisar nos agentes por determinados serviços. Pode, ainda, controlar o seu ciclo de vida e, em particular, comunicar-se com todos os outros pontos (agentes).

A comunicação entre os agentes é feita usando a troca assíncrona de mensagens. Não existe qualquer dependência temporal entre a comunicação dos agentes, isto é, o receptor e o emissor podem não estar disponíveis ao mesmo tempo.

Apesar deste tipo de comunicação, a segurança é preservada, desde que a aplicação a requeira. JADE provê um mecanismo próprio de autenticação e verifica direitos assinalados para os agentes. Quando necessário, uma aplicação pode verificar a identidade de quem enviou a mensagem e prevenir ações não permitidas.

Para facilitar a criação e o manuseio dos conteúdos das mensagens, JADE provê suporte para a conversão automática, em mão dupla, num conjunto de formatos que inclui XML, RDF e objetos JAVA, além de permitir a integração com algumas ferramentas de criação de ontologias, como o Protégé.

JADE é "opaco" (versão intermediária entre as *black box* e *white box*), ou seja, se inferências são necessárias para uma determinada aplicação específica, a ferramenta permite que programadores a reutilizem em seus sistemas preferidos. O JADE já está integrado e testado com o JESS e o PROLOG, neste caso para permitir a criação de agentes deliberativos, visto que o JAVA não é uma linguagem baseada em lógica. Para permitir escalabilidade, JADE pode ser executada em processamento paralelo (*multi-thread*).

Outra característica interessante do JADE é que, nos ambientes J2SE e Personal JAVA, existe o suporte e a **mobilidade de código** e de **estado de execução**. Isso explica o fato de um agente ter a execução paralisada num *host*, migrar para um diferente *host* remoto (sem a necessidade de ter um código agente já instalado nesse *host*) e reiniciar a execução no mesmo ponto em que ele foi interrompido.

A plataforma também inclui os serviços de nomes (garantindo que cada agente possua um nome único) e o serviço de páginas amarelas, que pode ser distribuído através de múltiplos *hosts*. Grafos federativos podem ser criados em seqüências para definir domínios estruturados de serviços de agentes.

2.4.3 PLUG-INS do JADE para o PROTÉGÉ

O *Beangenerator* (http://acklin.nl/page.php?id=34) pode gerar arquivos JAVA representando uma ontologia e pode ser usado com o JADE *Tookit* 3.1. O *Beangenerator* é implementado como um *plug-in* para o Protégé, através do qual se pode importar e exportar extensões OWL, RDF, RDFS.

Com a ferramenta *Beangenerator*, pode-se gerar ontologias de acordo com o padrão FIPA / JADE, a partir de RDF(s), XML, OWL e projetos do Protégé.

A Figura 2 mostra a geração de ontologias em JAVA a partir do Protégé.

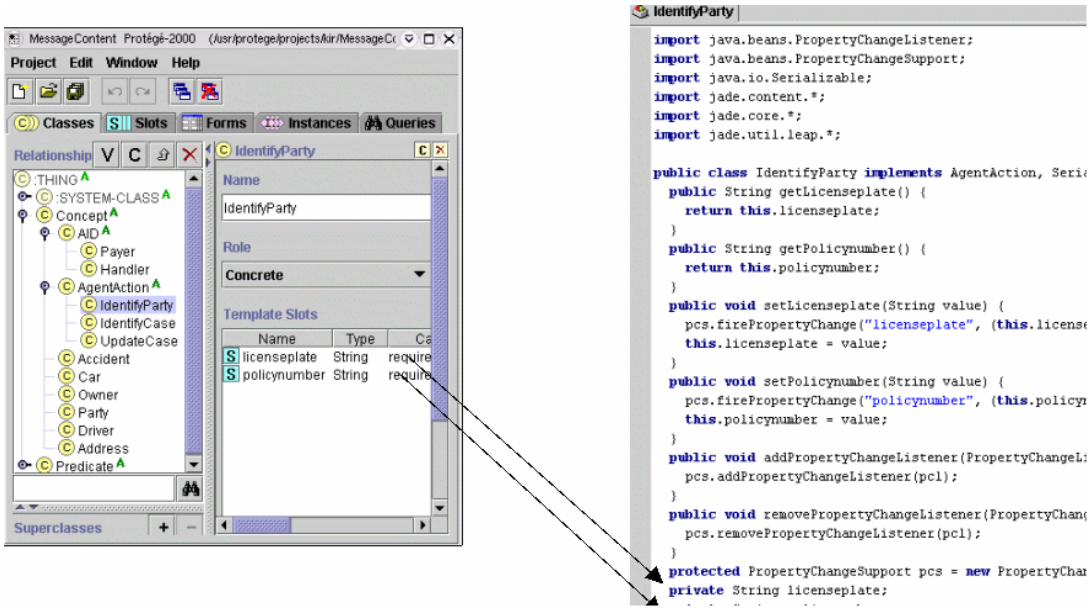


Figura 2 – Geração de ontologias em JAVA, a partir do Protege, usando-se o *Beangenerator*.

A Figura 3 apresenta a ferramenta incorporada ao Protégé.

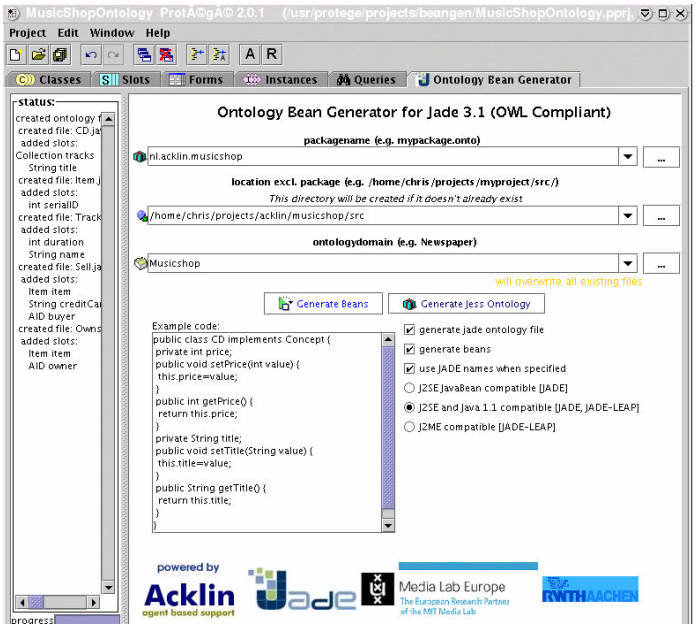


Figura 3 – *Beangenerator* incorporada ao Protégé.

2.5 Modelagem comportamental de usuários

A modelagem comportamental de usuários tem recebido a atenção dos pesquisadores há vários anos [PERRAULT, 1978; FININ, 1986; KOBSA, 1989; KOBSA, 1993; KOBSA, 2001], e foi a necessidade de se criarem sistemas que se adaptassem automaticamente aos seus usuários que mais contribuiu para o seu desenvolvimento. Os esforços se concentram tanto na criação de um modelo adequado quanto na evolução desse modelo, que deve ocorrer em paralelo às alterações comportamentais do usuário.

Diversas são as aplicações para tal modelagem, destacando-se: a interação homem-computador; interfaces inteligentes; engenharia cognitiva; recuperação inteligente de informação; sistemas tutores inteligentes; sistemas especialistas; e sistemas de simulação [KOBSA, 1993].

No âmbito tributário, a simulação de processos através de sistemas computacionais é reconhecidamente algo bastante eficaz para um bom planejamento fiscal. Nesse contexto, a administração fazendária brasileira, para fazer face às suas necessidades de incremento de

receitas próprias, vem estudando alternativas para aumentar a arrecadação de tributos, como mostram alguns trabalhos que tratam de aspectos como classificação e seleção de contribuintes para fiscalização e modelos de gerenciamento de arrecadação utilizando técnicas estatísticas [BARRETO, 2005; CORVALÃO, 2002; LIEBEL, 2004; BRAZ, 2001].

Este trabalho utiliza, na construção do agente modelador, um modelo comportamental de usuários para contribuintes de sistemas fiscais baseado em Nogueira [2006], cujo objetivo principal é construir um modelo computacional que permita inferir dados de ações futuras do usuário a partir de dados de ações passadas desse mesmo usuário. Ao contrário das demais abordagens, esse modelo pode ser aplicado tanto a grandes quanto a pequenas e médias massas de dados, pois permite que o comportamento do contribuinte seja tratado de forma individual.

A principal vantagem do modelo comportamental proposto é a sua capacidade de aplicação a usuários individuais, levando a um macro-modelo composto de micro-modelos individuais de contribuintes. Dessa forma, com amostras relativamente pequenas, pode-se fazer inferências de comportamento bastante úteis.

Basicamente, modelar o usuário significa adquirir seu modelo, representar o seu conhecimento e atualizá-lo. A aquisição desse modelo pode ser feita de forma explícita (o usuário informa como ele é), implícita (o sistema verifica como o usuário é) ou híbrida (combinando as duas formas anteriores) [KOBSA, 1999; SHARMA, 2001]. Um modelo genérico de um usuário, conforme proposto por Silva [2003], levado ao mundo computacional, pode ser representado como o conjunto:

$$M = \{Vei, Pp, Vei(t)\}. \quad \textbf{(1)}$$

Sendo:

- **Vei** é um vetor de atributos que definem uma fotografia do usuário num instante determinado (estado interno). É onde são armazenadas as propriedades dos

usuários relacionadas com seus objetivos, motivações e situação atual, podendo ser interpretado como a mente do usuário;

- **Pp** é um pacote de procedimentos de interação do usuário com o ambiente externo (percepção). Esse pacote é para que os usuários possam interagir com um ambiente de forma autônoma, decidindo qual ação adotar a cada momento;
- **Vei(t)** corresponde a uma função que determina o vetor **Vei** num instante **t**, obtida através de técnicas que permitam prever valores futuros com base em valores passados (comportamento). Isso leva à identificação e análise das ações que serão reproduzidas pelos usuários.

Para se definir alguns dos atributos do vetor **Vei,** referencia-se ao trabalho de Segura [2006]. Segura propõe que a decisão do contribuinte de pagar ou não um imposto resulta da comparação entre custos e benefícios obtidos pela evasão. O benefício coincide com a quota devida, caso o contribuinte deixe de pagar o imposto; e o custo está representado pelo valor do risco assumido nesse caso, sendo equivalente ao produto da quantidade que o contribuinte teria que pagar se fosse descoberto, multiplicado pela probabilidade caso isso ocorra, ou seja:

- CUSTO = (TRIBUTO DEVIDO + SANÇÃO) X PROBABILIDADE DE SER DESCOBERTO
- BENEFÍCIO = TRIBUTO DEVIDO

De acordo com esse modelo, se o benefício for considerado superior ao custo, o contribuinte provavelmente decidirá não pagar o imposto. Assim sendo:

- SE BENEFÍCIO > CUSTO $\Rightarrow$ FRAUDE

Então, um modelo dessa natureza deve conter basicamente três variáveis: valor do imposto (**Vi**), probabilidade de ser descoberto (**PSD**) e sanções aplicadas (**Vs**).

O valor do imposto determina o benefício imediato que o contribuinte obteria, caso decidisse evadir-se.

A segunda variável considerada – a probabilidade de ser descoberto (PSD) – pode ser associada à probabilidade de o contribuinte ser auditado, medida pela relação existente entre o número de auditorias realizadas e o número de contribuintes.

A última variável considerada no modelo são as sanções. Em seu tratamento analítico, as sanções e a PSD aparecem como variáveis intercambiáveis. Formalmente, pois, a Administração Tributária poderia decidir gastar menos reduzindo a PSD e compensá-la com um aumento no nível das sanções.

Levando-se em consideração o modelo genérico de usuários abordado anteriormente e as variáveis tributárias discutidas por Segura [2006], o conjunto modelo **M** de um contribuinte é dado por

$$M = \{Veic, Ppc, Veic(t)\} \quad (\mathbf{2})$$

$$Ppc = \{\text{“perceber PSD geral”}\}. \quad (\mathbf{3})$$

$$Veic(t) = \{E_f(t), E_c(t), V_{if}(t), V_{ic}(t), PSD(t), Vs(t)\}. \quad (\mathbf{4})$$

Nesse modelo, os itens $_{\mathbf{f}}$ **e** $_{\mathbf{c}}$ correspondem aos valores das variáveis percebidos pelo fisco e declarados pelo contribuinte.

A Equação 4 é composta dos seguintes elementos num instante **t** no futuro:

- $\mathbf{E_c}$ é o somatório dos serviços declarados pelo contribuinte ao fisco;
- $\mathbf{E_f}$ é o somatório dos serviços realizados pelo contribuinte e percebidos pelo fisco;
- $\mathbf{V_{if}}$ é o imposto calculado sobre os serviços percebidos pelo fisco;
- $\mathbf{V_{ic}}$ é o imposto calculado sobre os serviços declarados pelo contribuinte;
- **PSD** é a probabilidade de o contribuinte ser descoberto, em caso de omissão de informações corretas;

- **Vs** corresponde ao valor da sanção aplicada pelo fisco ao contribuinte, em caso de detecção de fraude.

Diferenças de valores entre os atributos $\mathbf{E_c}$ e $\mathbf{E_f}$ indicam que existe fraude para tal contribuinte. A **PSD** é, dessa forma, diretamente proporcional às variações de valores **ΔE**, na maioria dos casos [SEGURA, 2006]. O valor da sanção aplicada corresponde ao somatório dos valores da variação $\mathbf{\Delta V_i}$ ($\mathbf{V_{if} - V_{ic}}$) com a penalidade aplicada de acordo com a legislação específica de cada imposto [CTN, 1966].

A utilização de técnicas de *forecasting* (previsão), ou seja, técnicas utilizadas quando da existência de um histórico de dados, mostra-se muito atraente nesse caso. Isso porque, de posse do histórico dos valores de $\mathbf{E_c}$ e $\mathbf{E_f}$, $\mathbf{V_{if}}$ e $\mathbf{V_{ic}}$, **PSD** e **Vs**, pode-se inferir valores futuros de tais dados, obtendo, em determinado instante, o vetor comportamento do contribuinte.

A modelagem de percepções do agente contribuinte pode ser simplificada pela capacidade do contribuinte de adaptar sua variação de valor do imposto pago ($\mathbf{\Delta V_i}$) em relação à **PSD** de outros contribuintes.

Conhecendo-se o comportamento de contribuintes individuais, pode-se chegar ao comportamento global de uma determinada entidade fazendária. Isso pode ser obtido através do somatório dos comportamentos individuais, ou seja, pode construir o macro-modelo dos contribuintes. Dessa forma, a Equação 4 pode ser representada como:

$$Ves(t) = \{\sum E_f(t), \sum E_c(t), \sum V_{if}(t), \sum V_{ic}(t), \sum PSD(t)/n, \sum Vs(t)\}. \qquad (\mathbf{5})$$

onde **n** é o número de contribuintes modelados.

O tamanho do grupo de contribuintes (**n**) pode ser determinado pela identificação do subgrupo e das características-chave dos contribuintes desse grupo, como, por exemplo, identificando-os por grupo econômico, de acordo como a Classificação Nacional de Atividade Econômica (CNAE), por faixas de faturamento, etc., fazendo uma analogia com a

representação do conhecimento dos usuários através de estereótipos [RICH, 1979; RICH, 1983; BALLIM, 1991; CHIN, 1988].

O modelo comportamental em si pode ser simplificado pela utilização de séries temporais para cada atributo pertencente ao vetor **Veic**. Assim sendo, um método de previsão de valores futuros baseado em séries históricas pode ser utilizado.

Uma forma bastante atraente, devido à sua simplicidade e aos bons resultados obtidos em previsões de séries históricas, é aquela representada pelos modelos de suavização exponencial, em especial o modelo de Holt. Tais métodos usam uma ponderação distinta para cada valor observado na série temporal, de modo que valores mais recentes recebam pesos maiores. Assim, os pesos formam um conjunto que decai exponencialmente a partir de valores mais recentes.

O modelo de Holt emprega duas constantes de suavização, α e β (com valores entre 0 e 1), sendo representado por três equações [ARMSTRONG, 1999]:

$$L_t = \alpha z_t + (1-\alpha)(L_{t-1} + T_{t-1}) \quad (\mathbf{6})$$

$$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1} \quad (\mathbf{7})$$

$$Z'_{t+k} = L_t + kT_t. \quad (\mathbf{8})$$

As Equações 6 e 7 fazem uma estimativa do nível e da inclinação da série temporal, respectivamente. Já a Equação 8 calcula a previsão de valores para os próximos **k** períodos.

Nessas equações, $\mathbf{L_t}$ e $\mathbf{T_t}$ correspondem, nessa ordem, ao nível e à inclinação da série temporal no instante **t**, enquanto $\mathbf{L_t\text{-}1}$ e $\mathbf{T_t\text{-}1}$ se referem ao nível e à inclinação da série no instante **t-1**. $\mathbf{Z'_{t+k}}$ corresponde ao valor da previsão feita para **k** períodos após o instante **t**.

Outra forma de se prever valores em séries históricas são os Modelos de Redes Neurais Artificiais (RNA) [VARFIS, 1990].

2.6 Descrição das Metodologias Utilizadas

2.6.1 MAS-CommonKADS

A MAS-CommonKADS é uma metodologia de desenvolvimento de *softwares* para sistemas multiagentes. Ela estende a metodologia da engenharia do conhecimento CommonKADS com técnicas de orientação a objeto e metodologia de engenharia de protocolo [IGLESIAS et al., 1998]. Propõe as seguintes fases ou ciclos para o desenvolvimento de sistemas multiagentes:

- *Elicitação ou conceitualização*: fase de aquisição de conhecimento visando a uma primeira descrição do problema. Nessa etapa, os casos de uso podem ser determinados seguindo a UML;
- *Análise*: fase de identificação dos requisitos do sistema com base no enunciado do problema e de construção dos modelos de organização, tarefas, agentes, comunicação, coordenação e experiência;
- *Projeto*: caracterizada pela construção do modelo de projeto;
- *Codificação e teste dos agentes*: implementação do sistema e teste individual dos agentes;
- *Integração*: teste do sistema como um todo;
- *Operação e manutenção*: colocação do sistema em uso.

A fase de análise do MAS-CommonKads consiste em sete modelos:

- *Modelo de agentes*: especifica características de agentes, como capacidade de raciocínio, sensores / atuadores, serviços, grupos de agentes e hierarquias;
- *Modelo de tarefas*: descreve as tarefas que os agentes podem realizar, como, por exemplo, objetivos, decomposição, métodos de resolução de problemas, etc.;

- *Modelo de experiência (expertise model)*: define o conhecimento necessário, ou seja, modela a capacidade de raciocínio do agente para executar suas tarefas e atingir seus objetivos;
- *Modelo de organização*: descreve a organização social da sociedade do agente;
- *Modelo de coordenação*: ilustra a conversação entre agentes, ou seja, descreve o relacionamento dinâmico entre os agentes de *software*;
- *Modelo de comunicação*: detalha a interação agente-humano-*software*, descrevendo o relacionamento dinâmico entre os agentes humanos e seus respectivos agentes de *softwares*;
- *Modelo de projeto*: corresponde à fase de projeto. Verifica, também, aspectos relevantes da rede de agentes, selecionando a melhor arquitetura e a plataforma de desenvolvimento.

2.6.2 Metodologia MADEM

A MADEM (Multi-Agent Domain Engineering Methodology) [FARIA, 2004] é uma metodologia que visa facilitar a execução das fases de análise e projeto na Engenharia de Domínio de Multiagentes. Atualmente, a MADEM integra uma técnica para análise de domínio denominada GRAMO (Generic Requirement Analysis Method based on Ontologies) (Tabela 1), e uma outra técnica de projeto de domínio denominada de DDEMAS (Domain Design for Multi-Agent System) (Tabela 2).

Tabela 1 – Fase, atividade e produto da técnica GRAMO

TÉCNICA GRAMO	Fases	Atividades		Produtos
	Modelagem de Domínio	Modelagem de Conceitos	Modelagem de Objetivos	Modelo de Domínio (Modelo de Conceitos, Modelo de Objetivos, Modelo de Papéis e Modelo de Interações)
			Modelagem de Papéis	
			Modelagem de Interações	
		Modelagem de Variabilidades		
	Modelagem de Usuários	Aquisição		Modelo de Usuários
		Representação		
		Manutenção		

Tabela 2 – Fase, atividade e produto da técnica DDEMAS

Fases	Tarefas		Produtos
Modelagem de agentes, interações e atividades	Modelagem de agentes	Modelagem de interações e atividades	Modelo de agentes
			Modelo de interações
			Modelo de atividades
Projeto global	Construção do esboço do framework		Esboço do modelo arquitetural
	Seleção de padrão arquitetural		Modelo arquitetural
	Refinamento do framework		
Projeto detalhado	Detalhamento dos agentes		Modelo de atividades detalhado
	Seleção de padrão detalhado		Modelo de projeto detalhado
	Refinamento dos agentes		

O refinamento das técnicas GRAMO e DDEMAS compõe a MADEM. A Tabela 3 trás o resumo das fases de modelagem e tarefas da MADEM. O conhecimento dessa (MADEM) é representado na ONTOMADEM, uma ontologia que tem sido usada como ferramenta para captura e representação dos produtos do processo de Engenharia de Domínio. A ONTOMADEM foi desenvolvida no editor de Ontologia PROTÉGÉ.

Tabela 3 – Resumo das fases de modelagem e tarefas da metodologia de MADEM

Fases		Tarefas
Análise de Domínio		Modelagem de Conceito
		Modelagem de Meta
		Modelagem de Papel
		Modelagem de Variabilidade
		Modelagem de Interações de Papel
Projeto de Domínio	**Projeto Arquitetural**	Entendendo a área de resolução de problemas
		Mapeamento do modelo de papel em um primeiro desenho de um modelo de agente
		Mapeamento do modelo de interação de papel em um primeiro desenho de um modelo de interação de agente
		Reorganização da sociedade de agente por cooperação e mecanismos de coordenação
	Projeto Detalhado	Identificação de um padrão de projeto detalhado
		Definição do projeto de tipo de agente detalhado
		Modelagem do comportamento de agente
	Modelagem do conhecimento da sociedade de agente múltiplo	

3 ESTUDO DE CASO

Para um melhor entendimento da abrangência, dos limites e dos objetivos a serem alcançados, apresenta-se abaixo, na Figura 4, o Modelo de Objetivos. Para a sua elaboração, utilizaram-se a ferramenta Protégé e a técnica MADEM, apoiada na ontologia ONTOMADEM.

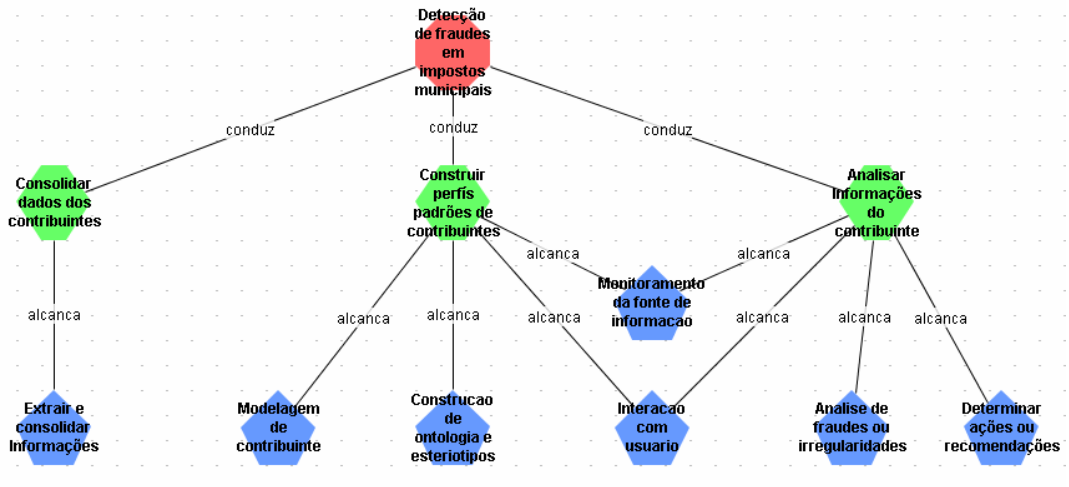


Figura 4. Modelo de Objetivos do Estudo de Caso.

Conforme se pode observar acima, temos:

- *um objetivo geral*: detecção de fraudes em impostos municipais;
- *três objetivos específicos*:

- Consolidar dados dos contribuintes, que consiste em extrair informações de diversas bases de dados produzidas pelo sistema de arrecadação, declaração mensal de serviços, emissão / controle de notas fiscais e sistemas estaduais e federais de tributação (informações de impostos retidos ou pagos pelas empresas). Este objetivo alcança a responsabilidade de extrair e consolidar informações;
- Construir perfis padrões de contribuintes, que consiste em classificar e construir padrões de contribuintes individuais e de grupos, possibilitando prever ações

futuras baseadas nas ações passadas. Este objetivo tem as responsabilidades de modelagem de contribuintes, construção de ontologias e estereótipos, interação com o usuário e monitoramento da fonte de informação;

- Analisar informações dos contribuintes, que consiste em analisar as informações armazenadas e, a partir de algoritmos e soluções pré-definidas, construir a base de conhecimento do sistema e determinar as ações e recomendações a serem tomadas. Este objetivo alcança as seguintes responsabilidades: monitoramento da fonte de informação; interação com o usuário; análise de fraudes ou irregularidades; e determinar ações ou recomendações.

Com base nos objetivos específicos expostos acima, descreveram-se três modelos utilizando agora a metodologia MAS-CommonKADS. Cada modelo trata de um objetivo específico.

3.1 Conceitualização

Conforme visto na descrição da MAS-CommonKADS, nesta fase se obtém a descrição geral do problema. Nela, deve-se descrever os atores e identificar os casos de uso.

3.1.1 Modelo 1

Este modelo trata do objetivo específico de consolidar dados dos contribuintes.

Na construção do modelo de extração de informações, foi observado um caso de uso, descrito abaixo.

3.1.1.1 Caso de uso Extrair Dados de Contribuintes

Sumário: Extrai e consolida informações de contribuintes provenientes de outros sistemas.

Atores: Sistema de Arrecadação; Sistema de Declaração Mensal de Serviços; Sistema de Informações de Autorização de Impressos Fiscais; Sistemas de Contribuintes Estaduais / Federais; Base de Dados Consolidada.

Pré-condições: os atores devem possuir informações suficientes sobre os contribuintes. O CNPJ é exigido.

Cenários principais:

- Extrair dados do Sistema de Arrecadação;
- Extrair dados do Sistema de Declaração Mensal de Serviços;
- Extrair dados do Sistema de Informações de Autorização de Impressos Fiscais;
- Extrair dados do Sistema de Contribuintes Estaduais / Federais;
- Gravar informações na Base de Dados Consolidada.

Cenários alternativos: caso não haja informações suficientes ou relevantes de um dos atores descritos acima, deve-se gravar exceções para tratamento posterior.

Requisitos especiais: a extração / consolidação das informações pode ser feita diariamente ou conforme período a ser estabelecido. Para a extração das informações do sistema de arrecadação, situado no *mainframe* UNISYS, base de dados DMSII, foi elaborado um programa em Cobol que gera um arquivo seqüencial que é transferido via FTP (*File Transfer Protocol*). Depois, usou-se o *Data Transformation Services* (DTS) da Microsoft para importar / exportar, extrair, transformar e consolidar dados de fontes heterogêneas. Esse processo será detalhado mais adiante, no item 3.2.1.1.

A Figura 5 apresenta do diagrama de caso de uso desse modelo:

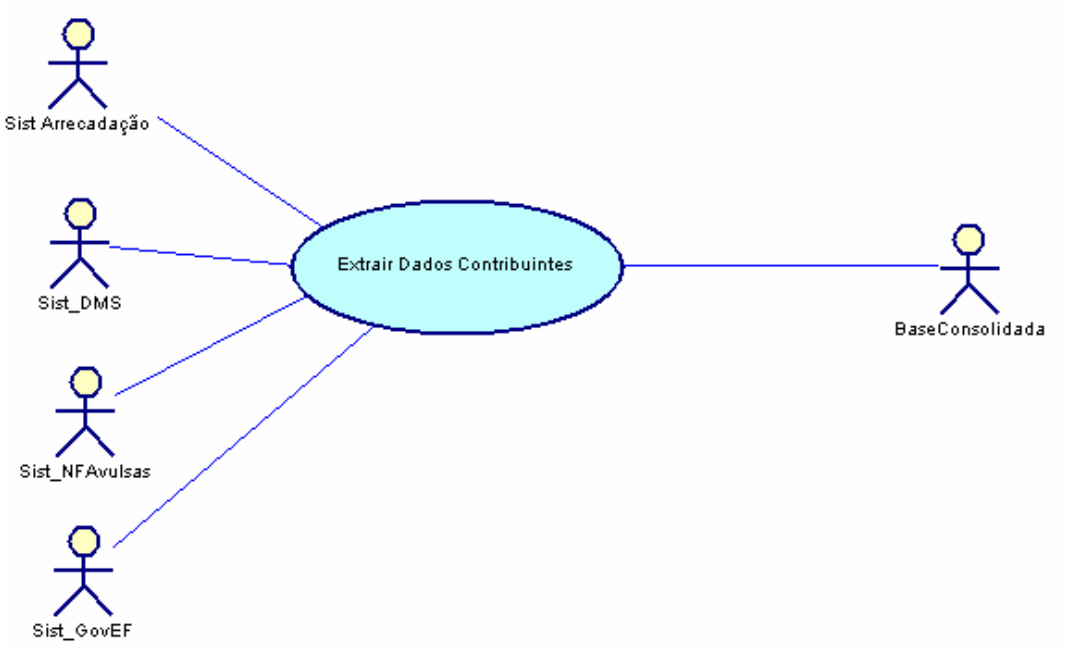


Figura 5 – Caso de Uso Extrair Dados de Contribuintes.

3.1.2 Modelo 2

Este modelo trata do objetivo específico de construir perfis padrões de contribuintes.

O Sistema Auditor (Analisador de Fraudes e Irregularidades) (ver Modelo 3, abaixo) necessita de informações referentes ao perfil do contribuinte, pois, a partir desse perfil, pode-se estabelecer parâmetros de comportamentos dos contribuintes de forma individual ou em grupos. Possibilita-se, assim, a verificação de mudanças bruscas de comportamento, o que indica possibilidade de fraude.

Os quatro casos de uso identificados na construção do perfil do contribuinte (vide Figura 4) foram:

- Interfaceador, também usado no Modelo 3;
- Modelador de Contribuintes;
- Construtor de Ontologia;

- Monitor das Fontes de Informação, também usado no Modelo 3, do qual é mais precisamente um módulo.

A Figura 6 apresenta o diagrama com os casos de uso identificados, cada caso de uso será detalhado a seguir.

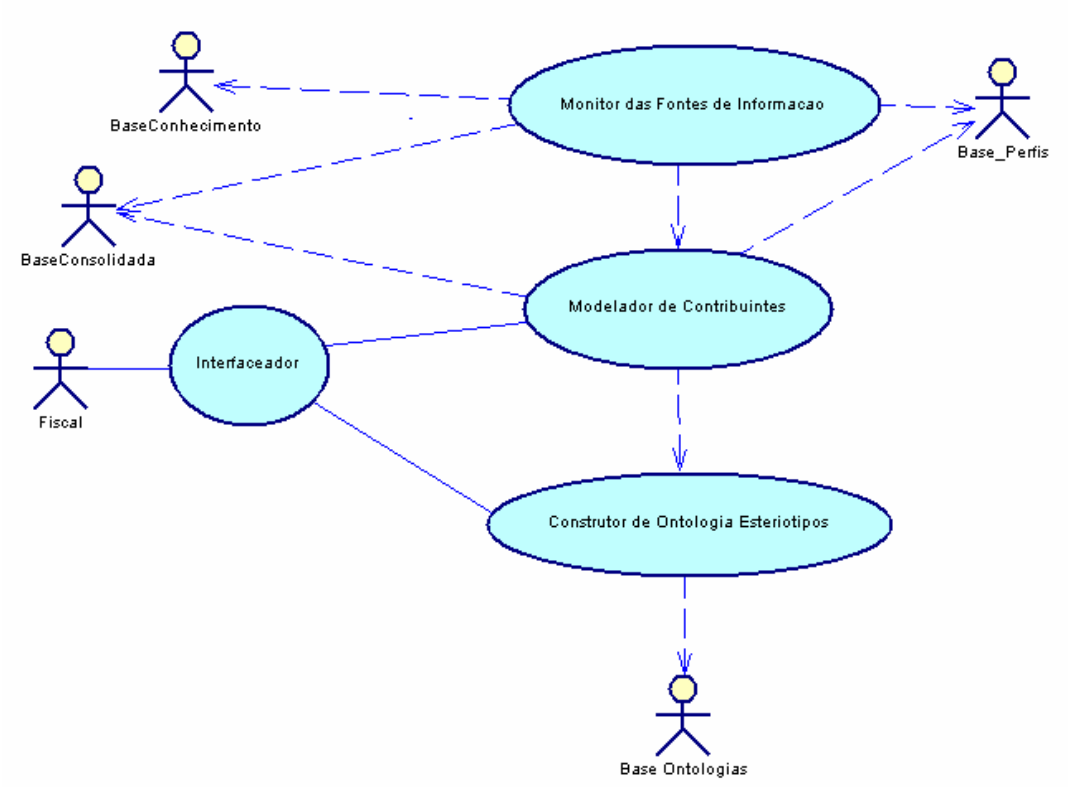


Figura 6 – Casos de uso do Modelo de Perfil de Contribuintes.

3.1.2.1 Caso de uso Interfaceador

Sumário: este caso de uso é iniciado pelo fiscal, que escolhe uma ação a ser realizada através da interface do usuário. Quando o módulo perfil é iniciado, as ações a serem realizadas são em boa parte automáticas, mas o fiscal pode intervir e determinar que ocorram certas ações, como: solicitar previsões de tributações futuras de um contribuinte específico; criar / alterar ontologias; solicitar informações de um contribuinte ou de um grupo.

Atores: fiscal.

Pré-condição: que a interface com o usuário esteja iniciada.

Cenários principais:

- o caso de uso Interfaceador é iniciado pelo ator fiscal, que pode solicitar alguma informação sobre dados futuros de algum contribuinte, por exemplo. Porém, o Modelador de Contribuinte também executa a grande maioria das ações de forma automática. O interfaceador também recebe do sistema e apresenta ao fiscal, o resultado de ações a serem tomadas quando da detecção de alguma irregularidade.
- As ações a serem realizadas pelo Modelador de Contribuinte, quer sejam manuais ou automáticas, acessam informações das bases consolidadas, de perfis e do Construtor de Ontologias (ontologias propriamente ditas).
- Quando a análise de informações feitas pelo Monitor das Fontes de Informação encontra alterações nos padrões de perfis, o Monitor invoca o Modelador, para que este realize as ações necessárias sobre o perfil do contribuinte alterado.

A interação entre os casos de uso pode ser visualizada nos Diagramas de Seqüência ou Colaboração, no item 3.2.4.

Cenários alternativos:

- o fiscal não é bem-sucedido ao enviar solicitação à interface;
- a interface é incapacidade de disparar as ações escolhidas.

3.1.2.2 Caso de uso Modelador de Contribuintes

Sumário: neste estudo, será realizada a análise individual do contribuinte (micro-modelo), enquanto o comportamento global (macro-modelo) será feito através do somatório dos comportamentos individuais (ver item 2.5). Os agrupamentos serão feitos por um supergrupo (todos os contribuintes) e através de agrupamentos baseados em outros critérios, como tipo de CNAE, grupo econômico, tamanho da empresa, faturamento, etc.

Essa classificação de perfil se baseia no estabelecimento de padrões por tipos de enquadramento de empresas através da utilização de *forecasting* (previsões).

Atores: Base Consolidada e Base de Perfis.

Pré-condições: que a Base Consolidada já possua informações suficientes e relevantes, para que seja possível fazer interações.

Cenários principais:

- criação de perfis individuais dos maiores contribuintes.
- Agrupamentos de perfis de acordo com grupos pré-estabelecidos.
- Algumas ações realizadas no caso de uso:
    - análise individual de cada contribuinte, comparando-se os dados históricos e verificando as tendências de comportamento ao longo do tempo;
    - determinação do comportamento global a partir de somatório de comportamentos individuais;
    - verifica novas entradas nos atores Base Consolidada e Base de Dados de Perfis analisando-as a partir de técnicas de previsão, construindo a base histórica dos contribuintes (perfis).

Cenários alternativos:

- insucesso na obtenção de informações na Base de Perfis.

Requisitos especiais: construção de um modelo estatístico de previsão (*forecasting*), vide item 2.5.

3.1.2.3 Caso de uso Construtor de Ontologias

A construção do mapa ontológico do sistema (visão macro da ontologia) é ilustrada na Figura 7.

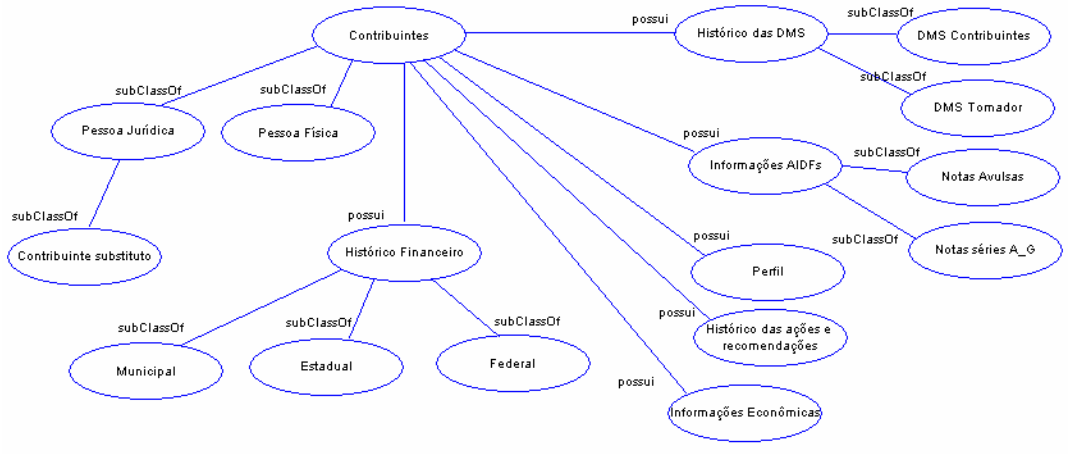


Figura 7. Mapa da Ontologia do Sistema Auditor – Macro Visão.

O mapa (nome aqui utilizado para representar o conjunto de classes envolvidas) traz uma visão geral da ontologia do sistema.

A classe "Contribuintes" possui relacionamento com as classes:

- Histórico das DMS **(Declaração Mensal de Serviços)**, que indica o detalhamento das declarações dos contribuintes e tomadores de serviços enviados mensalmente à prefeitura;
- Histórico financeiro, que possui as informações de todos os tributos devidos e pagos à prefeitura e aos governos estadual e federal (essas informações dependem de acordos a serem firmados com os respectivos órgãos competentes);
- Informações de AIDFs (Autorização de Impressos e Documentos Fiscais), que trazem a relação de todos os documentos fiscais autorizados e impressos;
- Informações de perfis, que possuem informações dos contribuintes ao longo do tempo e simulações de ações futuras;
- Histórico das ações e recomendações tomadas pelos agentes Analisador e Determinador ao longo da vida do sistema.

O detalhamento do modelo do domínio e da ontologia está no item 3.2.5 (Modelo de "Expertise") e no Anexo II. Nesses casos, são detalhados a ontologia e os atributos.

Inicialmente, não será feita nenhuma alteração na ontologia. Porém, caso informações adicionais sejam agregadas por razões de personalização e melhorias do domínio, as mesmas podem ser feitas.

3.1.2.4 Caso de uso Monitor das Fontes de Informações

Sumário: consiste em acompanhar e retornar informações sobre o estado das fontes de informações.

Atores: Base de Conhecimento, Base de Perfis e Base Consolidada.

Pré-condições: dispor de informações suficientes nas bases de dados, para que se possa avaliar os dados e ter mais precisão.

Cenários Principais:

- Acompanha o estado das fontes de informações visando à detecção de mudanças de dados relevantes, permitindo, assim, a realização de inferências, atualizando as informações ou informando os outros agentes sobre as alterações ocorridas;
- Envia informações para o Modelador e o Analisador sobre alterações nas fontes de informação;
- As fontes de informações monitoradas são:

  - Base de conhecimento;

  - Base de perfis;

  - Base consolidada.

Cenários Alternativos: não há.

3.1.3 Modelo 3

Este modelo possui o objetivo específico de analisar informações dos contribuintes.

Na observação dos cenários da busca e análise de informações dos contribuintes, obtiveram-se dois novos casos de uso, conforme ilustrado na Figura 8 e descrito abaixo.

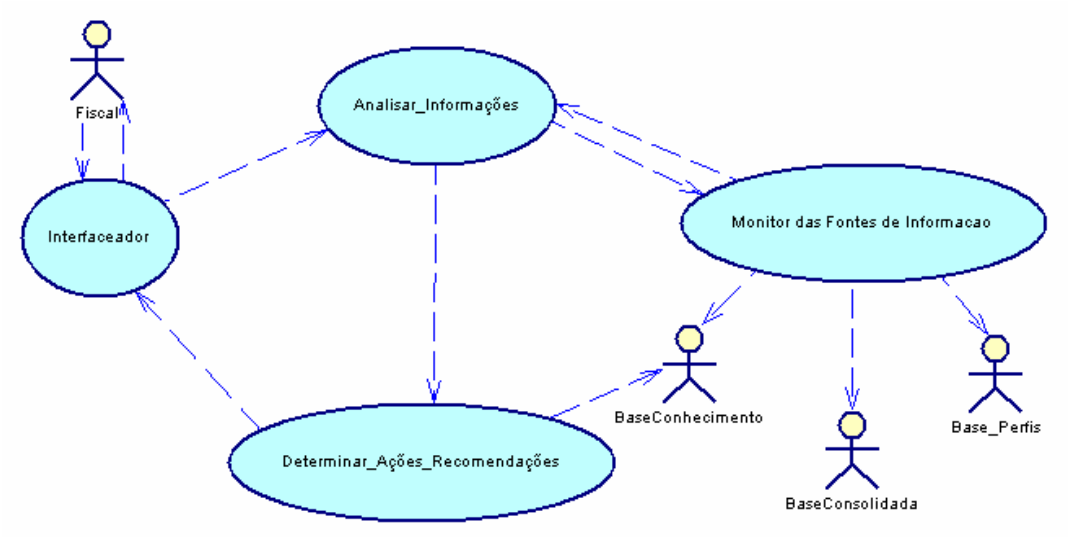


Figura 8 – Casos de Uso no Cenário de Busca e Análise de Informações.

3.1.3.1 Caso de uso Analisar Informações

Sumário: busca irregularidades e possíveis fraudes nas informações contidas no banco de dados consolidado e na análise de perfis-padrão.

O analisador recebe solicitações de ações do fiscal através do Interfaceador ou recebe um alerta do Monitor sobre irregularidades encontradas. Partindo de uma dessas entradas ou de ambas, analisa as informações, determinando as ações e recomendações a serem tomadas. As ações e recomendações são repassadas ao Interfaceador. O analisador retorna informações ao Monitor sobre a(s) decisão(ões) tomada(s), para que a base de conhecimento seja atualizada.

Atores: não há. No modelo proposto as bases ou os atores são acessados pelo Monitor das Fontes de Informações (Base Consolidada, Base de Perfis e Base de Conhecimento).

Pré-condições:

- A base de conhecimento e de perfis, acessadas e inferidas pelo Monitor, devem dispor de informações suficientes, de pelo menos seis meses, para que as inferências e análises possam ser feitas.

Cenários principais:

- Analisar as seguintes informações:
    - Documento fiscal não autorizado;
    - Serviços prestados com diferença de valor;
    - Serviços prestados não declarados;
    - Serviços prestados em duplicidade;
    - Serviços prestados com diferença na retenção;
    - ISS pago x ISS declarado;
    - Série histórica do ISS Mensal;
    - Série histórica do ISS Substituto;
    - Perfil de contribuinte diverge do seu padrão normal de comportamento;
    - Perfil de contribuinte difere do padrão determinado para aquele grupo de contribuintes.

Cenários alternativos:

Para os contribuintes que possuem informações insuficientes, a análise não deve ser feita.

Requisitos especiais:

Uso de técnicas avançadas de inteligência artificial.

3.1.3.2 Caso de uso Determinar Ações e Recomendações

Sumário: A partir das análises feitas pelo agente Auditor / Analisador, caso de uso Analisar Informações, e determinando-se a possibilidade de irregularidades, mensagens são passadas para o caso de uso "Determinar Ações e Recomendações", que irá proceder às ações e recomendações a serem realizadas.

Atores: Base de Conhecimento.

Pré-condições: Mensagens devem ser passadas pelo analisador.

Cenários principais:

- determina Ações;
- avisa o Fiscal sobre ações corretivas, preventivas ou punitivas a serem aplicadas ao contribuinte;
- as ações aplicadas devem ir para a base de dados de conhecimento.

Cenários alternativos:

- As recomendações inconsistentes devem ser apontadas pelo Fiscal para que o sistema aprenda e passe a ter mais consistência.

Resumindo os três modelos apresentados, apresenta-se, na Tabela 4, os atores e casos de uso.

Tabela 4. Atores e casos de uso.

Ator	Descrição	Casos de Uso
Sistema de Arrecadação	Contém informações cadastrais dos contribuintes, além do histórico de impostos pagos, lançados ou retidos	Extrair Dados de Contribuintes
Sistema de Declaração Mensal de Serviços	Contém informações fornecidas pelos contribuintes e tomadores de serviços sobre as notas fiscais emitidas e recebidas	Extrair Dados de Contribuintes
Sistema de Informações de Autorização de Impressos Fiscais	Contém informações referentes à impressão e emissão de notas fiscais e notas avulsas, com a numeração e a validade	Extrair Dados de Contribuintes
Sistemas de contribuintes estaduais / federais	Contêm informações sobre as declarações dos contribuintes nos âmbitos estadual e federal	Extrair Dados Contribuintes

Base Consolidada	È o produto do caso de uso "Extrair Dados Contribuintes". Possui informações consolidadas dos diversos atores descritos acima	Extrair Dados Contribuintes Monitor das Fontes de Informações Modelador de Contribuintes
Fiscal (Solicita informações ao sistema; seleciona ações)	Entidade que deve intervir buscando informações	Interfaceador
Base de Perfis	Contém as informações coletadas e inferidas sobre o perfil do contribuinte	Monitor das Fontes de Informações Modelador de Contribuinte
Base de Conhecimento	Produto de informações aprendidas pelos agentes. Depende também de informações da Base Consolidada e da Base de Perfis	Monitor das Fontes de Informações Determinador de ações e recomendações
Base das Ontologias	Contém as informações das ontologias e dos estereótipos	Construtor de Ontologias

3.2 Análise

Nesta parte, aplicam-se os seis modelos do MAS-CommonKADS para se obter uma especificação de requisitos do sistema multiagentes[2].

3.2.1 Modelo de Agentes

É a parte central da análise.

Segundo a metodologia MAS-CommonKADS, os agentes podem ser identificados seguindo as seguintes estratégias ou uma combinação delas [IGLESIAS et al., 1998]:

- Análise dos atores dos casos de uso definidos na fase de conceitualização. Os atores delimitam os agentes externos do sistema;
- Análise do problema. A análise sintática da descrição do problema pode ajudar a identificar alguns agentes. Os agentes candidatos são os sujeitos das sentenças, os objetos ativos. As ações executadas por esses sujeitos podem ser desenvolvidos pelos agentes com metas (com iniciativa) ou serviços (sobre demanda);

[2] Para reforçar e enriquecer a análise, apresentar-se-á, quando conveniente, alguns modelos implementados na técnica MADEM.

- Uso de heurísticas. Os agentes podem ser identificados, caso determinem a existência de alguma distância conceitual [BOND and GASSER, 1988], distribuição de conhecimento, distribuição geográfica, distribuição lógica ou distribuição organizacional;
- Uma tarefa inicial e modelos de *expertise* podem ajudar na identificação de funções necessárias e requisitos de conhecimentos, resultando na definição preliminar dos agentes. Os objetivos das tarefas podem ser atribuídos aos agentes;
- Aplicação da técnica de Casos de Uso Internos. Esta técnica é baseada em RDD (*Responsibility Driving Design*) [WIRFS-BROCK, 1990] e suas cartas de CRC (*Class Responsibility Collaboration*);
- Aplicação das cartas de CRC melhoradas. Uma CRC é preenchida para cada agente, descrevendo suas classes. Cada CRC é dividida em cinco colunas: objetivos pretendidos, planos para atingir esses objetivos, conhecimentos necessários para realizar o que se planejou, colaboradores desses planejamentos e serviços usados na colaboração.

Seguindo as duas primeiras estratégias citadas acima, as identificações dos agentes foram feitas com a descrição do modelo de objetivos, conforme ilustrado na Figura 4, e com as especificações dos casos de uso no item 3. Porém, nem todos serão efetivamente transformados em agentes.

3.2.1.1 Casos de Usos que Não Serão Transformados em Agentes

I Caso de uso Extrair dados dos Contribuintes. Este caso de uso poderia ter sido transformado em agente. Porém, por não ser o foco principal de nosso estudo, será usada

uma ferramenta[3] automática de extração, transformação e consolidação dos dados de várias fontes.

A Figura 9 traz o modelo de papel do Extrator de Informações, modelado no Protégé.

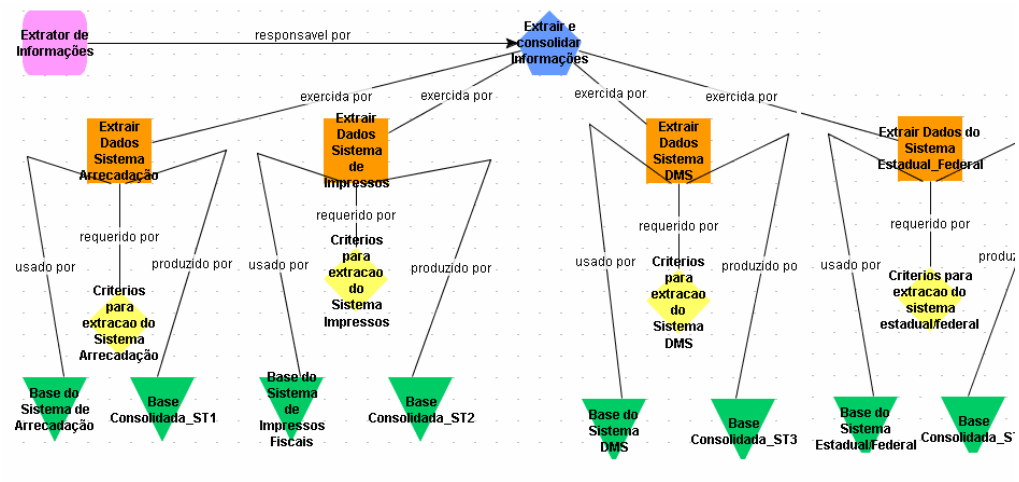


Figura 9 – Modelo de Papel do Extrator de Informações.

Conforme se pode observar acima, existem quatro tarefas de extração de informações, cada uma com seus critérios, dados de entrada e de saída.

A tarefa Extrair Dados do Sistema de Arrecadação é realizada numa primeira etapa por um programa feito em COBOL, que lê as estruturas existentes num banco de dados hierárquico, DMSII da UNISYS®, gerando um arquivo seqüencial. As tabelas lidas são referentes ao cadastro do contribuinte (dados da empresa e dos sócios), informações das estruturas econômicas (porte da empresa, CNAE, etc.) e financeiras (pagamentos realizados, impostos pagos, dívidas, etc.). Em seguida, esse arquivo seqüencial é transferido para um servidor via FTP e importado para a base consolidada via DTS (ver Figura 10).

As tarefas de extração de dados dos sistemas de impressos (dados armazenados num servidor com MS SQL 2000®) e de DMS (armazenado num servidor com base de dados PostgreeSQL) é feita diretamente usando o DTS, sendo esse processo de extração armazenado

[3] A ferramenta utilizada será o *Data Transformation Services*, da Microsoft.

numa base temporária MS SQL 2000. A base de dados temporária é então processada e gravada via *storage procedure* na base consolidada.

As informações enviadas à prefeitura pelos governos estadual e federal são transmitidas via arquivo seqüencial e estão de acordo com um *layout* previamente definido. Tais informações são também incorporadas à base consolidada via DTS.

A Base Consolidada possui várias instâncias, sendo atualizada em cada etapa do processo de extração.

Os critérios para extrações obedecem a regras bem definidas e serão aplicadas no processo.

A Figura 10 apresenta de forma esquemática como funciona esse processo de extração nas diversas fases.

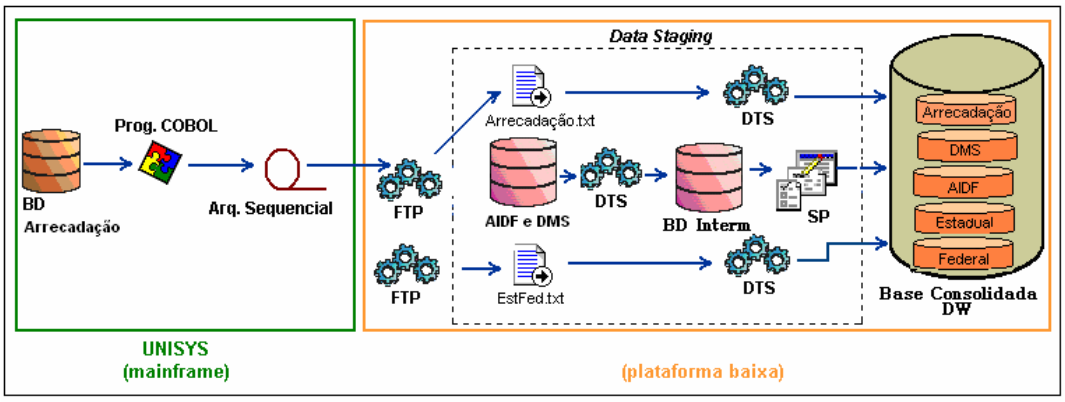


Figura 10 – Processo de Extração de Dados.

II Construir Mapa de Ontologia e Estereótipos

Este caso de uso sofrerá intervenções pontuais e manuais.

Conforme se pode observar na Figura 11, abaixo, este caso de uso consiste em diversas tarefas, que são:

- Construir e alterar os mapas das ontologias: consiste em construir e alterar as ontologias com base nas informações do domínio. Após aplicar os critérios para construir / alterar as ontologias, ter-se-á a representação ontológica em

forma de árvore (vide item 3.2.5), na forma de representação semântica e em lógica de primeira ordem. Neste caso, será usado o JESS (vide Capítulo 5);

- Consultar mapas das ontologias: após a construção das ontologias, deve-se permitir que elas sejam consultadas. Essas consultas são feitas nas representações da ontologia em lógica de primeira ordem;
- Construir e alterar listas de estereótipos: esta tarefa define os estereótipos de armazenamento, determinando e nomeando grupos de padrões e verificando afinidades;
- Consultar listas de estereótipos: consiste em permitir a consulta dos estereótipos pelos agentes.

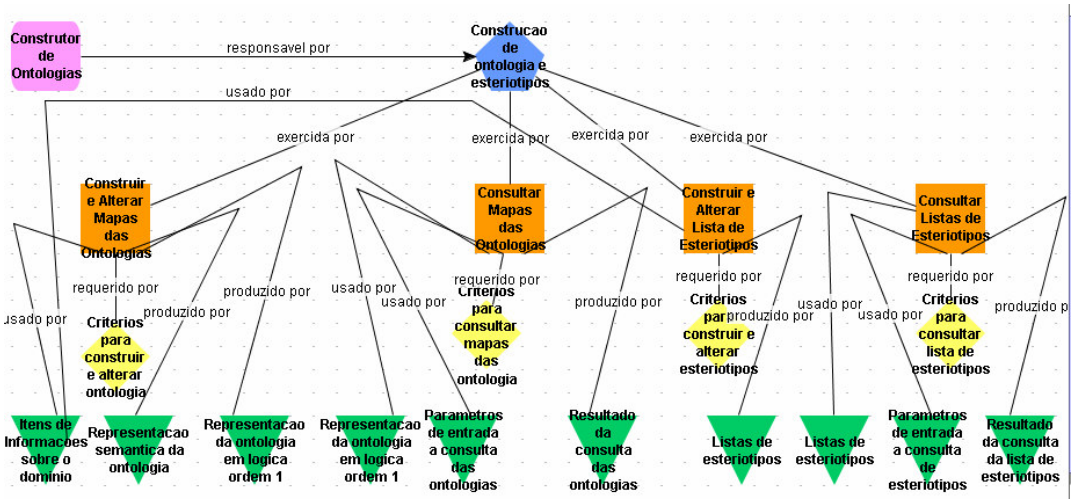


Figura 11 – Modelo de Papéis do Construtor de Ontologias.

3.2.1.2 Casos de Uso Transformados em Agentes

Os casos de uso a serem transformados em agentes são:

III Interfaceador (Agente Interfaceador);

IV Modelador de Contribuinte (Agente Modelador);

V Monitor das Fontes de Informações (Agente Monitor).

VI Analisar Informações (Agente Analisador);

VII Determinar Ações e Recomendações (Agente Determinador).

Na Figura 12 tem-se o Modelo de Agentes utilizando a técnica MADEM.

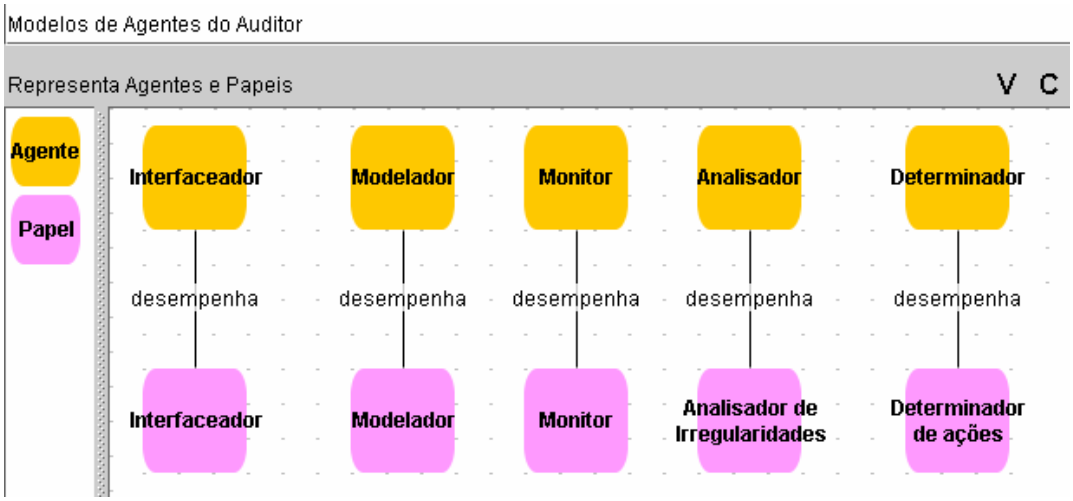


Figura 12 – Modelo de Agentes – MADEM.

Para melhor abstrair o alcance, a responsabilidade, as tarefas ou atividades e os dados de entrada e saída de cada atividade dos agentes, construiu-se o Modelo de Papéis utilizando a ferramenta Protégé e a técnica MADEM para cada um dos agentes.

A metodologia MAS-CommonKADS realiza a descrição textual de cada agente. Seguem abaixo as tabelas descritivas sobre as informações de cada agente.

3.2.1.2.1 Agente 1: Interfaceador

A Figura 13 ilustra o modelo de papéis do Interfaceador.

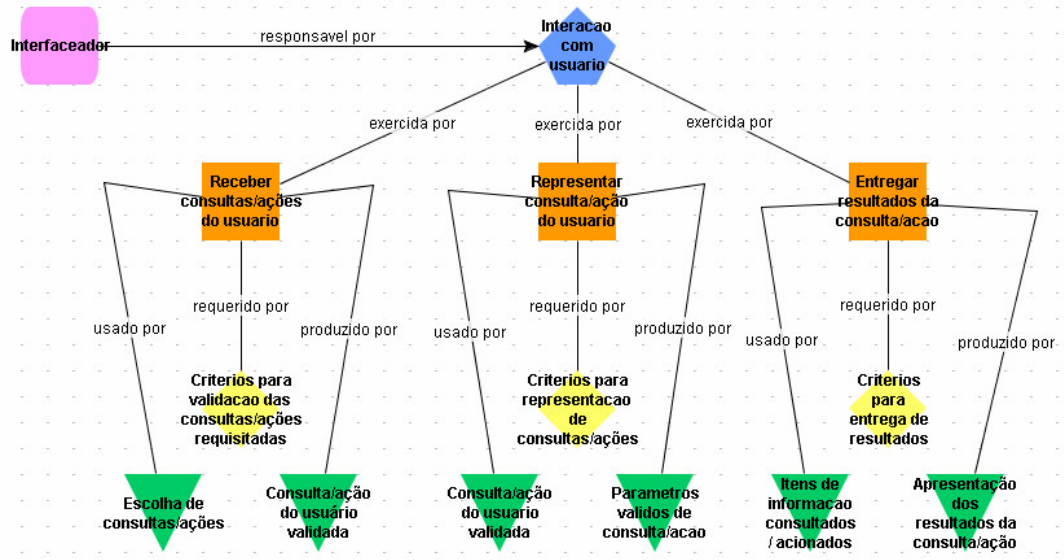


Figura 13 – Modelo de Papéis do Interfaceador.

- Na tarefa "Receber consultas / ações do usuário", o usuário / fiscal acessa o menu do sistema e determina a consulta ou ação que o sistema deve efetuar. Desde que os critérios de validação da consulta / ação sejam atendidos, a consulta é representada internamente pela tarefa "Representar consulta / ação do usuário", que passa as informações em forma de passagem de parâmetro para o agente analisador. O agente analisador processa a informação e retorna o resultado para o interfaceador.

A Tabela 5 resume as características desse agente.

Tabela 5 – Descrição do Agente Interfaceador.

Nome:	Interfaceador
Tipo:	Agente de *software*
Regra:	Fornece a interface com o usuário
Localização:	Na sociedade de agentes de perfis e análise
Descrição:	Fornece interface ao ator fiscal
Objetivo:	Intermediação entre os usuários e o sistema, tanto para a entrada de dados, na forma de necessidades de informação especificadas por aqueles, quanto para a saída de resultados, na forma de satisfações de informação fornecidas por este
Exceções:	Não se aplicam
Parâmetros de entrada:	Seleção de consulta ou tomada de ações feitas pelo fiscal, através de uma interface
Parâmetros de saída:	Resultado da consulta / ação solicitada
Serviços:	Solicita consultas e ações ao sistema e entrega os resultados

Expertise:	Critérios para validação das consultas; critérios para a representação de consultas do usuário; critérios para a entrega de resultados da consulta; critérios para a recepção de ações a realizar; critérios para entrega de resultados das ações
Comunicação:	Passagem de mensagem; entrada: fiscal ou usuário que interage com a interface e o retorno da consulta pelo agente determinador; saída: para o agente analisador
Coordenação:	Vide item 3.2.4

3.2.1.2.2 Agente 2 - Agente Modelador de Contribuinte

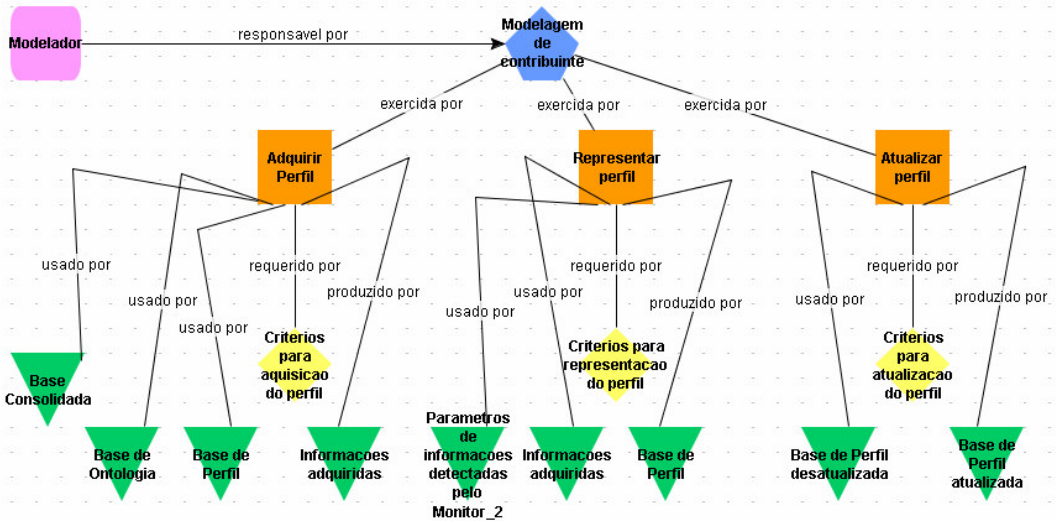


Figura 14 – Modelo de Papeis do Modelador.

Observando a Figura 14, vemos que o agente modelador consulta as bases consolidada, de ontologia e de perfil e, a partir de critérios (descritos no item 2.5), o agente representa o perfil do contribuinte, podendo prever ações futuras a partir de ações passadas. A tarefa atualizar perfil faz o mesmo processo da aquisição de perfil só que o contribuinte já tem uma base histórica de perfil construída.

A Tabela 6 resume as características deste agente.

Tabela 6 – Descrição do Agente Modelador.

Nome:	Modelador de Contribuinte
Tipo:	Agente de *software*
Regra:	Adquire perfis de usuários e grupos
Localização:	Na sociedade de agentes de perfis
Descrição:	Consiste em criar perfis de contribuintes; prevê ações futuras com base em ações passadas

Objetivo:	Verificar enquadramentos e distorções dos perfis de comportamento e padrões, através de algoritmos de *forecasting* e análise estatísticas
Exceções:	Para a análise dos usuários e grupos, trabalha apenas com *forecasting* (previsões), conforme descrito no item 2.5
Parâmetros de entrada:	Base de Ontologia; Base de Perfis; Base Consolidada
Parâmetros de saída:	Informações adquiridas; Base de Perfis atualizada
Serviços:	Modela o perfil do contribuinte; infere sobre comportamentos futuros do usuário e do grupo
Expertise:	Critérios para a aquisição do perfil; critérios para a representação e atualização dos perfis, usando equações para a definição de um perfil para cada contribuinte
Comunicação:	Via passagem de mensagem; recebe informações do Monitor
Coordenação:	Vide item 3.2.4

3.2.1.2.3 Agente 3: Monitor das Fontes de Informação

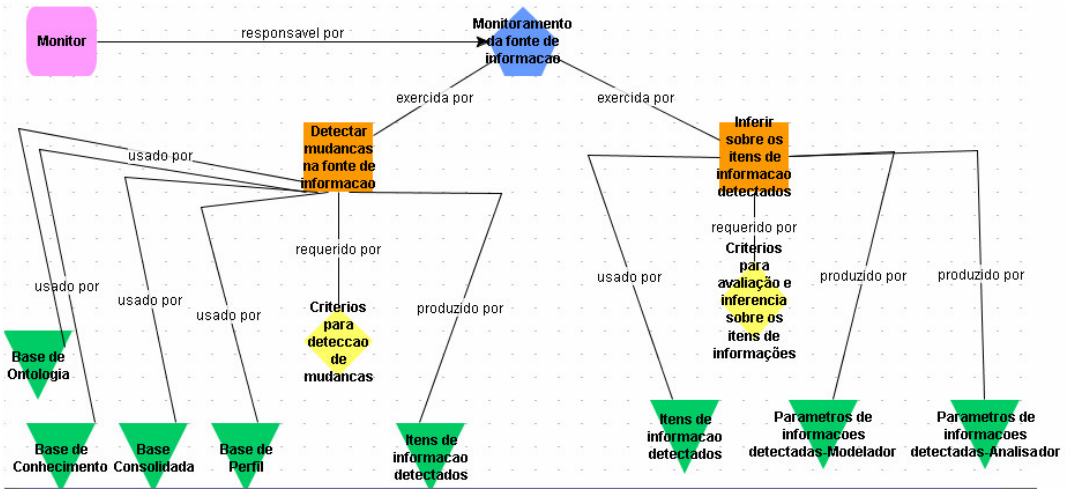


Figura 15 – Modelo de Papéis do Monitor.

Conforme se pode observar na Figura 15, a primeira tarefa deste agente tem o objetivo de detectar mudanças nas diversas fontes de informação, a partir de critérios pré-definidos. Quando existem itens detectados, tarefa dois, o agente executa inferências e envia mensagens para o agente modelador (apenas quando ocorrem alterações no perfil) e para o agente analisador, que verifica se essas alterações apontam para alguma tendência de fraude.

A Tabela 7 resume as características deste agente.

Tabela 7 – Descrição do Agente Monitor.

Nome:	Monitor das Fontes de Informação
Tipo:	Agente de *software*
Regra:	Monitora as fontes de informação
Localização:	Na sociedade de agentes de perfis e análise
Descrição:	Analisa alterações nas bases de dados e aciona os agentes, informando sobre as alterações sofridas
Objetivo:	Acompanhamento do estado das fontes de informações para detectar mudanças nesses itens, permitindo a realização de inferências As fontes de informações monitoradas são: - Base de Conhecimento; - Base Consolidada; - Base de Perfis; - Base de Ontologias
Exceções:	Não se aplicam
Parâmetros de entrada:	Base de Conhecimento; Base de Ontologias; Base de Perfis; e Base Consolidada
Parâmetros de saída:	Parâmetros de informações detectadas enviadas para o agente modelador, analisador ou ambos
Serviços:	Detecção automática dos itens de informação e interação com outros agentes, informando as detecções
Expertise:	Critérios para a detecção de mudanças; critérios para a avaliação e inferência sobre os itens de informações alterados
Comunicação:	Passagem de parâmetro; envia informações para os agentes modelador e/ou analisador
Coordenação:	Vide item 3.2.4

3.2.1.2.4 Analisador de Informações (Auditor)

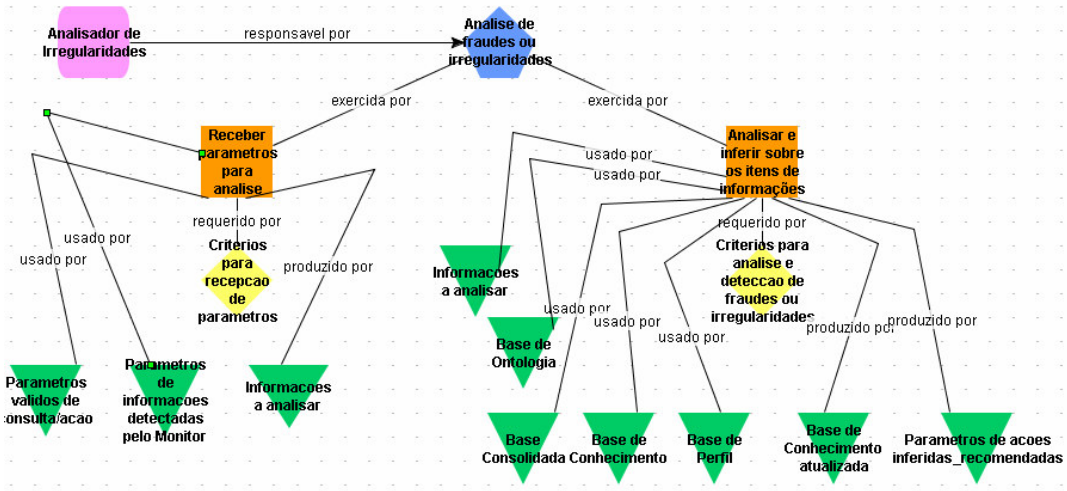


Figura 16 – Modelo de Papéis do Analisador (Auditor).

Conforme se pode observar na Figura 16, a tarefa "Receber parâmetros para análise" recebe parâmetros de entrada provenientes dos agentes interfaceador (Parâmetros válidos de consulta / ação) e monitor (Parâmetros de informações detectadas pelo monitor). A partir das informações passadas, analisa e infere sobre os itens de informações (tarefa 2), atualizando a base de conhecimento e enviando parâmetros para o agente determinador, para que as ações recomendadas sejam realizadas.

A Tabela 8 resume as características deste agente.

Tabela 8 – Descrição do Agente Analisador.

Nome:	Analisador
Tipo:	Agente de *software*
Regra:	Analisa as informações solicitadas ou repassadas por outros agentes, determinado ou não a existência de fraudes
Localização:	Na sociedade de busca e análise de informações
Descrição:	A partir de solicitações feitas pelo interfaceador (usuário) ou de alterações detectadas pelo monitor, o agente analisador verifica os parâmetros passados, consulta as bases de informações, quando necessário, e faz inferências, atualizando as informações na base de conhecimento e determinando ações a serem realizadas
Objetivo:	Analisar informações e determinar as ações a serem realizadas
Exceções:	Quando as informações detectadas como fraudes são errôneas; o fiscal deve intervir, informando ao agente como preceder nesses casos
Parâmetros de entrada:	Parâmetros passados pelo interfaceador, parâmetros passados pelo monitor, pela base consolidada, pela base de conhecimento, pela base de perfis e pela base de ontologia
Parâmetros de saída:	Solicitações de ações a serem realizadas e atualizações na base de conhecimento
Serviços:	Análise e detecção de fraudes ou irregularidades
Expertise:	Critérios para analisar e detectar fraudes ou irregularidades. Essas análises serão feitas a partir de algoritmos e soluções pré-definidas, retro-alimentando a base de conhecimento do sistema e determinando as ações e recomendações e a serem realizadas
Comunicação:	Passagem de parâmetro; interfaceador e/ou monitor (recebimento / entrada), determinador (saída / envio).
Coordenação:	Vide item 3.2.4

3.2.1.2.5 Agente 5: Determinador de Ações e Recomendações

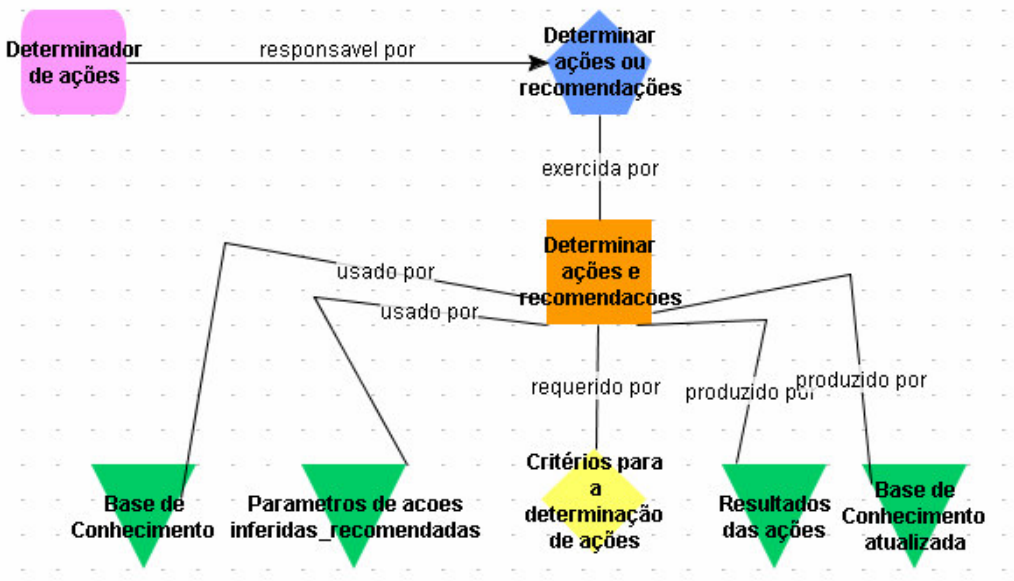


Figura 17 – Modelo de Papeis do Determinador.

A Figura 17 ilustra que este agente possui apenas uma tarefa, a qual é acionada após o recebimento do(s) parâmetro(s) do analisador sobre fraudes e irregularidades encontradas. Com isso, a ação é executada, a partir de critérios pré-estabelecidos, e o(s) resultado(s) da ação é(são) apresentado(s) ao fiscal através do interfaceador. A base de conhecimento é atualizada sempre que uma nova regra é aplicada, possibilitando assim o aprendizado.

A Tabela 9 resume as características deste agente.

Tabela 9 – Descrição do Agente Determinador.

Nome:	Determinador
Tipo:	Agente de *software*
Regra:	Determina as ações e recomendações a serem realizadas
Localização:	Na sociedade de agentes de busca e análise de informações
Descrição:	Determina ações a partir de informações repassadas pelo analisador
Objetivo:	A partir de mensagens passadas pelo analisador, serão determinadas as ações e recomendações a serem realizadas
Exceções:	Quando as informações detectadas como fraudes são errôneas, o fiscal deve intervir, informando para o agente como preceder nesses casos
Parâmetros de entrada:	Informações sobre as ações a serem realizadas, passadas pelo analisador e a base de conhecimento
Parâmetros de saída:	Resultados das ações e recomendações e atualizações da base de conhecimento
Serviços:	Infere sobre ações; capacidade de aprendizagem

Expertise:	Critérios para a determinação de ações (informações passadas pelo analisador, juntamente com o conjunto de ações e recomendações já realizadas e aprendidas, armazenadas na base de conhecimento)
Comunicação:	Passagem de parâmetros com os agentes analisador (recebimento / entrada) e o interfaceador (envio / saída)
Coordenação:	V. item 3.2.4

3.2.2 Modelo de Tarefas

Descreve as tarefas que os agentes podem executar.

O MAS-CommonKADS não inclui estrutura gráfica própria. Por isso, usa-se o diagrama de atividades do UML para representar o fluxo de atividade e o modelo textual para descrever a tarefa (nome, descrição breve, etc.).

As tarefas apresentadas abaixo, em cada agente, foram retiradas dos modelos de papéis presentes no item 3.2.1.2.

3.2.2.1 Tarefas do Agente Interfaceador

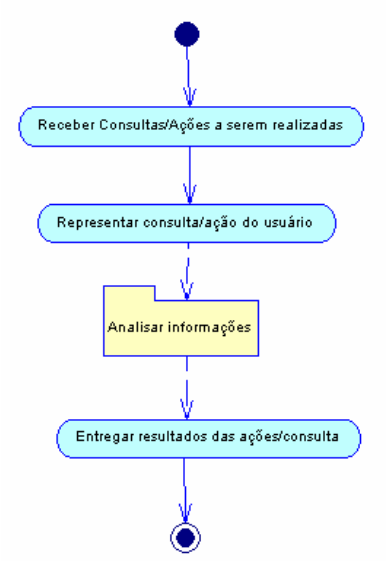


Figura 18 – Diagrama de Atividades do Agente Interfaceador

Modelo Textual:

Tarefa 1: Receber consultas / ações do usuário

- Objetivo: apresentar a interface para o usuário e receber (da tela preenchida pelo usuário) os parâmetros de ações administrativas ou solicitações de intervenções / tarefas a serem executadas pelo sistema;
- Descrição: é apresentada uma interface que irá conter ações administrativas ou solicitações de intervenções / tarefas a serem executadas pelo sistema;
- Ingredientes: informações de consultas pré-definidas e consultas construídas pelo usuário;
- Restrições: não tem;
- Exceções: tipo de consulta que o usuário necessita não está disponível.

Tarefa 2: Representar consultas / ações do usuário

- Objetivo: representar a consulta do usuário em forma de parâmetros aceitos pelo agente que irá receber a mensagem;
- Descrição: após receber as opções requisitadas pelo usuário, o agente interfaceador as transforma em parâmetros aceitos pelo agente analisador;
- Ingredientes: modela os parâmetros para o analisador;
- Restrições: não tem;
- Exceções: não tem.

Tarefa 3: Entregar resultados das ações / consultas

- Objetivo: receber a resposta da consulta / ação;
- Descrição: recebe o resultado das consultas ou ações requisitadas, formatando-as e apresentando-as ao usuário. Recebe ou indica o resultado da intervenção ou ação realizada sobre o contribuinte;
- Ingredientes: lista as ações / consultas realizadas;

- Restrições: não tem;
- Exceções: a consulta realizada pelo usuário não teve resultado disponível.

3.2.2.2 Tarefas do Agente Modelador

As tarefas do agente modelador encontram-se representadas na Figura 19.

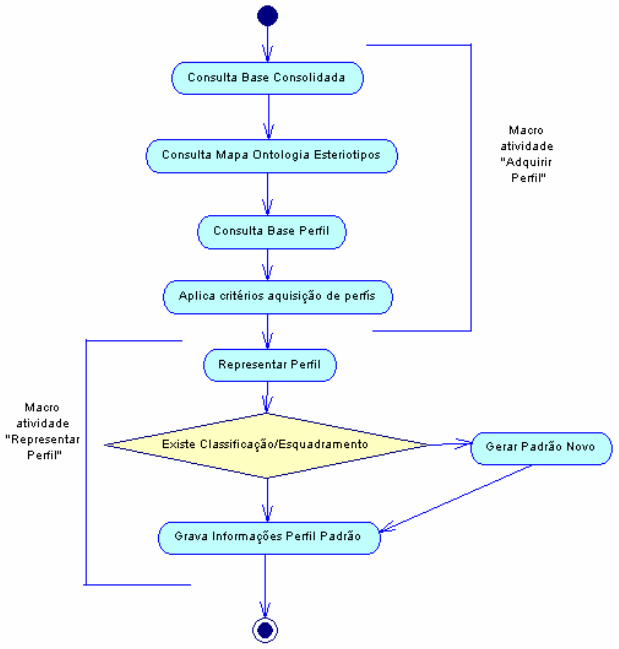


Figura 19 – Diagrama de Atividades do Agente Modelador.

Tarefa 4: Adquirir Perfil

- Objetivo: obter informações para a construção de perfis de contribuintes;
- Descrição: esta macro-atividade é composta de quatro outras, sendo três delas responsáveis por consultar as bases de informações sobre o contribuinte e ontologias: consulta base consolidada; consulta base de ontologia e estereótipo; consulta base de perfil; e uma tarefa referente à aplicação dos critérios ou inferências para a construção dos perfis;

- Ingredientes: constrói perfis de comportamentos individuais (micro-modelos) e de grupos (macro-modelos, soma dos comportamentos individuais) com base em ações ou informações passadas;
- Restrições: necessita de uma base consolidada de pelo menos seis meses de informações sobre o(s) contribuinte(s) para uma melhor previsão;
- Exceções: não tem.

Tarefa 5: Representar Perfil

- Objetivo: representar o perfil de acordo com as classificações ou enquadramentos padrões pré-existentes (apenas no caso de grupo de usuários);
- Descrição: após a aplicação dos critérios na tarefa anterior, o agente vai procurar o(s) enquadramento(s) do(s) contribuinte(s) de acordo com os grupos pré-definidos (por classificação CNAE, setor econômico, faixa de faturamento, etc.), gravando as informações no perfil-padrão de grupo enquadrado;
- Ingredientes: uso de inferências para a geração dos macro-modelos;
- Restrições: necessita de uma base consolidada de pelo menos seis meses de informações sobre o(s) contribuinte(s) para uma melhor previsão;
- Exceções: não tem.

Tarefa 6: Atualizar Perfil

- Objetivo: atualização sistemática da base de perfis;
- Descrição: esta é uma atividade automática do agente. O agente modelador, sempre que o agente monitor informa sobre alterações nas

fontes de informação, redefine o perfil individual e de grupos e atualiza a base de perfis. Essa atividade faz exatamente o que foi descrito nas tarefas 4 e 5;

- Ingredientes: desperta e executa inferências sempre que o monitor informa sobre novas alterações nas fontes de informação;
- Restrições: não tem;
- Exceções: não tem.

3.2.2.3 Tarefas do Agente Monitor das Fontes de Informações

As tarefas do agente Monitor encontram-se representadas na Figura 20.

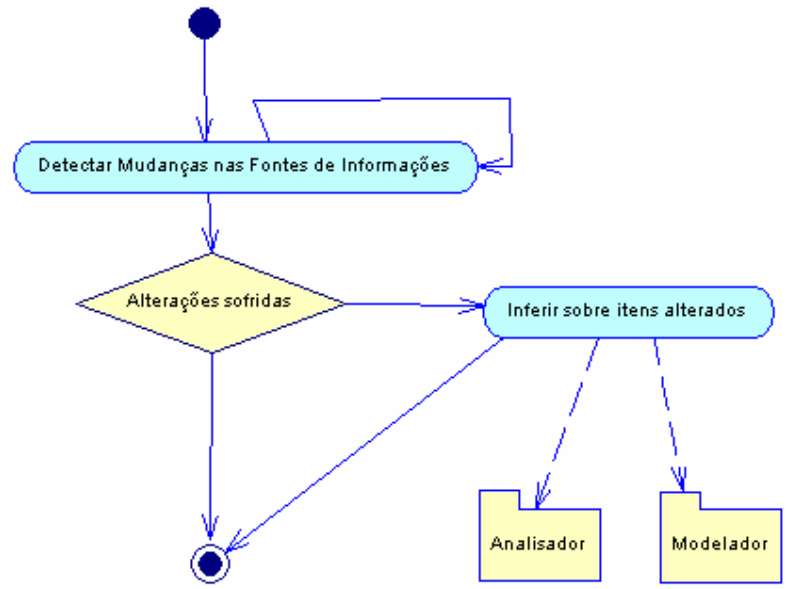


Figura 20 – Diagrama de Atividades do Agente Monitor das Fontes de Informações.

Tarefa 7: Detectar Mudanças nas Fontes de Informações

- Objetivo: detectar mudanças nas fontes de informações (base de conhecimento, base consolidada, base de perfil e ontologias), que podem ocorrer pela inclusão, alteração, exclusão ou novo enquadramento de itens de informação;

- Descrição: analisa as bases de informações, procurando por alterações sofridas. Os itens de informações alterados são marcados, para que possam ser inferidos pela tarefa 8;
- Ingredientes: monitora alterações nas fontes de informações;
- Restrições: não tem;
- Exceções: não tem.

Tarefa 8: Inferir sobre os Itens de Informações Alterados

- Objetivo: com as alterações verificadas na tarefa 7, encaminham-se mensagens para os agentes analisador e modelador;
- Descrição: com base nas informações alteradas, encaminham-se mensagens para que os agentes modelador e analisador atualizem as bases de perfil e de conhecimento, respectivamente;
- Ingredientes: possui a lista de alterações sofridas;
- Restrições: não tem;
- Exceções: não tem.

3.2.2.4 Tarefas do Agente Analisador de Informações

Este agente é o principal produto do referido sistema, pois, a partir de parâmetros recebidos pelos agentes interfaceador e monitor (no caso de parâmetros recebidos pelo interfaceador, o analisador consulta também as bases de informações), aplicam-se os algoritmos ou heurísticas necessárias para criar / alterar a base de conhecimento e determinar ações e recomendações a serem realizadas pelo fiscal / administrador.

Segue na Figura 21, o diagrama de atividades do analisador:

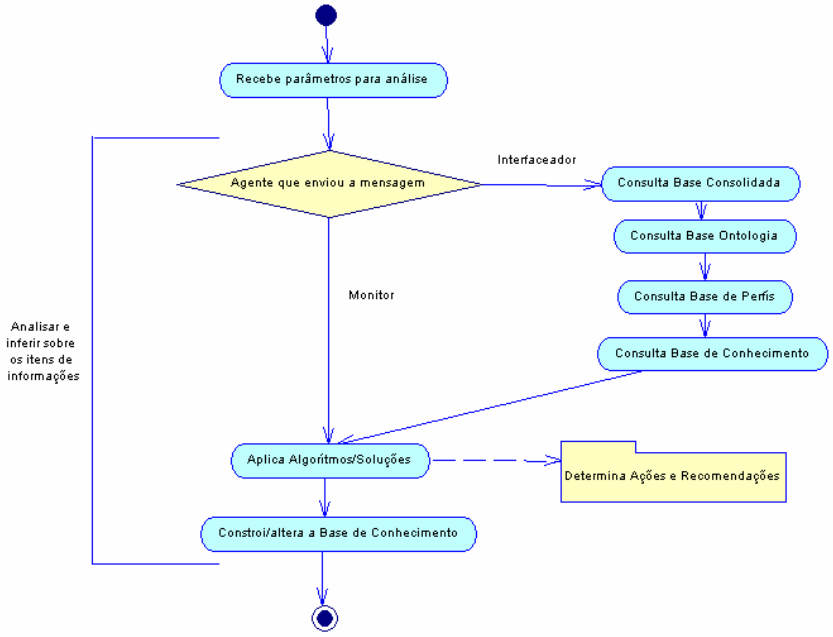


Figura 21 – Diagrama de Atividades do Agente Analisador de Informações.

Tarefa 9: Receber parâmetros para análise

- Objetivo: receber os parâmetros dos agentes;
- Descrição: recebe parâmetros dos agentes interfaceador e/ou monitor para análise;
- Ingredientes: identifica o agente chamador;
- Restrições: não tem;
- Exceções: não tem.

Tarefa 10: Analisar e inferir sobre os itens de informações

- Objetivo: analisar e inferir sobre os itens de informações, visando encontrar indícios de fraudes e irregularidades dos contribuintes de ISS;
- Descrição: quando os parâmetros recebidos vêm do agente interfaceador ou provêm de informações fornecidas pelo usuário, o analisador consulta informações das bases de dados (sub-tarefa 10-1) e aplica algoritmos e soluções (sub-tarefa 10-2). Quando os parâmetros são provenientes do monitor, o analisador faz apenas a sub-tarefa 10.2,

pois os parâmetros passados contêm todas as informações necessárias para o analisador. A sub-tarefa 10.3 e a mensagem para o agente determinador são realizadas independentemente do originador da mensagem inicial;

- Ingredientes: verificação de procedimentos de análise (v. item 4 – Implementação).
- Restrições: não tem;
- Exceções: a consulta ou requisição do interfaceador (feita pelo usuário) pode não ter informações suficientes para a análise.

A tarefa 10 é composta de algumas sub-tarefas, que são:

Sub-tarefa 10-1: Consultar as bases de informações

- Objetivo: consulta as informações contidas nas Bases de Dados Consolidadas, na Base de Dados de Conhecimento, na Base de Dados das Ontologias e nas Bases de Dados de Perfis;
- Descrição: consulta as fontes de informações baseadas nos parâmetros recebidos pelo interfaceador;
- Ingredientes: não se aplicam;
- Restrições: não tem;
- Exceções: a consulta ou requisição pode não ter informações suficientes para a análise.

Sub tarefa 10-2: Aplica Algoritmos / Soluções

- Objetivo: aplicação de algoritmos e soluções pré-definidas ou aprendidas, objetivando encontrar indícios de fraudes ou irregularidades;

- Descrição: aplicação de algoritmos, inferências, soluções de IA para a construção da base de conhecimento e análise de informações que serão transformadas em mensagens a serem repassadas ao agente, que irá determinar as ações e recomendações a serem realizadas;
- Ingredientes: soluções pré-definidas e aprendidas ao longo do tempo;
- Restrições: não tem;
- Exceções: a consulta ou a requisição feita pelo interfaceador pode não ter informações suficientes para a análise.

Sub tarefa 10.3: Constrói / altera a Base de Conhecimento

- Objetivo: construir a base de conhecimento;
- Descrição: a sub-tarefa 10-2 irá fornecer informações a serem gravadas na base de conhecimento, servindo para prover conhecimento baseado no aprendizado das inferências e ações realizadas;
- Ingredientes: a base de regras é sempre atualizada;
- Restrições: não tem;
- Exceções: não tem.

3.2.2.5 Tarefas do Agente Determinador de Ações e Recomendações

A Figura 22 ilustra as tarefas realizadas pelo agente determinador.

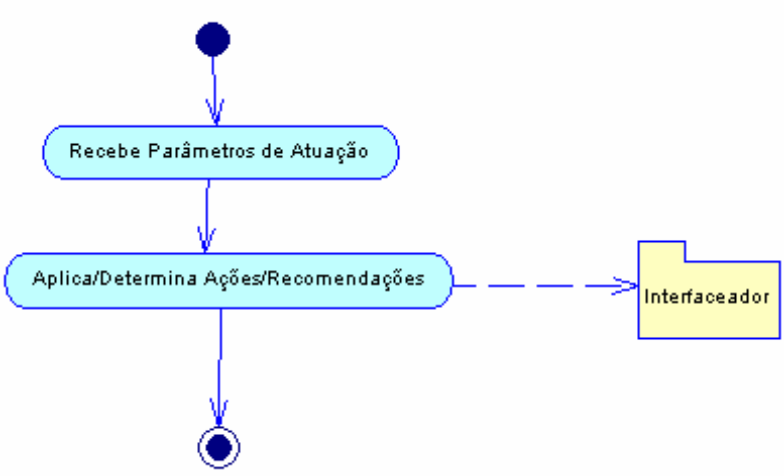


Figura 22 – Diagrama de Atividades do Agente Determinador de Ações e Recomendações.

Tarefa 11: Recebe parâmetros de atuação

- Objetivo: receber os parâmetros que irão nortear a atuação do agente;
- Descrição: recebe as informações sobre a atuação a realizar;
- Ingredientes: não tem;
- Restrições: não tem;
- Exceções: não tem.

Tarefa 12: Aplica / determina ações e recomendações

- Objetivos: aplicar o que foi determinado baseado no parâmetro / mensagem passada pelo analisador; devolver os resultados das ações realizadas ao interfaceador;
- Descrição: o agente vai apenas aplicar o que foi determinado pelo agente analisador e informar os resultados ao interfaceador;
- Ingredientes: não tem;
- Restrições: não tem;
- Exceções: não tem.

3.2.3 Modelo Organizacional

O CommonKADS define este modelo representando a organização na qual o sistema baseado em conhecimento está sendo introduzido. Aqui, o modelo é estendido da mesma maneira que o modelo de agente o é para a modelagem da organização dos agentes.

No modelo organizacional, mostra-se o relacionamento estático ou estrutural entre os agentes. Segundo Iglesias et al. [1998], usa-se a notação gráfica baseada na OMT [RUMBAUGH, 1991] para expressar esses relacionamentos, adicionando um símbolo especial para distinguir entre agentes e objetos.

O diagrama de hierarquia dos agentes do nosso estudo de caso é ilustrado na figura abaixo; o símbolo de agregação é usado para expressar grupos de agentes.

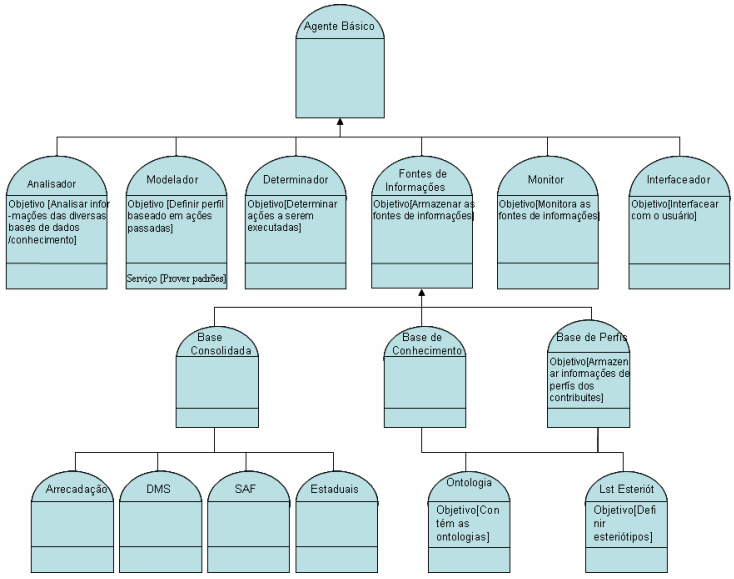


Figura 23 – Modelo Organizacional dos Agentes do Sistema Auditor.

O símbolo de agente é muito similar ao símbolo de classe proposto pela OMT, mas tem diferente significado. A caixa superior não armazena os atributos definidos, como na OMT, mas o estado mental e os atributos internos de um agente, como seus objetivos, crenças, planejamentos, etc. A caixa inferior armazena os atributos externos dos agentes, serviços, sensores e executores. O relacionamento de herança entre os agentes é definido

como a união dos valores das classes precedentes para cada atributo. Por exemplo, uma classe de agente tem seus próprios objetivos e os objetivos das classes de agentes precedentes. Se um agente definir um atributo como exclusivo, os valores serão sobrescritos.

Os benefícios potenciais do desenvolvimento deste modelo são a especificação dos relacionamentos estruturais entre humanos e/ou agentes de *software* e o relacionamento com o ambiente. O estudo da organização é uma ferramenta para a identificação de possíveis impactos do sistema multiagente, quando instalado. Da mesma maneira, este modelo pode prover informações sobre as funções, fluxos de trabalho, processos e estrutura da organização que permite o estudo da exeqüibilidade das soluções propostas. Este modelo representa tanto o diagrama de classes de agentes quanto o diagrama de instâncias de agentes, mostrando o relacionamento particular com o ambiente. Em contraste com outro paradigma, o orientado a objetos, o diagrama de instância de agente é freqüentemente mais relevante do que o diagrama de classe do agente.

3.2.4 Modelo de Coordenação

Diferentemente do modelo organizacional, o modelo de coordenação mostra o relacionamento dinâmico entre os agentes. Nele, inicia-se com a identificação das conversações entre agentes, cujos casos de uso mostram novamente uma importante regra. Nesse nível, cada conversação consiste numa simples interação e na possível resposta, as quais são descritas por significados (palavras) em modelos.

Das conversações identificadas nos diversos cenários temos algumas:

I - Entre os agentes e atores do cenário referente ao objetivo específico "Construir perfis padrões de contribuintes" – Modelo 2.

Neste modelo, identificaram-se três situações:

- **Situação 1**

- Objetivo: construir, a partir das fontes de informação, os perfis dos contribuintes;
- Agente: modelador;
- Iniciador: modelador;
- Serviço: geração de perfis;
- Descrição: o agente modelador consulta as fontes de informação (base consolidada, base de perfis e base de ontologia), aplicando critérios de criação de perfis individuais para enquadramento e a representação de grupos de perfis, inserindo e alterando informações na base de perfis (criando e alterando perfis individuais e de grupos);
- Pré-condição: informações nas fontes de informação suficientes, de pelo menos seis meses;
- Pós-condição: perfis criados e/ou alterados;
- Condição de encerramento: apenas no término da criação de perfis.

- **Situação 2**:
  - Objetivo: detectar mudanças nas fontes de informação que acarretem alterações no perfil de contribuinte, informa ao Modelador sobre as alterações encontradas;
  - Agentes: monitor e modelador;
  - Iniciador: monitor;
  - Serviço: alteração nas fontes de informação que gera a chamada para que o modelador verifique, gere ou altere os perfis individuais ou de grupo;
  - Descrição: o agente monitor detecta alterações nas fontes de informação que afetam o enquadramento de perfis, enviando mensagem

/ parâmetro ao agente modelador, para que este infira sobre as informações passadas. Após a inferência, grava informações na base de perfis;

- Pré-condição: fonte de informações alteradas;
- Pós-condição: novos perfis individuais ou de grupos gerados **;**
- Condição de encerramento: no final da geração de novos perfis ou na insuficiência de informações.

- **Situação 3**
  - Objetivo: obter informações de comportamento de contribuintes individuais ou de grupo(s), a partir da solicitação do usuário;
  - Agentes: interfaceador e modelador;
  - Iniciador: interfaceador;
  - Serviço: fornecer informações sobre o padrão de comportamento de contribuinte(s) ou de grupo(s);
  - Descrição: o usuário / administrador solicita informações sobre o perfil de comportamento de contribuinte(s) e/ou grupo(s) existentes;
  - Pré-condição: que o perfil já tenha sido gerado nas situações 1 e 2 acima;
  - Pós-condição: lista as informações solicitadas;
  - Condição de encerramento: processo abortado pelo usuário / fiscal ou ausência de informações sobre o(s) perfil(is) solicitado(s).

II - Entre os agentes e atores do cenário referente ao objetivo específico "analisar informações do contribuinte" – Modelo 3.

Neste modelo, identificaram-se duas situações:

- **Situação 1**

- Objetivo: analisar informações de possíveis fraudes ou irregularidades;
- Agentes: monitor, analisador e determinador;
- Iniciador: monitor;
- Serviço: monitora as fontes de informação, analisa e determinada recomendações e ações;
- Descrição: o agente monitor detecta alterações nas fontes de informação (base de conhecimento, base consolidada, base de perfis, base de ontologias) e envia essas informações ao analisador, que realiza inferências e, encontrando indícios de fraudes ou irregularidades, envia mensagem / parâmetro para que o agente determinador efetue as ações recomendadas. O determinador envia informações das ações ao interfaceador;
- Pré-condição: fontes de informação alteradas;
- Pós-condição: ações ou recomendações enviadas ao interfaceador;
- Condição de encerramento: no final do processo ou quando o analisador não encontrar qualquer indício de irregularidades.

- **Situação 2**
  - Objetivo: analisar a solicitação do usuário;
  - Agentes: interfaceador, analisador e determinador;
  - Iniciador: interfaceador;
  - Serviço: fornecer as informações solicitadas pelo usuário / fiscal;
  - Descrição: o ator fiscal / administrador através do interfaceador envia solicitação para o analisador, que consulta as fontes de informações, analisa-as e retorna a informação consultada ao interfaceador e, dependendo do que foi encontrado, também para o determinador;

- Pré-condição: o usuário deve efetuar a consulta através do interfaceador;
- Pós-condição: retorno das informações requisitadas;
- Condição de encerramento: processo abortado pelo usuário / fiscal ou ausência de informações.

No próximo passo, modelam-se as trocas de dados em cada interação, especificando atos de fala e tipos de sincronização. Coletam-se todas essas informações na forma de diagrama de seqüências e modelos textuais, conforme indicado acima. A segunda fase no modelo de coordenação consiste em analisar as interações de forma a determinar a sua complexidade.

Nas Figuras 24, 25 e 26, apresentam-se os diagramas de seqüências das situações de interações referentes ao Modelo 2 (“Construir perfis padrões de contribuintes”), respectivamente as situações 1, 2, 3 já descritas acima.

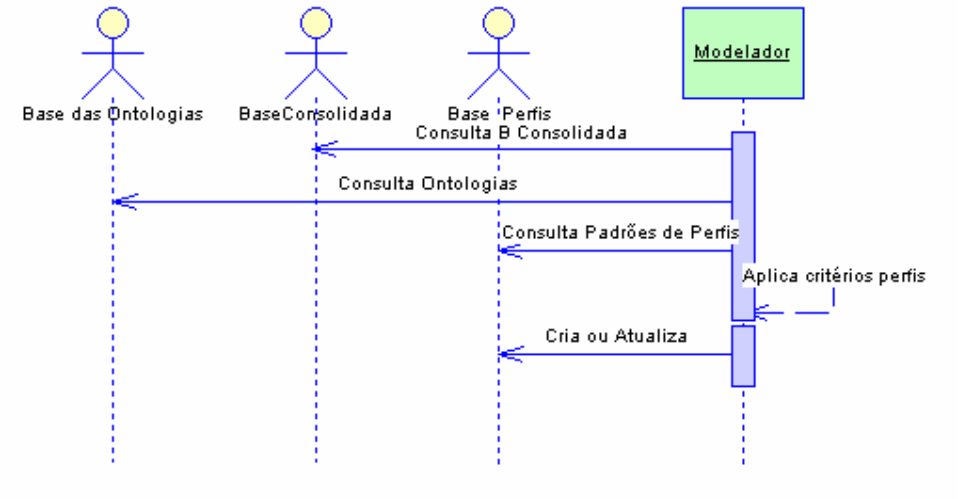


Figura 24: Diagrama de Seqüência Cenário 2 – Situação 1.

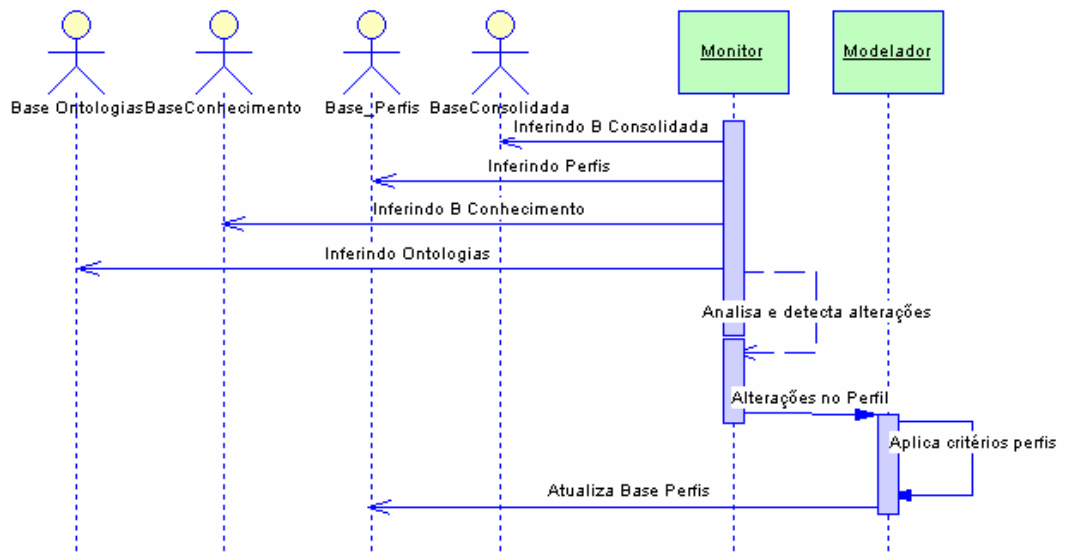


Figura 25: Diagrama de Seqüência Cenário 2 – Situação 2.

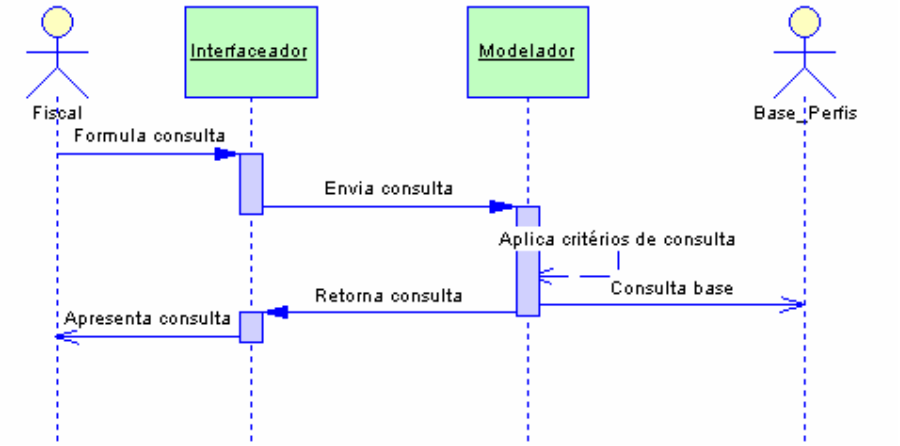


Figura 26: Diagrama de Seqüência Cenário 2 – Situação 3.

As Figuras 27 e 28 apresentam os diagramas de seqüências das situações de interações referentes ao Modelo 3 (“Analisar informações do contribuinte”), respectivamente as situações 1 e 2.

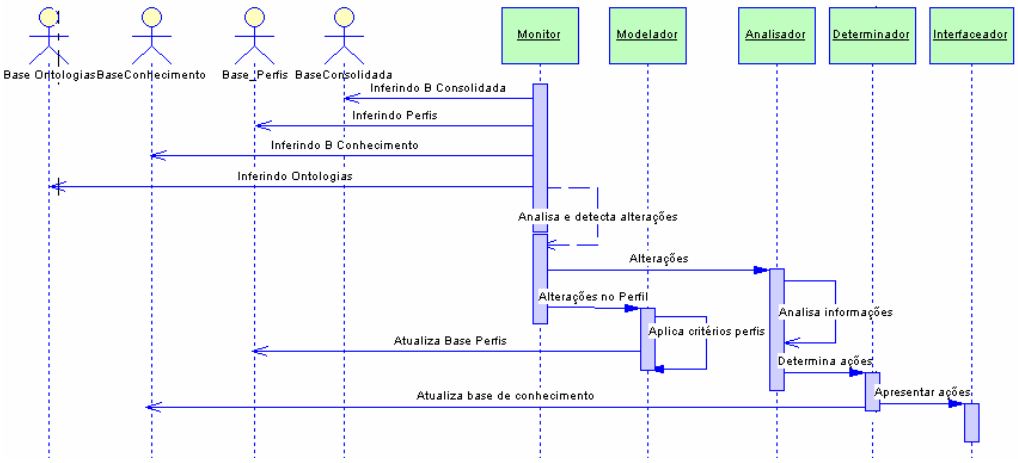


Figura 27: Diagrama de Seqüência Cenário 3 – Situação 1.

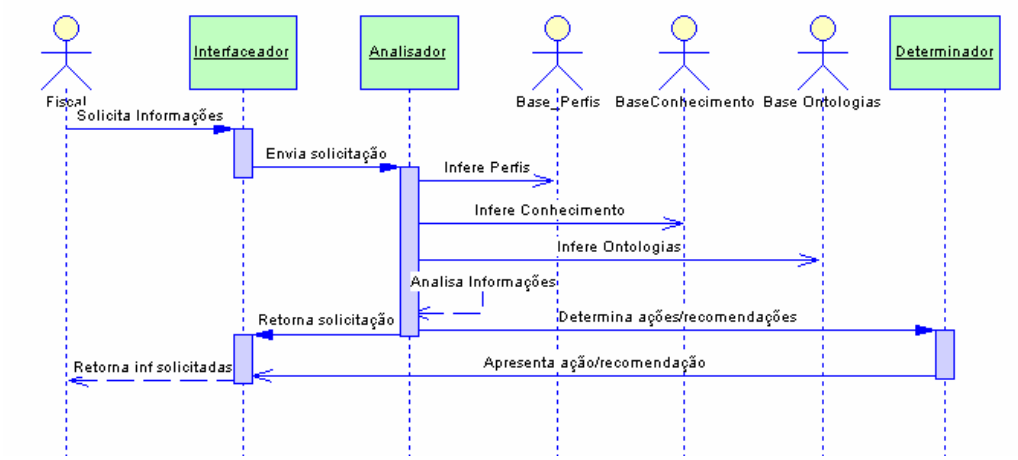


Figura 28: Diagrama de Seqüência Cenário 3 – Situação 2.

3.2.5 Modelo de Conhecimento (*expertise)*

É voltado para o desenvolvimento do conhecimento da aplicação e a definição do método de resolução de problemas.

Para o desenvolvimento do conhecimento da aplicação, determinam-se: o conhecimento do domínio, onde se definem as ontologias e modelos de domínio; o conhecimento da tarefa, que especifica o conhecimento necessário para que a tarefa atinja seus objetivos; e o conhecimento da inferência, que representa os passos de inferências necessários para resolver uma tarefa, ou seja, como um agente pode interpretar os eventos que recebe de outros agentes ou do mundo.

3.2.5.1 Conhecimento do Domínio

Representa o conhecimento declarativo do problema, modelado como conceitos, propriedades, expressões e relacionamentos, e que define a ontologia e os modelos do domínio.

A ontologia auxilia o uso da tecnologia de SMA, pois promete um entendimento comum e compartilhado do domínio, que pode ser transmitido entre pessoas e sistemas de *software* [DAVIES et al., 2003].

Uma ontologia dá um relato explícito parcial de uma conceitualização; ela é uma estrutura semântica intencional que codifica as regras explícitas, contendo a estrutura de um pedaço da realidade. A ontologia captura o conjunto de conceitos, termos e relacionamentos usados para descrever o conhecimento de sistema de *software* [DAVIES et al., 2003].

Em geral, existem duas abordagens enquanto modelando, a *top-down* e a *botton-up*. A abordagem a ser usada neste trabalho será o *top-down*. Com ela, inicia-se modelando conceitos num nível bastante genérico, refinando-os subseqüentemente.

A Figura 7 mostra uma visão macro da ontologia do sistema.

Na Figura 29 demonstra-se o mapa ontológico do sistema. O mapa utilizado foi fundamentado numa ontologia própria, utilizando como referência ontologias vistas em DAVIES [2003] e BASTOS [2003].

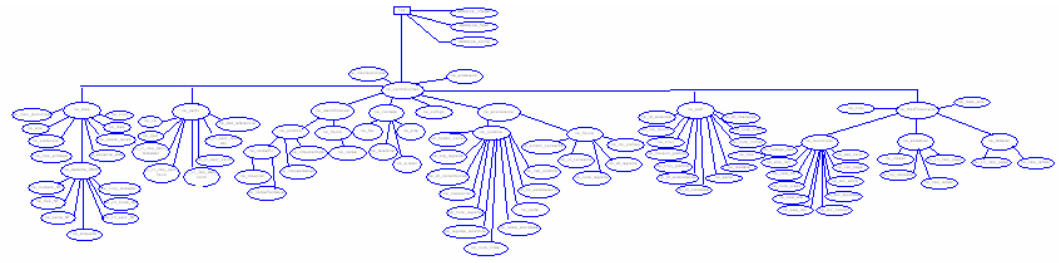


Figura 29 – Árvore ontológica do sistema.

Devido à quantidade de nós a figura acima ficou inteligível, no anexo II esta árvore está representada em um formato maior, permitindo assim a visualização dos itens.

A Figura 29, denominada árvore ontológica do sistema, possui como níveis básicos os seguintes nós:

- **nó-identificação**: representa a identificação do contribuinte através dos sub-nós **nó-jurídico** e **nó-físico**. Eles são a especificação da existência do contribuinte;

- **nó-contato**: é o meio de contato com o contribuinte, permitindo, com suas informações, acesso mais rápido ao contribuinte;

- **nó-econômico**: possui as informações econômicas do contribuinte, sendo subdividida em dois sub-nós (jurídico e físico). O nó-jurídico possui as especificações de natureza jurídica (firma individual, sociedade limitada, sociedade anônima, associação, cooperativa, fundação, sociedade em comandita, etc.), os ramos de atividade (indústria, comércio, serviço, agropecuária e pesca, indústria e comércio, etc.), o número do CNAE fiscal, o regime de recolhimento (normal, estimativo ou substituto) e detalhes da constituição da empresa, como datas de constituição e abertura, tipo de documento de constituição, órgão de registro, número de registro, capital social;

- **nó-dms**: traz dois sub-nós, com informações de declaração apresentadas pelo próprio contribuinte e pela pessoa do tomador de serviço ou contribuinte substituto;

- **nó-aidf**: possui informações detalhadas sobre todos os impressos fiscais, como numeração das notas, data de emissão e validade, série, etc.;

- **nó-perfil**: assume as informações dos perfis dos contribuintes com os dados de PSD, VS, etc, para maiores detalhes sobre estas variáveis vide item 2.5;

- **nó-historico_fin**: possui informações sobre os impostos devidos e pagos à Prefeitura.

O anexo II contém maiores informações sobre a ontologia que foi desenvolvida no Protégé.

3.2.5.2 Conhecimento das Tarefas

O conhecimento das tarefas é representado na Tabela 10 montada a partir de informações obtidas do item 3.2.2 – Modelo de Tarefas.

Tabela 10 – Conhecimento das Tarefas.

Agente	Tarefa Genérica	Conhecimento
Interfaceador	- Receber consultas / ações do usuário; - Representa ação / consulta do usuário; - Entrega resultados da consulta / ação.	- Armazena as consultas mais realizadas, criando um ranking.
Modelador	- Adquirir perfil; - Representar perfil; - Atualizar perfil.	- Infere na criação, representação e atualização de perfis individuais e de grupos.
Monitor	- Detectar mudanças nas fontes de informações; - Inferir sobre os itens de informações detectados.	- Identifica mudanças e infere sobre elas, informando as alterações aos agentes analisador e modelador.
Analisador	- Receber parâmetros para análise; - Analisar e inferir sobre os itens de informações.	- Constrói a base de conhecimento; - Aprende a partir das análises feitas.
Determinador	- Determinar ações e recomendações.	- Aprende com as ações realizadas.

3.2.5.3 Conhecimento de inferência

Para o conhecimento de inferência, usa-se um diagrama de inferência, o qual apresenta os passos de inferência feitos para a solução de uma tarefa.

No diagrama de inferência, as caixas representam fontes de informações; os ovais ilustram as inferências feitas pelo agente em questão; e as setas indicam os fluxos entre as fontes de informações e as inferências.

Deve-se construir um diagrama de inferência para cada tarefa. Porém, como se pode observar na Figura 30, existe uma grande similaridade de representação com os modelos de papéis apresentados no item 3.2.1. Seria desnecessário, portanto, fazer essa representação

para todas as tarefas usando o diagrama de inferências visto que as mesmas já foram representadas.

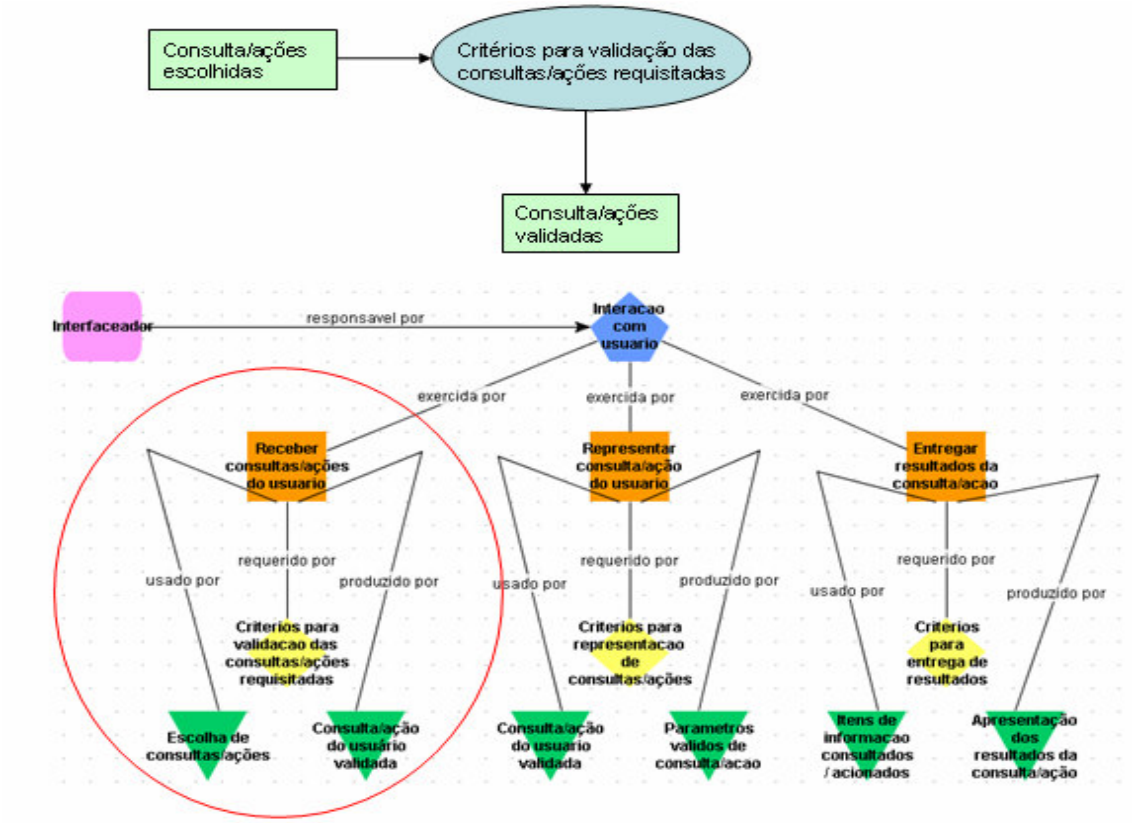


Figura 30 – Diagrama de Inferência *versus* Modelo de Papéis do Interfaceador. Tarefa: Receber consulta / ações do usuário.

Na definição do método de resolução de problemas, usou-se o ASSESSMENT como metodologia para gerar esquemas de organização do conhecimento [SCHREIBER et al., 2000]. Essa metodologia inclui a representação de conhecimento relevante, ou seja, conhecimentos que geram ações especiais no sistema. Por exemplo: notas fiscais declaradas sem a devida autorização ou contribuinte com comportamento mensal muito acima do perfil-padrão; definição de normas que garantam integridade, tais como informações de DMS sem o CNPJ correspondente. Será usada a JESS como a linguagem de representação do conhecimento. No item 4 (implementação), serão apresentadas algumas regras de representação do conhecimento implementadas em JESS e associadas ao JADE.

3.2.6 Modelo de Comunicação

No caso deste estudo, o modelo inclui apenas interações entre o fiscal (agente humano) e outros agentes. São usados modelos similares àqueles do modelo de coordenação, mas são levados em consideração fatores humanos, como as facilidades para entender as recomendações dadas pelo sistema. Este modelo não foi aprofundado neste trabalho.

3.2.7 Modelo de Projeto

Segundo Iglesias [1996], esta fase consiste em:

a) *Design* ou projeto da rede de agentes: a rede de agentes consiste num conjunto de agentes que mantêm e compõem a rede, conhecimentos e facilidades de coordenação. Algumas das facilidades requeridas podem ser:

- Facilidades de rede: serviços de nomes, serviços de páginas brancas e amarelas, criptografia e autenticação, etc.;
- Facilidades de conhecimento: servidores de ontologia, tradutores de linguagem de representação do conhecimento, etc.;
- Facilidades de coordenação: servidores de protocolos, facilidades de gerenciamento de grupos, etc.;

O resultado do compartilhamento de facilidades comuns pelos agentes permite a comunicação eficiente entre eles, sendo expressa numa ontologia. A ferramenta JADE utilizada na implementação da rede de agentes fornece todas as facilidades descritas acima.

Na Figura 31 apresenta-se o modelo de projeto do *framework* do sistema auditor[4] referente ao projeto da rede de agentes. Esse modelo representa a colaboração entre os agentes.

[4] Utilizou a ferramenta Protégé.

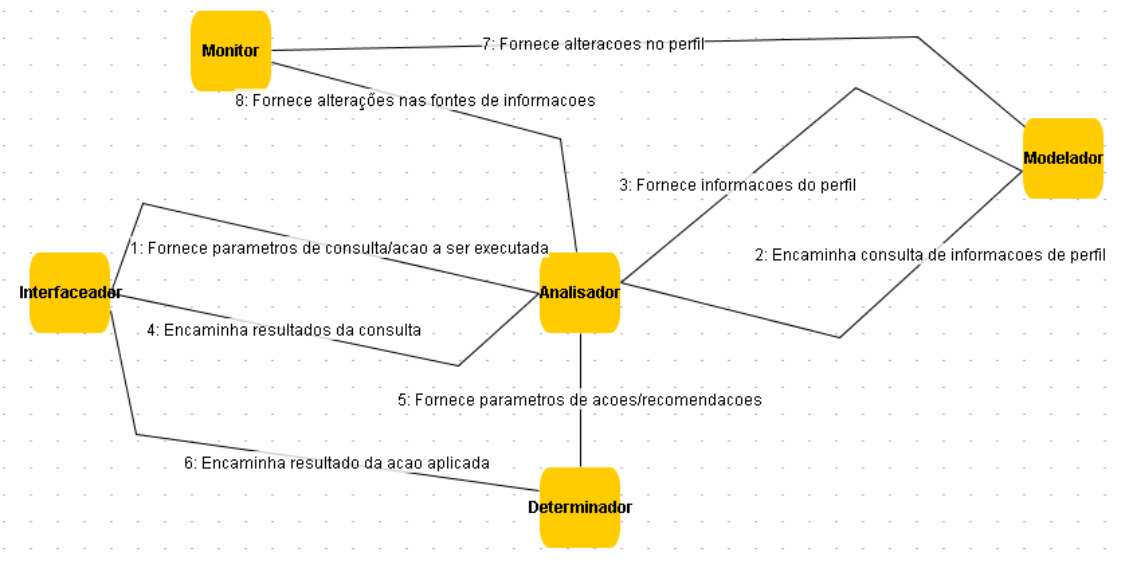


Figura 31 – Modelo de Projeto do *Framework* do Sistema Auditor.

Conforme ilustrado na Figura 31 temos:

- o agente Interfaceador (interface com o usuário) envia mensagens para o analisador (1), estas mensagens consistem de parâmetros de consultas ou de ações que devem ser executadas pelo sistema. O interfaceador recebe o retorno das mensagens do Analisador sobre as consultas realizadas (4) e mensagens referentes aos resultados das ações aplicadas pelo Determinador (6);
- o Analisador além das interações já relatadas acima com o Interfaceador, encaminha mensagens para os agentes Modelador (2) e Determinador (5);
- O agente Monitor fornece aos agentes Analisador e Modelador informações sobre alterações sofridas nas bases de dados;
- O agente Modelador recebe consulta do analisador sobre informações de perfis (2) e retorna mensagem sobre a consulta encaminhada (3).

b) *Design* ou projeto de agente: nessa fase, a arquitetura mais adequada é determinada para cada um dos agentes, alguns dos quais podem ser introduzidos ou

subdivididos. Cada agente pode ser subdividido em módulos para a comunicação com o usuário ou com outros agentes **.** Define também se o agente será deliberativo ou reativo, o conhecimento ou *expertise* de cada agente, etc.

No estudo de caso, conforme é observado na implementação, os dois agentes implementados o Analisador ou Auditor e o Determinador, serão respectivamente deliberativo e reativo. O conhecimento de cada agente foi descrito no item 3.2.1.2.4 (Analisador) e 3.2.1.2.5 (Determinador).

A Figura 32 ilustra o diagrama de classes dos agentes interagindo com as classes do JADE e as regras do JESS.

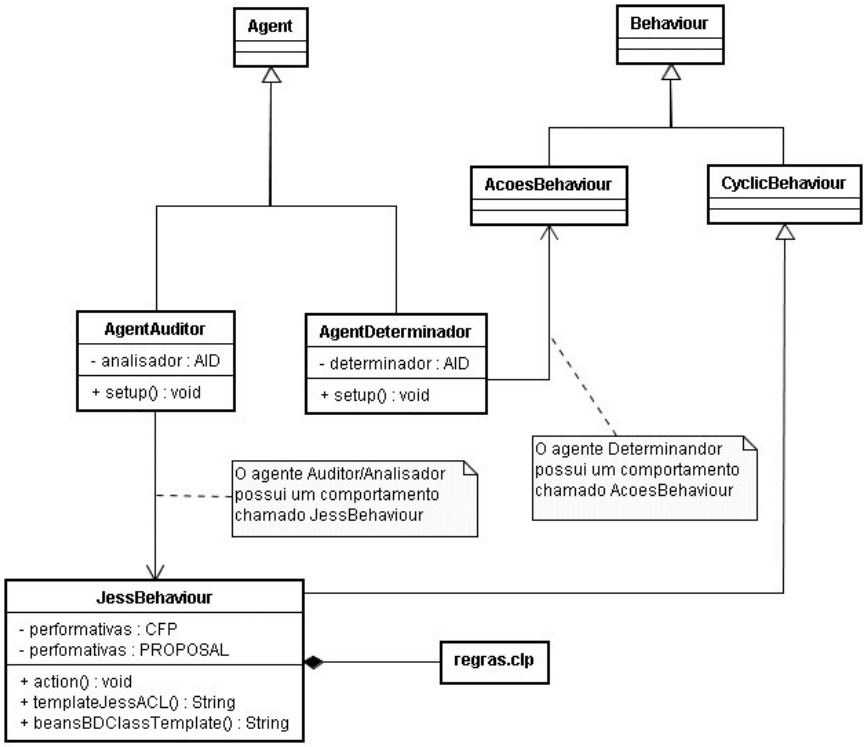


Figura 32 – Diagrama de Projeto dos Agentes Implementados.

A Figura 32 apresenta a arquitetura do sistema multiagente por meio do diagrama de classes da UML e promove uma visão global e externa da aplicação, sendo insumo necessário à implementação dos agentes. Nesta arquitetura pode-se observar os principais

aspectos envolvidos com as colaborações entre as classes dos agentes *AgentAuditor* e *AgentDeterminador*.

As classes *AgentAuditor* e *AgentDeterminador* definem os agentes que estendem a classe *Agent* do pacote *jade.core.agent* e utilizam outras classes para adicionar um ou mais comportamentos específicos a cada uma delas – pois nesta estrutura utiliza-se um comportamento do tipo cíclico e outro padrão, respectivamente para as classes do agente Analisador e Determinador.

Uma característica importante na arquitetura apresentada está na classe *JessBehaviour* . Esta classe possui como agregação uma base de conhecimento, esta base possui as regras que cobrem o processo de análise de dados e que foram coletadas através de entrevistas direcionadas aos especialistas, neste caso os auditores. Estas regras são utilizadas como informação para se inferir um conhecimento através do motor de inferência do JESS.

A classe *JessBehaviour* esta agregada ao agente Analisador (i.e. o nome do agente da classe *AgentAuditor*), visto que o Analisador possui comportamento cognitivo, e por esta razão definiu-se este agente como um agente deliberativo. Por outro lado, o agente Determinador (i.e. nome do agente da classe *AgentDeterminador*) será reativo, visto que seu comportamento será tipicamente baseado em estímulo resposta no ato de suas ações e recomendações que serão dependentes dos parâmetros passados pelo agente Analisador.

c) Projeto da plataforma: consiste na seleção do *software* de desenvolvimento do ambiente multiagente e o *hardware* necessário. Nesse caso, o desenvolvimento será feito utilizando-se o *framework* / ambiente JADE, o JESS para as regras e o Protégé para as ontologias (vide item 2).

4 IMPLEMENTAÇÃO

Com o objetivo de se validarem os modelos e o sistema, utilizaram-se dados provenientes da Prefeitura de São Luís-MA.

Para que seja possível a implementação de algumas funcionalidades do sistema de detecção de fraudes, o Código Tributário Municipal deve possuir algumas particularidades, as quais se pode citar as mais relevantes, tomando como exemplo o CTM de São Luís-MA [CTMSL, 1998].

O Código Tributário Municipal de São Luís (CTMSL) prevê, em seu artigo 10, incisos 1º, 2º e 3º, a obrigação principal, que é o pagamento do ISS (Imposto Sobre Serviço) e a obrigação acessória, que consiste na declaração dos rendimentos, quer sejam eles positivos ou negativos. O código indica também que a declaração deve ser feita por meios eletrônicos.

Na Sessão III, artigo 149, o CTMSL cria a pessoa do contribuinte substituto ou tomador dos serviços prestados, sendo este responsável pela retenção e pelo recolhimento do imposto. Os tomadores de serviços que realizarem a retenção do ISS fornecerão ao prestador de serviço recibo de retenção na fonte do valor do imposto e ficam obrigados a enviar à Fazenda Municipal as informações sobre o objeto da retenção de ISS, no prazo estipulado em regulamento. Com isso, torna-se possível efetuar o cruzamento de informações entre os tomadores e contribuintes.

Com o que foi dito nos parágrafos anteriores é possível efetuar cruzamentos entre os contribuintes e entre os contribuintes e tomadores.

O agente auditor verifica as bases de informações, apresentadas na Figura 16. Esse agente possui várias regras, das quais destacamos, a seguir, apenas sete, com as suas respectivas implementações no JESS.

1 – O contribuinte emite nota fiscal diferente da declarada pelo tomador. Essa situação é caracterizada quando o contribuinte tende a sonegar o imposto, declarando um valor menor do que o real;

2 – O contribuinte não declara a nota fiscal, mas o tomador declara. Caracteriza-se como sonegação de serviços prestados e do valor que deveria ser recolhida ao Erário Público Municipal;

3 – Documento fiscal não autorizado ou inexistente. Caracteriza-se quando o contribuinte emite notas sem a devida AIDF (Autorização para a Impressão de Documentos Fiscais), ou seja, emite notas fiscais frias;

4 – Notas fiscais emitidas em duplicidade. Um ou mais tomadores de serviços ou o contribuinte declaram notas com a mesma numeração, mas datas e valores diferentes, caracterizando a prática, por parte do contribuinte, de emissão de nota fiscal paralela ou em duplicata, na qual se trabalha com dois talonários com a mesma numeração;

5 – Notas fiscais recebidas não declaradas. Nesse caso, o tomador não informa ou declara a retenção, caracterizando apropriação indébita;

6 – Notas fiscais recebidas com diferença de retenção. O contribuinte informa que o serviço teve o imposto retido na fonte pelo tomador, mas o tomador informa valor menor;

7 – Análise comportamental do(s) contribuinte(s). É feita pelo agente modelador, baseando-se no modelo de Holt [SEGURA, 2006; MARCELO, 2006].

A seguir vamos relatar a implementação de dois agentes o Analisador e o Auditor, onde as regras de 1 a 7 citadas acima foram realizadas.

4.1 Analisador

Para a criação do agente Analisador implementou-se a classe *AgentAuditor*, de acordo com a arquitetura imposta no diagrama de classes da Figura 32, estendendo-se a superclasse *Agent* e sobrescrevendo-se o seu método *setup* que é o método mais importante da classe e foi onde se implementou o código de configuração inicial do agente e das mensagens, vide a Figura 33.

```
package auditor;
import comportamentos.JessBehaviour;
import jade.core.*;
import jade.lang.acl.ACLMessage;

public class AgentAuditor extends Agent{

     private String msg;
     ...
     public void setup(){
        myGui = new AuditorGui(this);
        addBehaviour(new JessBehaviour(this,"regras.clp",1));
            ...
       }
     ...
      }
public void sendMessage(String msg) {
     ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST);
     aclMessage.addReceiver(new AID("Determinador",AID.ISLOCALNAME));
     aclMessage.setContent(msg);
     this.send(aclMessage);
}
```

Figura 33 – Trecho de código da implementação do Agente Analisador.

Conforme mencionado no item 3.2.7 – Modelo de Projeto, o agente Analisador é um agente deliberativo que utiliza como comportamento a classe *JessBehaviour* composta por um motor de inferência baseado no JESS. O Analisador tem como responsabilidade deduzir o que vai acontecendo com os contribuintes e tomadores, no ato de suas atividades declaradas, a partir de leituras na base de dados de declarações municipais - DMS.

Observa-se também, na Figura 33, que a classe *JessBehaviour* ao ser invocada pelo agente Analisador irá inferir os resultados de acordos com as regras presentes no arquivo "regras.clp", que é a base de conhecimento deste agente, e onde o mesmo irá utilizar para

inferir seu processo de deliberação. A base de conhecimento tem sua estrutura representada no formalismo simbólico de acordo com as regras apresentada na Figura 34.

Os códigos da implementação estão apresentados no Anexo III.

```
;*****************************************************************************************************;
;                                        Regras de Decisão                                            ;
;*****************************************************************************************************;
(defrule regra1
    (mensagemACL (remetente ?rmt))
    (test (neq ?rmt (trataBeansBD CONTRIBUINTE VALORNF ?rmt VALORNF)))
     =>
    (enviaMSG "Sonegação de imposto")
    (printout t "Sonegação de imposto" crlf)
    (printout t "O Contribuinte emitiu um valor menor do que o real" crlf)
)
(defrule regra2
    (mensagemACL (remetente ?rmt))
    (mensagemACL (conteudo ?cont))
    (test (neq ?cont (trataBeansBD CONTRIBUINTE NF ?cont  NF)))
    (test (eq ?cont (trataBeansBD TOMADOR NF ?cont  NF)))

     =>
    (enviaMSG "Sonegação de servicos prestados")
    (printout t "Sonegação de servicos prestados" crlf)
    (printout t "O Contribuinte não declarou a Nota Fiscal" crlf)
)

(defrule regra3
    (mensagemACL (remetente ?rmt))
    (test (neq "SIM" (trataBeansBD NOTAFISCAL AUTORIZADA 'SIM' AUTORIZADA)))
     =>
    (enviaMSG "Documento Fiscal não autorizado ou inexistente")
    (printout t "Documento Fiscal não autorizado ou inexistente" crlf)
    (printout t "O Contribuinte emitiu notas sem Autorização para Impressão de Documentos Fiscais" crlf)
)
```

Figura 34 – Regras de produção feita em JESS para o modulo de decisão do Agente Analisador.

Uma característica importante na estrutura de regras do agente Analisador é tanto no padrão LHS (*Left Hand Side*), quanto no padrão RHS (*Right Hand Side*) visto que se tem possibilidade em definir algumas funções que podem ser chamadas para retornar os valores necessário ao sistema, como é o caso da função *trataBeansBD*, vista na Figura 35.

Observando esta função, apresentada na Figura 35, nota-se em seus comandos internos, propriedades da tecnologia *JavaBeans,* e através dos métodos *set* e *get*, da classe BeansBD, que as informações (i.e. os *beans* da classe) a serem recuperadas na base de dados de declarações dos contribuintes são feitas por meio de instruções SQL (*Structure Query*

Language), tendo como retorno o valor da tupla ou coluna alvo de informação que o agente Analisador procura.

```
(deffunction trataBeansBD (?tab ?colCons ?colFiltro ?colRes)
    (bind ?bBD (new BeansBD))
    (definstance myBeans ?bBD)
    (call ?bBD setConsulta ?tab ?colCons ?colFiltro ?colRes)
    (bind ?rr (call ?bBD getConsulta))
    (return ?rr))
```

Figura 35 – Função TrataBeansBD

Com o uso da técnica *JavaBeans*, a função *trataBeansBD* realiza a consulta na base de dados acessando a classe *BeansBD* pela instância *myBeans* que é amarrada a uma variável *?bBD* para prover referencia aos *beans* da classe. Informações sobre integração do *JavaBeans* com *Jess* podem ser obtidas no em FRIEDMAN-HILL [2006].

4.2 Determinador

A implementação do agente Determinador tem características de um agente baseado em ação-reflexão. A estrutura do código para a criação deste agente segue basicamente o mesmo utilizado pelo agente Analisador, diferenciando-se no comportamento apresentado por este por se tratar de um agente do tipo reativo. A Figura 36 demonstra parte do código do agente Determinador quando invocado o seu método *setup*.

```
package determinador;
import comportamentos.*;
import jade.core.AID;
import jade.core.Agent;
import jade.lang.acl.ACLMessage;
import gui.DeterminadorGui;

public class AgentDeterminador extends Agent{

      private String msg;
      protected AgentDeterminador myAgent;
      ...

      public void setup(){

         myGui = new DeterminadorGui(this);
         addBehaviour(new Receptor(this);

            ...

          }
            sendMessage(msg);
       }
     public void sendMessage(String msg) {
        ACLMessage aclMessage = new
        ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
        aclMessage.addReceiver(new AID("Analisador",AID.ISLOCALNAME));
        aclMessage.setContent(msg);
        this.send(aclMessage);
     }
```

Figura 36 – Trecho de código para a implementação do Agente Determinador.

Com as estruturas das classes mapeadas para a implementação, foi possível representar as interações entre os agentes Analisador e Determinador. Os próximos esforços empenhados no desenvolvimento do sistema fornecem uma projeção das mensagens trocadas entre os agentes Analisador e Determinador, através do diagrama de seqüência e interações entre eles, modelando e mapeando a solução computacional para os relacionamentos vistos nos diagramas da etapa de análise.

As mensagens da Figura 37 são caracterizadas pelas intenções de comunicação entre os agentes. No ambiente JADE estas mensagens são ditas performativas de comunicação, ou seja, o protocolo que define um conjunto de possíveis mensagens que podem ser trocadas entre os agentes. Maiores detalhes a cerca das performativas de comunicação podem ser encontrados em Caire [2003].

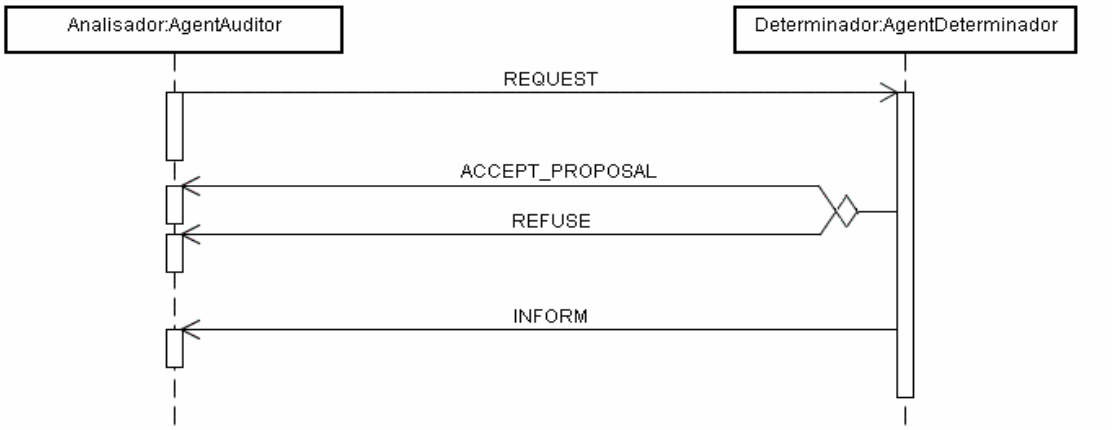


Figura 37. Comunicação entre os agentes

A mensagem do tipo REQUEST enviada pelo agente Analisador solicita ao agente Determinador que este determine as ações e recomendações a serem tomadas, baseadas nas análises inferidas por aquele. O agente Determinador pode escolher entre enviar uma mensagem ACCEPT_PROPOSAL, indicando que aceita executar a operação solicitada pelo agente Analisador, ou recusar enviando-lhe a mensagem REFUSE.

A mensagem INFORM pode ser usada para indicar uma proposição verdadeira, no caso deste agente decidir que um determinado contribuinte deva ser autuado por sonegar impostos.

Assim, uma vez demonstradas as interações e colaborações entre os agentes Analisador e Determinador, agora através da ferramenta de administração *rma* e *sniffer*, verificam-se as mensagens trocadas entre eles, vide Figura 38.

O *rma* é um agente do JADE, e é responsável por fornecer funcionalidades por meio de um *menu bar* onde se pode gerenciar os agentes do contêiner principal. Ressalta-se ainda, que tanto o *rma* quanto o *sniffer* são agentes que só afetam o próprio *framework* JADE e não influem na estrutura dos demais agentes.

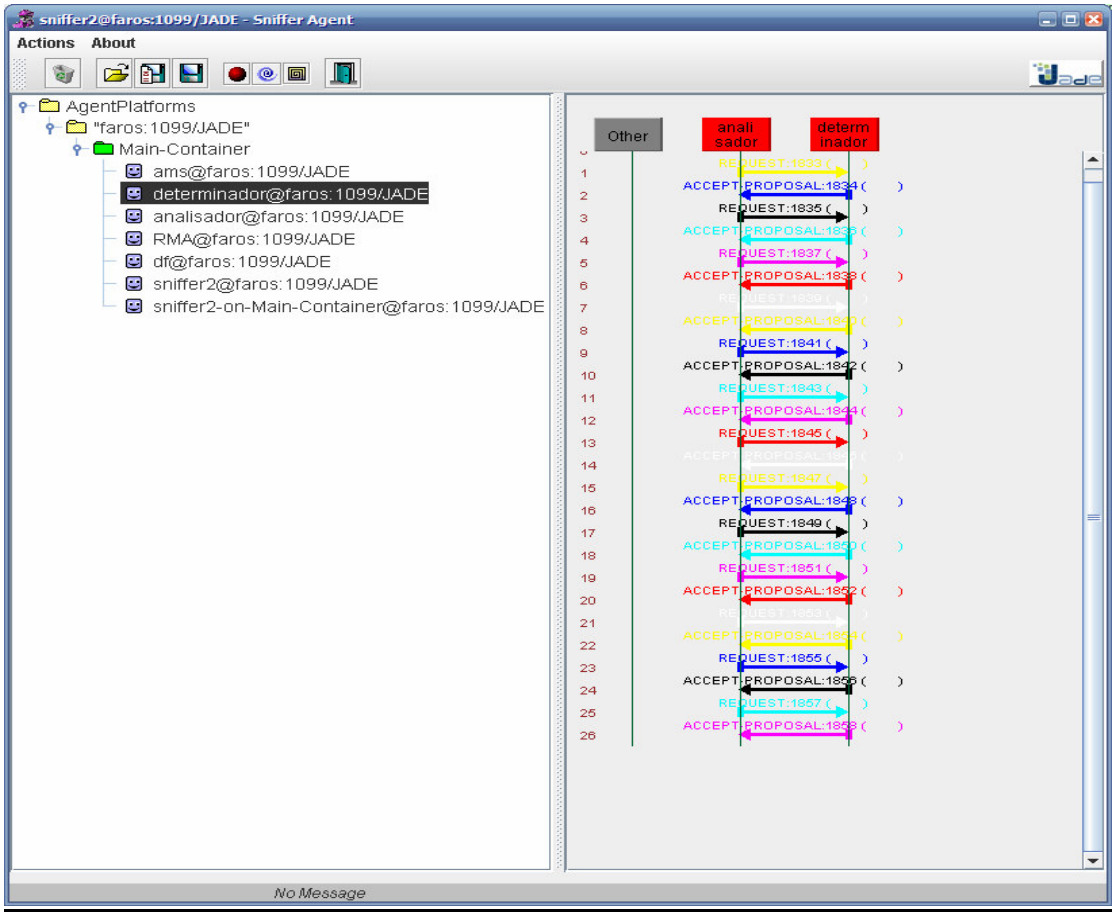


Figura 38. Trocas de mensagens entre os agentes Analisador e Determinador

As mensagens trocadas entre os agentes podem ser verificadas também através de linhas de comando, a Figura 39 demonstra parte destas mensagens. Nesta interação o agente Analisador envia uma mensagem contento um **CNPJ***MES***ANO***TP*** representada por "**04552245500019***9903***200***601***" que é recebida pelo agente Determinador. Neste caso temos o CNPJ do contribuinte, o mês e o ano e o tipo de irregularidade encontrada (divergência no valor da nota fiscal declarada).

```
03/04/2007 13:54:44 jade.core.Runtime beginContainer
INFO: ---------------------------------
   This is JADE 3.3 - 2005/03/02 16:11:05
   downloaded in Open Source, under LGPL restrictions,
   at http://jade.cselt.it/
---------------------------------------
03/04/2007 13:54:45 jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
03/04/2007 13:54:45 jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
03/04/2007 13:54:45 jade.core.messaging.MessagingService boot
INFO: MTP addresses:
http://EGITO:7778/acc
03/04/2007 13:54:45 jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
03/04/2007 13:54:45 jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
03/04/2007 13:54:45 jade.core.AgentContainerImpl joinPlatform
INFO: ----------------------------------------
Agent container Main-Container@JADE-IMTP://CLAUDIO is ready.
---------------------------------------------
Agente:Determinador
Agente:Analisador
Determinador: Estou recebendo a seguinte mensagem:
(REQUEST
 :sender  ( agent-identifier :name Analisador@CLAUDIO:1098/JADE  :addresses (sequence
http:// CLAUDIO:7778/acc ))
 :receiver  (set ( agent-identifier :name Determinador@CLAUDIO:1098/JADE ) )
 :content  "04552245500019903200601"
)
```

Figura 39 – Mensagens entre os Agentes através de linhas de comando.

4.3 Resultados obtidos

A parte da inferência do modelo comportamental dos contribuintes foi testada usando o modelo estatístico de Holt, visto no item 2.5.

Foram feitas simulações de comportamentos de contribuintes do ISS da Prefeitura de São Luís, tomando-se como medida de comparação para os métodos os valores globais de entradas, arrecadações e sanções aplicadas aos contribuintes por não cumprimento da obrigação tributária. Estes valores globais (dados reais) foram comparados ao somatório dos valores individuais previstos para cada contribuinte. Foram feitas simulações para previsão dos valores dos atributos do vetor Vei dos contribuintes nos instantes t+1 e t+6, ou seja, um mês e seis meses após o mês atual.

A Tabela 11 mostra os valores dos erros médios obtidos para as previsões das séries históricas para o ano de 2006 referente ao atributo Vif (valor do imposto detectado pelo fisco), usando como entradas as informações obtidas até um mês atrás e até seis meses atrás.

Tabela 11 – Erro médio de Vif obtido

Mês	Holt	Holt	Mês	Holt	Holt
	1 passo a frente	6 passos a frente		1 passo a frente	6 passos a frente
6Jan	4,18	27,05	**Jul**	6,80	27,63
Fev	4,64	29,66	**Ago**	9,27	27,11
Mar	5,98	30,33	**Set**	7,63	26,81
Abr	6,17	28,69	**Out**	4,44	31,41
Mai	3,78	27,32	**Nov**	3,16	25,23
Jun	5,32	32,91	**Dez**	3,63	28,31
			Média	5,42	28,54

Os erros médios obtidos apresentaram um percentual variando de 3,16% a 9,27% para o modelo de Holt um passo à frente e de 25,23% a 32,91% para o modelo de Holt seis passos à frente. Os resultados mostraram maior precisão para as previsões de um passo a frente, ou seja, para o próximo mês. Com a validação do modelo comportamental proposto, pôde-se implementar o agente modelador, tendo-se como base os erros médios obtidos.

Foram realizadas, também, análises a partir de dados dos contribuintes da Prefeitura de São Luís no período de janeiro a dezembro de 2006. Os critérios adotados para a seleção automática de contribuintes para a fiscalização foram os maiores índices de: 1 – sonegação de serviços prestados; 2 – pagamento menor que o declarado; 3 – notas fiscais frias; 4 – quantidade ou tipo de ocorrências; 5 – divergência entre a análise comportamental e o padrão histórico.

Inúmeras irregularidades foram encontradas, de acordo com as regras enumeradas acima. A Tabela 12 apresenta a síntese do que foi encontrado.

Tabela 12 – Quantidade de irregularidades encontradas

Período / Critério	1º Trimestre 2006	2º Trimestre 2006	1º Trimestre 2006	2º Trimestre 2006
1 e 2	1241	1253	1293	667
3	169	178	253	72
5	1036	923	868	430

Existem 24.622 contribuintes ativos e em situação regular perante o fisco municipal. Destes, 153 são contribuintes substitutos ou tomadores de serviços devidamente regulamentados. Os contribuintes substitutos são grandes empresas tomadoras de serviços que, por determinação de decreto municipal, ficam obrigadas a reter o ISS na fonte, repassando-o posteriormente à Prefeitura.

5 CONCLUSÃO

Neste trabalho, foi proposta a modelagem e a construção de uma comunidade de agentes inteligentes, objetivando a detecção de fraudes em impostos municipais. Para a modelagem do SMA utilizou-se metodologias baseadas em agentes, e para validar o modelo proposto implementou-se os agentes auditor e determinador. Construiu-se, também, a ontologia e as regras de inferência.

Propôs-se ainda, um modelo comportamental genérico de usuários, baseado na inferência de ações futuras a partir de ações passadas desses usuários. O modelo em questão foi aplicado a um tipo especial de usuários, os contribuintes de sistemas fiscais. O resultado desta proposta é um modelo comportamental baseado em funções de inferência comportamental. Para a obtenção dessas funções foi proposta a abordagem baseada em métodos estatísticos clássicos.

Os dados para os testes foram feitos com autorização da Secretaria de Fazenda de São Luís, onde tivemos o cuidado de não revelar informações confidenciais dos contribuintes.

Com os resultados obtidos na fase de implementação, viu-se o objetivo geral ser atendido, visto que a inteligência de decisão no processo de detecção de fraudes foi colocada em uma base de conhecimento, onde se pôde inferir sobre elas, além do que foi possível construir o perfil individual dos contribuintes baseados em ações passadas utilizando-se os métodos estatísticos propostos.

As dificuldades encontradas na realização deste trabalho foram todas na fase de implementação, inicialmente por se conhecer pouco a ferramenta JADE.

Sugere-se, para trabalhos futuros, a melhoria do módulo auditor, com a implementação de novas regras e dos agentes modelador e monitor das fontes de informação (que constitue um módulo do agente auditor).

Este trabalho com todos os módulos concluídos pode servir como uma ferramenta importante para o fisco municipal, podendo ser adaptado ao fisco estadual.

REFERÊNCIAS

[ARMSTRONG, 1999] ARMSTRONG, J. S.: Principles of forecasting: a handbook for researchers and practitioners, Kluwer, Philadelphia, (1999).

[BALLIM, 1991] Ballim, Afzal and Wilks, Yorick: Beliefs, stereotypes and dynamic agent modeling. User Modelling and User-Adapted Interaction, 1(1):33-65, 1991.

[BARRETO, 2005] BARRETO, Alexandre Serra: Previsão De Comportamento e Classificação de Contribuintes Tributários: Uma Abordagem por Modelos Lineares Generalizados Hierárquicos, Qualificação de Doutorado, Universidade Federal de Santa Catarina, Florianópolis-SC, Brasil, Setembro (2005).

[BASTOS, 2003] BASTOS, Othon C. B. F. Modelagem do Usuário para o Sistema ICS de Comércio Eletrônico, Qualificação de Mestrado, Universidade Federal do Maranhão-UFMA, 2005.

[BOND and GASSER, 1988] Alan H. Bond and Les Gasser. An analysis of problems and research in DAI. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 3-36. Morgan Kaufmann Publishers: San Mateo, CA, 1998.

[BOUNABAT, 2001] BOUNABAT, B. "A New Formal Approach for the Specification and the Verification of Multi-agent Reative System Operating Modes". In: Information Processing and Technology, pp. 25-47, Nova Science Publishers, Inc. New York, USA, 2001.

[BRAZ, 2001] Braz, Eugênio Rubens Cardoso. Um Modelo para Gerenciamento, Avaliação e Planejamento da Arrecadação de Tributos Estaduais, Qualificação de Doutorado, Universidade Federal de Santa Catarina, Florianópolis-SC, Brasil, Abril 2001.

[CAIRE, 2003] Caire, Giovanni. JADE Tutorial - *JADE Programming for Beginners* - http://jade.tilab.com. JADE Board, 2003.

[CHIN, 1988] Chin, David Ngi. Intelligent Agents as a Basis for Natural Language Interfaces. PhD thesis, Computer Science Division (EECS), University of California, Berkeley, CA94720, jan, 1988.

[CORVALÃO, 2002] Corvalão, Eder Daniel. Previsão da arrecadação do imposto sob circulação de mercadorias e serviços em Santa Catarina: Aplicação da Abordagem Geral para Específico em Modelos Dinâmicos, Qualificação de Mestrado, Universidade Federal de Santa Catarina, Florianópolis-SC, Brasil, Setembro, 2002.

[CTML, 1998] Código Tributário do Município de São Luís - MA, Secretaria Municipal de Fazenda, www.semfaz.saoluis.ma.gov.br/legislacao/ctm/ctmc.pdf , 1998.

[CTN, 1966] Código Tributário Nacional, Secretaria da Receita Federal do Brasil, www.receita.fazenda.gov.br/Legislacao/CodTributNaci/ctn.htm, Brasília – DF, Brasil, Outubro, (1966).

[DAVIES at al, 2003] Davies, J., D. Fensel and F. Van Harmelen, 2003, Towards the Semantic Web: Ontology-driven Knowledge Management. John Wiley & Sons, LTD, West Sussex, England.

[FARIA, 2004] Faria, C. Uma Técnica para a Aquisição e Construção de Modelos de Domínio e Usuários baseados em Ontologias para a Engenharia de Domínio Multiagente,

Dissertação (Mestrado em Engenharia de Eletricidade) – Área de Ciência da Computação, Departamento de Engenharia Elétrica, Universidade Federal do Maranhão – UFMA, 2004.

[FEBER, 1999]. Ferber, Jacques. **Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence**, Addison-Wesley Pub Co; 1st Edition, February 25 1999.

[FININ, 1986] FININ, T. W. and DRAGER, D. A general user modeling system. Proc. of the 6th Canadian Conference on Artificial Intelligence, Montreal, Canada, pp 24-29, 1986.

[FININ et al, 1997].Finin, T., LABROU, Y., and MAYFIELD, J. **KQML as an agent communication language,** Computer Science Department. University of Maryland Baltimore County. Baltimore, USA, 1997.

[FIPA, 2005]. FIPA – Foundation for Intelligent Physical Agents, Disponível em: http://www.fipa.org, Acessado em: maio de 2005.

[FRANZONI, 1998] FRANZONI, Luigi Alberto: Tax evasion and tax compliance, Encyclopaedia of Law and Economics, University of Bologna, Itália, September, (1998).

[FRIEDMAN-HILL, 2006] Friedman-Hill, E. J., JESS The Rule Engine for the Java Plataform - Language Reference – version 7.0b7 (11 may 2006) DRAFT, Sandia National Laboratories. Livermore, CA, USA.

[GIRARDI, 2004]. Girardi, R. 2004. Engenharia de Software Baseada em Agentes. Congresso Brasileiro de Ciência da Computação, Itajaí, 2004, 913-937.Itajaí,SC-Brasil, ISSN 1677-2822.

[GIRARDI and FARIA, 2003] Girardi, R, and Faria, C. **A Generic Ontology for the Specification of Domain Models**, In Proceedings of the 1st International Workshop on Component Engineering Methodology (WCEM 2003) at Second International Conference on Generative Programming and Component Engineering, Efurt, Germany, Ed. Sven Overhage and Klaus Turowski, pp. 41-50, 2003.

[GROSSE, 1998] Grosse, A., presented as position paper at First SIG Meeting, September 24 & 25, 1998, Brussels, Belgium.

[IGLESIAS, 1996] C.A. Iglesias, Mercedes Garijo, José C. González, and Juan R. Velasco. A methodological proposal of multiagent systems development extending CommonKADS. In B. Gaines and M. Musen, editors, Proceeding of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, volume 1, pages 25-117, Banff, Canada, November 1996. KAW. Track Agent-Oriented Approaches to Knowledge Engineering.

[IGLESIAS et al, 1998] C. Iglesias, M. Garijo, J. Centeno-Gonzalez, J. R. Velasco. Analysis and Design of Multiagent Systems Using MAS-CommonKADS. Agent Theories, Architectures and Languages. Lecture Notes in Artificial Intellegence. Vol. 1365, pages 313-326. Springer-Verlang. 1998.

[JADE, 2005].JADE - Java Agent Development Framework, Disponível em: http://jade.cselt.it, Acessado em: maio de 2005.

[JAT LITE] Jat Lite. Disponível em: http://java.stanford.edu/, Acessado em: maio de 2005.

[JENNINGS, 2000] JENNINGS, N. R. **On Agent-based Software Engineering**, *Artificial Intelligence*, v. 117, pp. 277-296, 2000.

[KOBSA, 1989] Kobsa, A. A taxonomy of beliefs and goals for user models in dialog systems.-In: A. Kobsa and Wahlster (eds). User Models in Dialog Systems. Spring-Verlag, Berlin, Heidelberg, pp. 52-68, 1989.

[KOBSA, 1993] Kobsa, A. User Modeling: Recent Work, Prospects and Hazards. In: M. Schneider-Hufschmidt, T. Kühme, U. Malinowski, eds.: Adaptive User Interfaces: Principles and Practise. Amsterdam: North Holland Elsevier, 1993.

[KOBSA, 1999] Kobsa, A. Personalised Hypermedia Presentation Techniques for Improving Online Customer Relationships, GMD Report 66, 1999.

[LIEBEL, 2004] LIEBEL, Marlon Jorge: Previsão de Receitas Tributárias – O Caso do ICMS no Estado do Paraná, Qualificação de Mestrado, Universidade Federal do Rio Grande do Sul - RS, Brasil, (2004).

[NOGUEIRA, 2006] NOGUEIRA, Marcelo Luis Lobato: Modelagem comportamental de usuários: abordagem aos contribuintes fazendários, Dissertação (Mestrado em Engenharia de Eletricidade) – Área de Computação. Universidade Federal do Maranhão – UFMA, 2006.

[ODELL, 2002] ODELL, J., PARUNAK, H. D., and BAUER, B. **Extending UML for agents,** In Proceedings of the 2nd Int. Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS 2000), Austin, USA, pp. 3–17, 2000.

[PERRAULT, 1978] PERRAULT, C. R., ALLEN, J. F. and COHEN, P. R. Speech acts as a basis for understanding dialogue conference. Report 78-5, Department of Computer Science, University of Toronto, Canada, 1978.

[POGGI, RIMASSA & TOMAIUOLO, 2001] POGGI, Agostino; RIMASSA, Giovanni; TOMAIUOLO, Michele. **Multi-user and security support for multi-agent systems**. Artigo - Dipartimento di Ingegneria dell'Informazione, Università di Parma, Parma, Italy, 2001. Disponível em : http://sharon.cselt.it/projects/jade/papers/WOA2001.pdf.

[POGGI, RIMASSA & TURCI, 2000] .POGGI, A.; RIMASSA, G. ; TURCI, P.. **An Object-Oriented framework to realize agent systems**. Artigo - Dipartimento di Ingegneria dell'Informazione, Università di Parma, Parma, Italy. 2000.

[RICH, 1979] Rich, Elaine. User modeling via stereotypes. Cognitive Science, 3:329-354, 1979.

[RICH, 1983] Rich, Elaine. Users are individuals: Individualizing user models. International Journal of Man-Machine Studies, 18:199-214, 1983.

[RUMBAUGH, 1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen. Object-Oriented Modeling and Design. Prentice Hall. 1991.

[RUMBAUGH, 1999] J. Rumbaugh, I. Jacobson, G. Booch. The Unified Modeling Language Reference Manual. Addison Wesley 1999.

[RUSSEL e NORVIG, 2004] RUSSEL, S. and NORVIG, P. **Artificial Intelligence: A Modern Approach**, Prentice-Hall, 2004.

[SCHRIBER et al, 2000] G. Schreiber et al, Knowledge Engineering and Management: The CommonKADS Methodology. Cambridge, Massachussets: A. Bradford Book, 2000. 455p.

[SEGURA, 2006] SEGURA, José V. Sevilla: A arrecadação potencial como meta da administração tributária, Centro Interamericano de Administrações Tributárias – CIAT, 40a. Assembléia Geral, Florianópolis – SC, Brasil, (2006).

[SHARMA, 2001] SHARMA, Amit. A Generic Architecture for User Modeling Systems and Adaptive Web Services, In: Workshop on E-Business & the Intelligent Web, 2001.

[SILVA, 2003] SILVA, Cláudio Antônio. Modelagem comportamental para agentes autônomos em ambientes reais, Qualificação de Mestrado, Universidade do Estado do Rio de Janeiro, Rio de Janeiro – RJ, Brasil, Setembro 2003.

[SILVA FILHO, 2004] SILVA FILHO, José Henrique A. ONTOCADE: **Um ambiente case baseado em Ontologias para Análise e Projeto na Engenharia de Domínio Multiagente**, Dissertação (Mestrado em Engenharia de Eletricidade) – Área de Ciência da Computação, Departamento de Engenharia Elétrica, Universidade Federal do Maranhão – UFMA, 2005.

[SODRÉ, 2002] SODRÉ, A. C. S and Girardi, R. **A Methodology for Multi-Agent Application Development,** In 6th International Conference on Intelligent Tutoring Systems, Proceedings of the ITS´2002 Workshops - Architectures and Methodologies for Building Agent-Based Learning Environments, Biarritz, v. 1, pp. 58-66, 2002.

[VARFIS, 1990] VARFIS, A., VERSINO, C., Univariate Economic Time Series Forecasting, Cambridge University Press, Cambridge, 1990.

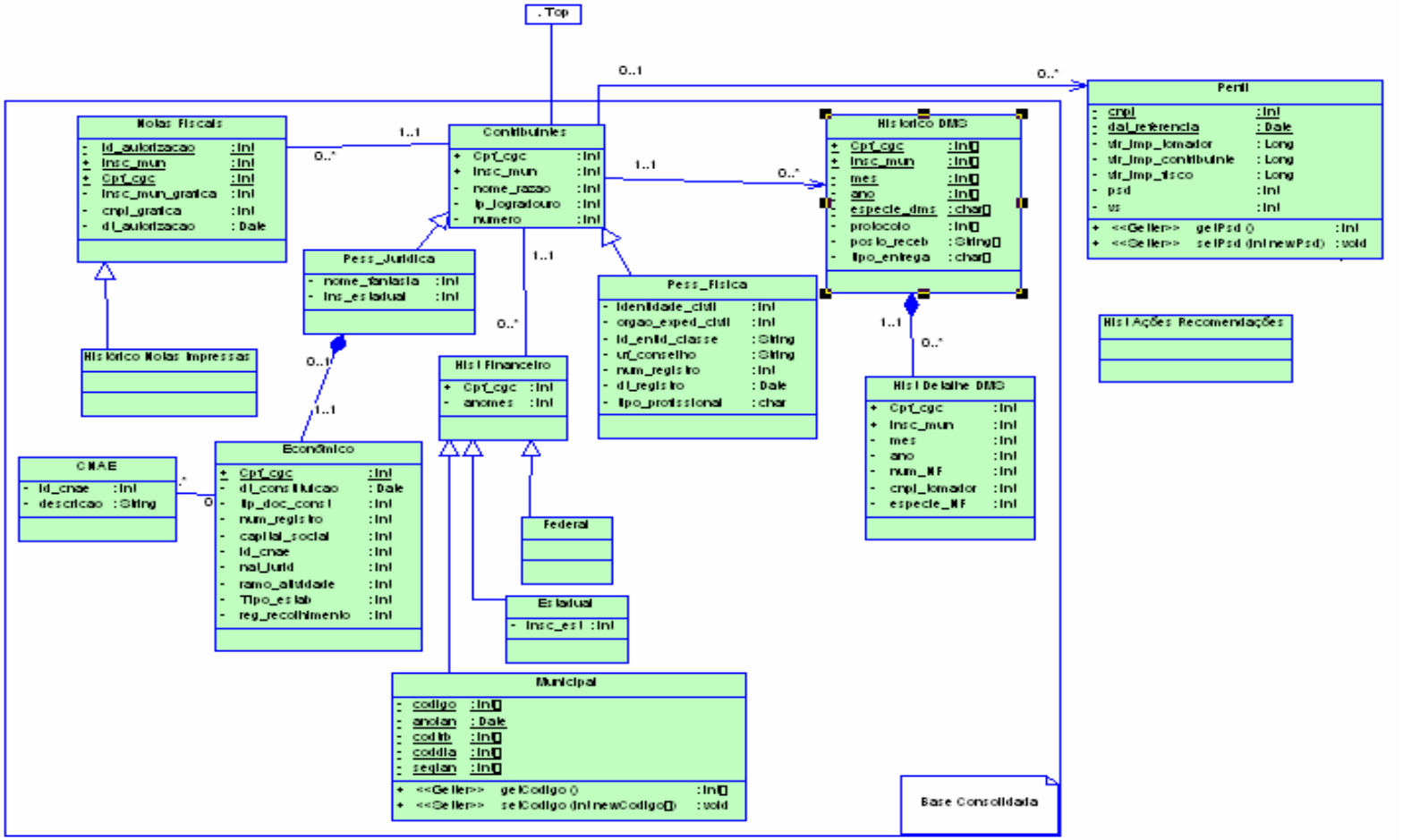
[WEISS, 1999] WEISS, Gerhard.Multiagent Systems.A modern approach to distributed artificial inteligence.MIT Press. Cambridge Massachussetts. 1999.

[WEYNS, 2005]. Weyns Danny; Parunak H.Van Dyke;Michel, Fabien. Environment for Multiagent Systems. State-of-the-Art and Research Challenges.2005.

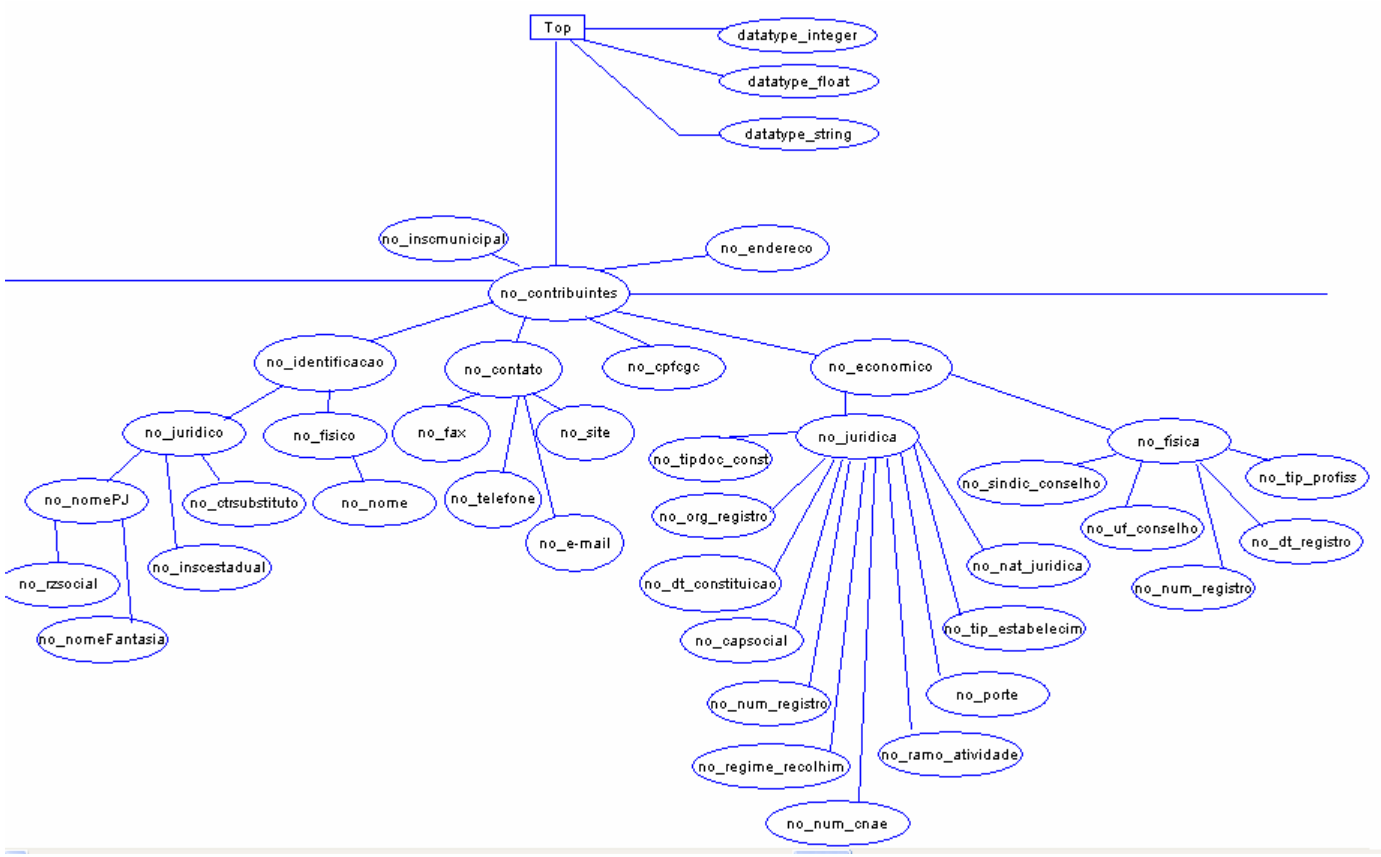
[WIRFS-BROCK, 1990] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. Designing Object-Oriented Software. Prentice-Hall, 1990.

[WOOLDRIDGE, 2002] WOOLDRIDGE, M. Na Introduction to Multi-Agent Systems, John Wiley & Sons Ltda. Chichester, Inglaterra, 2002.

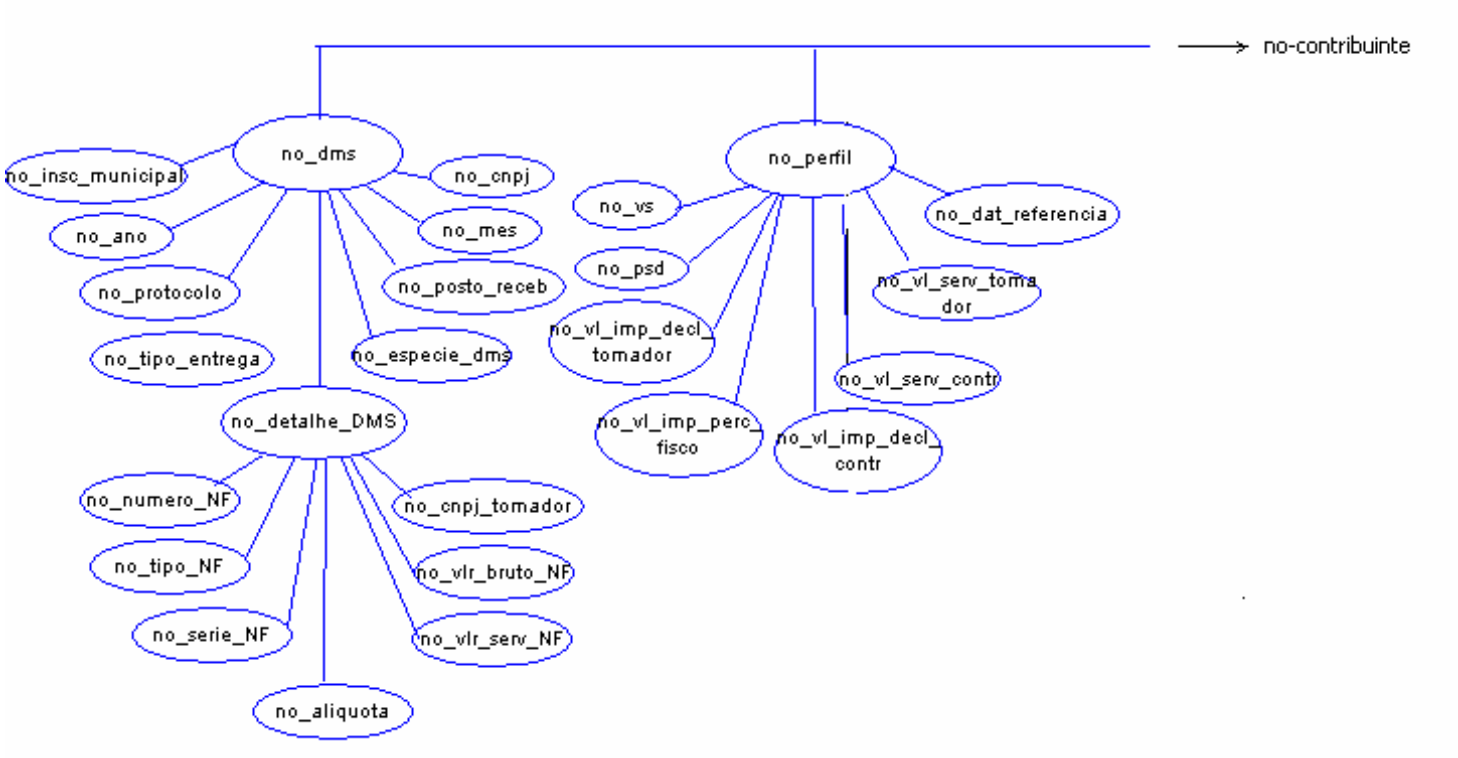
ANEXO I – DIAGRAMA DE CLASSES



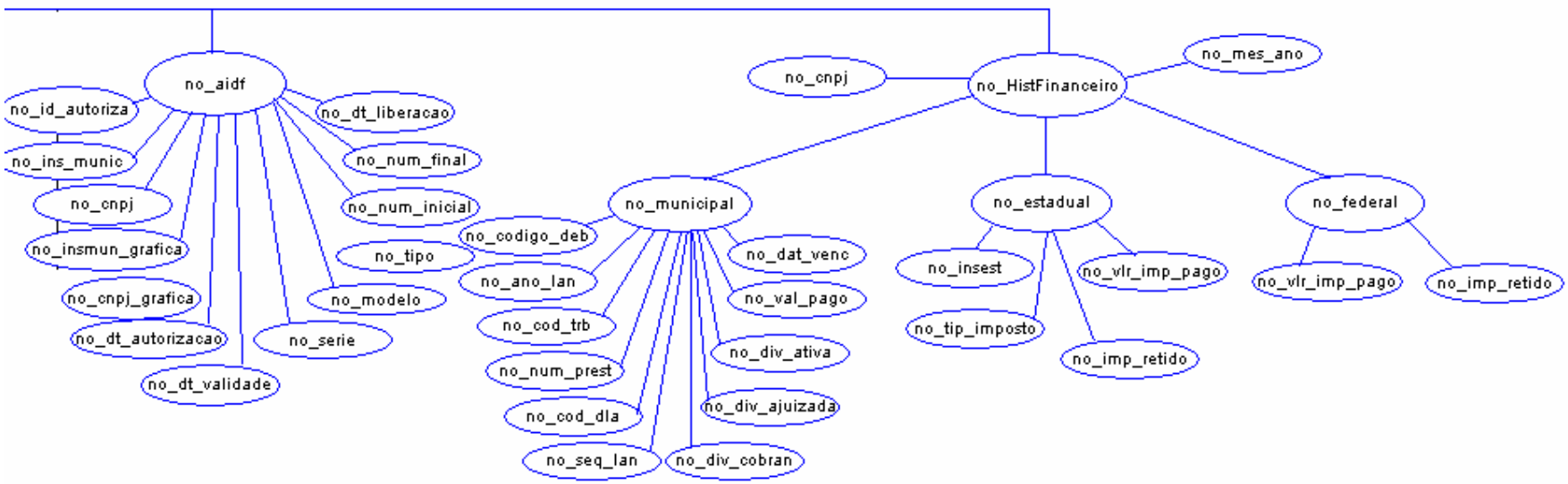
ANEXO II – ONTOLOGIA



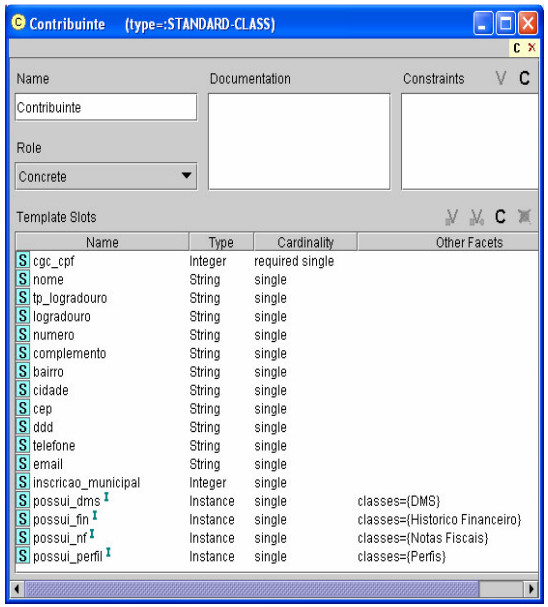
Parte da ontologia referente aos nó Contribuinte (sub-nós identificação, contato, informações econômicas).



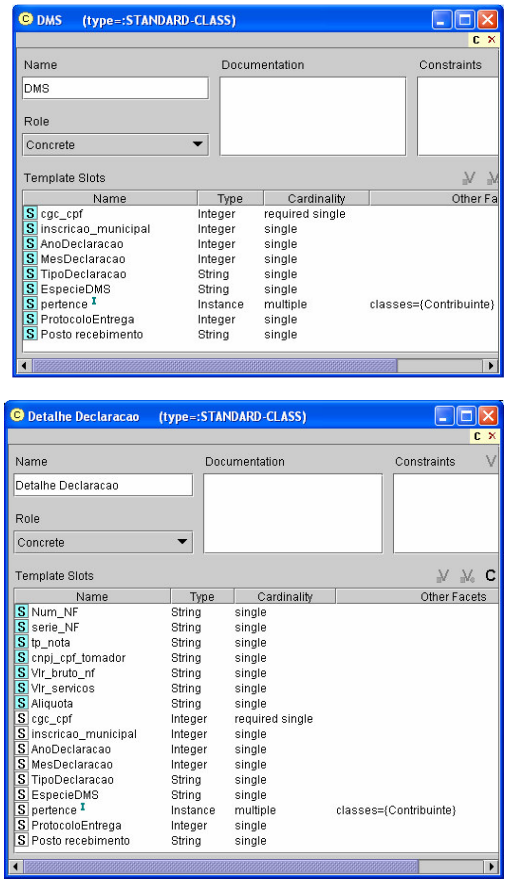
Parte da ontologia referente aos nós: DMS (e o sub-nó detalhe da DMS) e perfil.



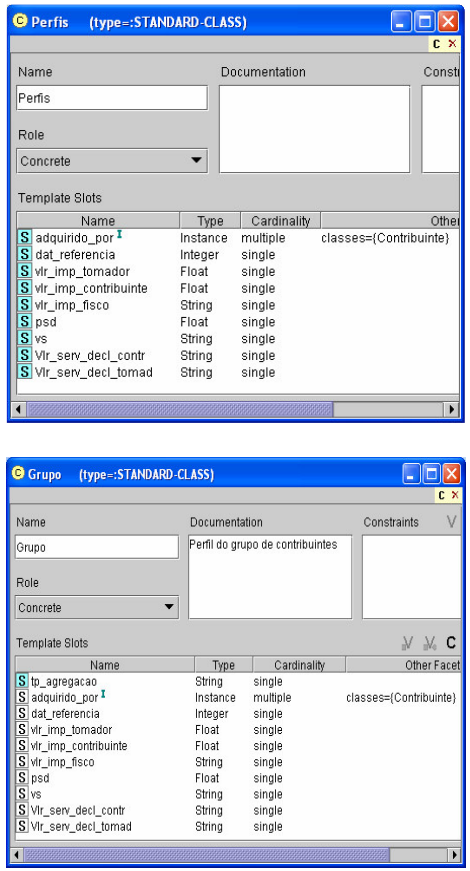
Parte da ontologia referente aos nós: AIDF e histórico financeiro (com os sub-nós de informações financeiras municipais, estadual e federal.



Contribuinte representado como classes ontológicas no Protégé



DMS e Detalhes da DMS como classes ontológicas no Protégé



Perfil individual e de grupo como classes ontológicas no Protégé

ANEXO III – CÓDIGO FONTE DOS AGENTES E REGRAS

```
package comportamentos;
import jade.core.behaviours.CyclicBehaviour;
import jess.*;
import jade.core.*;
import jade.lang.acl.ACLMessage;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.List;
import java.util.Hashtable;
import java.util.Iterator;
import utilities.*;
import messagens.*;


public class JessBehaviour extends CyclicBehaviour{

        //instancias para tratamento do conteúdo das mensagens
    TrataMensagens tratMsg = new TrataMensagens();
        TrataAID tratAID      = new TrataAID();


        /** JessSend é a classe interna que implementa
         * a interface Userfunction do jess para extender
         * linguagem com comandos definidos pelo usuário
         * */
        public class JessSend implements Userfunction {

                Agent ag;
                JessBehaviour jb;

                public JessSend(Agent a, JessBehaviour jessbehaviour){
                        ag = a;
                        jb = jessbehaviour;

                }

                /* nome do metodo pelo qual a função aparece no Jess (i.e. userFunction)*/
                public String getName() {

                        return("enviaMSG");

                }

    /**Este método só é chamado quando se invoca o comando enviaMSG no código Jess*/
    public Value call(ValueVector vv, Context context) throws JessException {
         // se no código jess for invocado o comando (enviaMSG ?m)
        if(vv.get(1).type() == RU.VARIABLE)
                vv =
context.getEngine().findFactByID(vv.get(1).factValue(context).getFactId());
        //senão se no código jess for invocado o comando (enviaMSG (assert (mensagemACL ...)))
        else if(vv.get(1).type() == RU.FUNCALL){
                Funcall fc = vv.get(1).funcallValue(context);
                vv =  fc.get(1).factValue(context);
        }
        ACLMessage msg = jb.fatosACLJess(context, vv);
        ag.send(msg);
         return Funcall.TRUE;
    }

  } // fim classe JessSend

                /* |||variáveis de classe||| */
        //instância da Classe rete (i.e. motor de inferência responsável pelo módulo de decisão
do agente)
        Rete  rete;
        //instância da classe agente para referenciar os agentes Auditor e Determinador
        Agent myAgent;
        //mantém o controle de passos a serem alcancados cada vez que o Jess for rodado
        int   numMaxPassos =  0;
        //conta o número de passos dado pelo jess na execução anterior
        int   totalPassos = -1;
```

```
    public JessBehaviour(Agent agent, String jessFile){
      myAgent = agent;
      tratAID.hashTabAID = new Hashtable<String, AID>();
      // cria o motor jess
      rete = new Rete();
      try {
                        // definição do template que trata o BD com Beans
            rete.eval(beansBDClassTemplate());
            //definição do template que trata as mensagens ACL
                        rete.eval(templateJessACL());
                    // definição do template templateAgente
                        rete.eval("(deftemplate templateAgente (slot nomeAgente))");
                        rete.addUserfunction(new JessSend(myAgent,this));
                        // inserção do fato (templateAgente (nomeAgente ...))
                        rete.eval("(deffacts templateAgente (templateAgente (nomeAgente " +
myAgent.getName() + ")))");
                        // chamada ao arquivo .clp (arquivo das regras)
                        FileReader fr = new FileReader(jessFile);
                        Jesp j = new Jesp(fr, rete);
                        j.parse(false);
                    } catch (JessException re){
                        System.out.println(re);
                    } catch (FileNotFoundException e) {
                System.out.println(e);
                    }
       }

     public JessBehaviour(Agent agent, String jessFile, int nMxPassos){
        this(agent,jessFile);
        numMaxPassos = nMxPassos;
     }


     /*Executa o comportamento do agente */
     public void action() {

        ACLMessage msg;
        // espera por mensagens
        if (totalPassos < numMaxPassos) {
                System.out.println(myAgent.getName()+ " esta bloqueado pra esperar
mensagens...");
                msg = myAgent.blockingReceive();
                // insere o fato no Jess
                assertFato(mgsAClJess(msg));
        } else {
                System.out.println(myAgent.getName()+ " verificando se existem mensagens...");
                msg = myAgent.receive();
                if (msg != null)
                        assertFato(mgsAClJess(msg));
        }
        // inicia a execução do Jess
        try {
                if (numMaxPassos > 0) {
                        totalPassos = rete.run(numMaxPassos);
                        System.out.println("Realizado(s) "+totalPassos+" passo(s)");
                }
                else
                        rete.run();

        } catch (JessException re) {
                re.printStackTrace(System.err);
        }
     }
     //verifica a consistência das mensagens
     private boolean vazio(String string) {

        return string == null || string.equals("");

     }

      /* insere um fato representando uma mensagem ACL no Jess*/
      private void assertFato(String fato) {

             try{
                  rete.eval(fato);
               }
```

```
                catch (JessException re) {
                      re.printStackTrace(System.err);
                }
          }
          /**Cria o template de comunicação no jess**/
          public String templateJessACL() {

                String cmd = "(deftemplate mensagemACL " +
                                        "(slot performativa) " +
                                        "(slot remetente) " +
                                        "(multislot destinatario) " +
                                        "(slot conteudo))";

                return cmd;
          }
          /*Cria a instancia da classe BeansBD para acesso às propriedades set e get
           * utilizadas para as consultas na base de dados dos Contribuites, Tomadores e NF*/
          public String beansBDClassTemplate() {

                String cmd = "(import servicos.*)" +
                                "(defclass myBeans BeansBD)"+
                                "(ppdeftemplate myBeans)";

                return cmd;
          }

          /*Mensagem enviada pelo remetente */
          public ACLMessage fatosACLJess(Context context, jess.ValueVector vv) throws
jess.JessException {

                int perf = ACLMessage.getInteger(vv.get(0).stringValue(context));
                ACLMessage msg = new ACLMessage(perf);

                System.out.println("******** Remetente ********* " + vv.get(1).toString());

                if (vv.get(1).stringValue(context) != "nil")

msg.setSender(tratAID.obtemAIDAgentes(vv.get(1).stringValue(context)));

                if (vv.get(2).toString() != "nil") {
                        List l = tratAID.obtemListaAgentes(context,
vv.get(2).listValue(context));
                        for (int i=0; i<l.size(); i++)
                                msg.addReceiver((AID)l.get(i));
                }
                if (vv.get(3).stringValue(context) != "nil") {
                        msg.setContent(tratMsg.semCote(vv.get(3).stringValue(context)));
                }
          return msg;
          }


  /** Manipula o template mensagemACL retornando o
   * comando da lingugem Jess*/
   public String mgsAClJess(ACLMessage msg){
          /* fat inicia a inserção das linhas dos comando da linguagem Jess a serem
codificados
           * no arquivo .clp*/
           String fat;

           if (msg == null)
                  return "";

           fat = "(assert (mensagemACL (performativa " +
ACLMessage.getPerformative(msg.getPerformative());

           if (msg.getSender() != null) {
                  fat = fat + ") (remetente " + msg.getSender().getName();
                  tratAID.adicionaAID(msg.getSender());
           }
           Iterator i = msg.getAllReceiver();
           if (i.hasNext()) {
                  fat = fat + ") (destinatario ";
                  while (i.hasNext()) {
                          AID aid = (AID)i.next();
                          tratAID.adicionaAID(aid);
                          fat = fat + aid.getName();
```

```
                }
            }
            if (msg.getContent() != null)
                    fat = fat + ") (conteudo " + tratMsg.cotaStr(msg.getContent());

            if (msg.getReplyByDate()!=null)
                    fat=fat+") (reply-by " + msg.getReplyByDate().getTime();

            fat=fat+")))";
            return fat;
    }

} // fim JessBehaviour


**************************************************************************

package auditor;
import comportamentos.JessBehaviour;
import jade.core.AID;
import jade.core.Agent;
import jade.lang.acl.ACLMessage;
import gui.AuditorGui;

public class AgentAuditor extends Agent{

        private String msg;
        protected AuditorGui myGui;
        /** Registro de conteúdo de linguagens
          * Registro de ontologias
          * inicialização de comportamentos*/
        public void setup(){
                //configura a gui
            myGui = new AuditorGui(this);
            addBehaviour(new JessBehaviour(this,myGui.selArquivoAud(),1));

            myGui.setVisible(true);

            //array de argumentos a serem passados na linha de comando
               Object[] args = getArguments();
               if (args != null && args.length>0){
                       msg = (String) args[0];
               }else{
                       System.out.println("Nenhuma mensagem foi especificada para o agente
"+this.getLocalName());
                       doDelete();
               }
               //getLocalName() retornará o nome do agente
               System.out.println("Agente:"+this.getLocalName());
               sendMessage(msg);
        }
        /**Envia as mensagens
         * @param
         * String msg mensagem a ser enviada para um agente*/
        public void sendMessage(String msg) {

            ACLMessage aclMessage = new ACLMessage(ACLMessage.REQUEST);
            aclMessage.addReceiver(new AID("Determinador",AID.ISLOCALNAME));
            aclMessage.setContent(msg);
            this.send(aclMessage);
          }

}

**************************************************************************

package determinador;
import comportamentos.*;
import jade.core.AID;
import jade.core.Agent;
import jade.lang.acl.ACLMessage;
import gui.DeterminadorGui;


public class AgentDeterminador extends Agent{

        private String msg;
```

```
        protected DeterminadorGui myGui;
        protected AgentDeterminador myAgent;

        public void setup(){

                //configura a gui
            myGui = new DeterminadorGui(this);
            addBehaviour(new Receptor(this));
            myGui.setVisible(true);
            //array de argumentos a serem passados na linha de comando
                Object[] args = getArguments();
                if (args != null && args.length>0){
                        msg = (String) args[0];
                }else{
                        System.out.println("Nenhuma mensagem foi especificada para o agente
"+this.getLocalName());
                        doDelete();
                }
                //getLocalName() retornará o nome do agente
                System.out.println("Agente:"+this.getLocalName());
                sendMessage(msg);

        }
        /**Envia as mensagens
         * @param
         * String msg mensagem a ser enviada para um agente*/
        public void sendMessage(String msg) {

            ACLMessage aclMessage = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
            aclMessage.addReceiver(new AID("Analisador",AID.ISLOCALNAME));
            aclMessage.setContent(msg);
            this.send(aclMessage);
          }

}

*************************************************************************
package gui;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import javax.swing.*;
import auditor.*;
import gui.AuditorGui;

public class AuditorGui extends JFrame implements ActionListener{
        private JFileChooser fc = new JFileChooser();
        private AgentAuditor myAgent;
        private JButton btnEnviar = new JButton("Enviar");
        private JButton btnAbrir = new JButton("Abrir");
    private JTextField tfMensagem = new JTextField(20);

        public AuditorGui (AgentAuditor agent){
                myAgent = agent;
                setTitle("SMA-Auditor - Agente " + myAgent.getName());
         JPanel base = new JPanel();
            //registra os Listeners
         btnEnviar.addActionListener(this);
         btnAbrir.addActionListener(this);
         base.add(btnAbrir);
         base.add(btnEnviar);
         base.add(tfMensagem);
         getContentPane().add(base);
            setSize(470, 100);

        }
        public void imprime(String msg){
                System.out.println(msg);
        }
        public String selArquivoAud(){
                String fileName = "";
                int returnVal = fc.showOpenDialog(btnAbrir);
         if (returnVal == JFileChooser.APPROVE_OPTION) {
             File file = fc.getSelectedFile();
             fileName = file.toString();

         }
```

```
      return fileName;
       }

       public void actionPerformed(ActionEvent e) {
           if(e.getSource() == btnEnviar){
               myAgent.sendMessage(tfMensagem.getText());
               imprime(tfMensagem.getText());
           }
           if(e.getSource() == btnAbrir){
                selArquivoAud();
           }
       }

}


*************************************************************************
package gui;
import java.awt.event.*;
import java.io.File;
import javax.swing.*;
import determinador.*;
import gui.DeterminadorGui;


public class DeterminadorGui extends JFrame implements ActionListener{

       private JFileChooser fc = new JFileChooser();
       private AgentDeterminador myAgent;
       private JButton btnEnviar = new JButton("Enviar");
       private JButton btnAbrir = new JButton("Abrir");
       private JTextField tfMensagem = new JTextField(20);

       public DeterminadorGui (AgentDeterminador agent){
              myAgent = agent;
              setTitle("SMA-Auditor - Agente " + myAgent.getName());
        JPanel base = new JPanel();
           //registra os Listeners
        btnEnviar.addActionListener(this);
        btnAbrir.addActionListener(this);
        base.add(btnAbrir);
        base.add(btnEnviar);
        base.add(tfMensagem);
        getContentPane().add(base);
           setSize(470, 100);

       }
       public void imprime(String msg){
              System.out.println(msg);
       }

       public String selArquivoDet(){
              String fileName = "";
              int returnVal = fc.showOpenDialog(btnAbrir);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fc.getSelectedFile();
            fileName = file.toString();
        }
        return fileName;
       }

       public void actionPerformed(ActionEvent e) {
           if(e.getSource() == btnEnviar){
               myAgent.sendMessage(tfMensagem.getText());
               imprime(tfMensagem.getText());
           }
           if(e.getSource() == btnAbrir){
               selArquivoDet();
           }
       }

}
```

```
*************************************************************************

package comportamentos;
import jade.core.Agent;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.core.behaviours.SimpleBehaviour;

public class Receptor extends SimpleBehaviour{

                public Receptor(Agent agent) {
                  super(agent);
                }

                public void action() {
                      MessageTemplate mt =
                      MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);


                        while (true) {
                      ACLMessage aclMessage = myAgent.receive(mt);
                      if (aclMessage!=null) {
                        System.out.println(myAgent.getLocalName()+": Recebendo mensagem:
\n"+aclMessage);
                      } else {
                        this.block();
                      }
                    }
                }
                //manda terminar (true) após a execução do método action()
                public boolean done() {
                  return false;
                }

}


*************************************************************************
package servicos;
import java.io.Serializable;
import java.beans.*;
import java.net.*;

import javax.swing.JOptionPane;

public class BeansBD implements Serializable{
        private PropertyChangeSupport pcs = new PropertyChangeSupport(this);
        private String coluna;
        private String tabela;
        private String dado;
        private String resConsulta;
        private String consulta;
        private int verDuplicidade;
        int registros;

        Busca busca = new Busca();

        public String getColuna() {
                return coluna;

        }
        public String getTabela() {
                return tabela;

        }

        public int getVerDuplicidade() {
                return registros;

        }

        public String getConsulta() {
                return consulta;

        }
        public String getDado() {
```

```
                return dado;
        }

       /** Ajusta a consulta pra recuperar colunas no banco de dados
        * na forma SELECT * FROM <<tab>> WHERE <<col>> = <<val>>, mais um
        * parametro de retorno.
        * @param tab
        * Tabela do banco a ser pesquisado
        * @param col
        * Coluna do banco a ser localizada
        * @param val
        * Valor do filtro da Cláusula WHERE para a tabela especificada
        * @param colRetorno
        * Coluna de retorno da consulta
        * */
        public void setConsulta(String tab,String col, String val, String colRetorno) {
                String temp = tabela;
                tabela = tab;
                coluna = col;
                dado = val;
                consulta = busca.obtemDado("select * from "+tab+" where "+col+" =
"+val+"",colRetorno);
                /* realiza a troca do valor da propriedadetemporario pelo novo valor setado e
                 * */
                pcs.firePropertyChange("tabela", temp, val);
        }
        public void setVerDuplicidade(String tab,String col, String val, String valGroupBy,
String colRetorno){
                 String temp = tabela;
                tabela = tab;
                coluna = col;
                dado = val;
                registros = busca.obtemDado("select * from "+tab+" where "+col+" = "+val+"
group by "+valGroupBy+"",colRetorno);
                /* realiza a troca do valor temporario da propriedade pelo novo valor setado
                 * */
                pcs.firePropertyChange("tabela", temp, val);
        }
        public void setColuna(String s) {
                String temp = coluna;
                coluna = s;
                pcs.firePropertyChange("coluna", temp, s);
        }
        public void setDado(String s) {
                String temp = dado;
                dado = s;
                pcs.firePropertyChange("dado", temp, s);
        }

        //espera que aconteca algum evento nas propriedades
        public void addPropertyChangeListener(PropertyChangeListener pcl) {
                pcs.addPropertyChangeListener(pcl);

        }
        public void removePropertyChangeListener(PropertyChangeListener pcl) {
                pcs.removePropertyChangeListener(pcl);

        }
       }
**********************************************************************

;##############################################################################################;
;       Módulo de decisão para o comportamento do agente Analisador                  ;
;##############################################################################################;

;**********************************************************************************************;
;                        Função de tratamento e consulta de dados                              ;
;**********************************************************************************************;

;afunção trataBeansBD realiza a consulta no bd acessando a classe BeansBD pela instância
;myBeans que é amarrada a uma variável ?bBD para prover referencia às
;propriedades beans da classe
(deffunction trataBeansBD (?tab ?colCons ?colFiltro ?colRes)
    (bind ?bBD (new BeansBD))
        (definstance myBeans ?bBD)
    (call ?bBD setConsulta ?tab ?colCons ?colFiltro ?colRes)
    (bind ?rr (call ?bBD getConsulta))
```

```
    (return ?rr)
    )

(deffunction verDupliNF(?tab ?colCons ?colFiltro ?colGroupBy ?colRes)
    (bind ?bVD (new BeansBD))
        (definstance myBeans ?bVD)
    (call ?bVD setConsulta ?tab ?colCons ?colFiltro ?colRes)
    (bind ?rr (call ?bVD getVerDuplicidade))
    (return ?rr)
    )

;(bind ?bBT (new BeansTexto))
;       (definstance myBeansTexto ?bBT)
(deffunction textoMSG ()
    (bind ?bBT (new BeansTexto))
        (definstance myBeansTexto ?bBT)
    (bind ?tx (call ?bBT getTxtMsg))
    (return ?tx)
    )

;*****************************************************************************;
;                        Regras de tratamento de mensagens                    ;
;*****************************************************************************;
;variáveis globais para armazenar o nome dos agentes
(defglobal ?*agRem* ="")
(defglobal ?*agDes* ="")

;quando uma mensagem do tipo 'REQUEST' é recebida de um agent ?s
;esta regra insere uma mensagem 'ACCEPT_PROPOSAL' para o mesmo agente
;que a enviou e depois  limpa a variável ?m na qual foi amarrado
;o padrão LHS (i.e. premissa) da regra
;enviaMSG é a userFunction definida na classe JessBehaviour

(defrule proposalMensagem
 ?m <- (mensagemACL (performativa REQUEST) (remetente ?s) (conteudo ?c) (destinatario ?r))
 =>
 (store AgRem ?s)
 (store AgDes ?r)
 (bind ?*agRem*(fetch AgRem))
 (bind ?*agDes*(fetch AgDes))
 (enviaMSG (assert (mensagemACL (performativa ACCEPT_PROPOSAL) (destinatario ?s) (conteudo ?c)
)))
 (assert (mensagemACL (performativa ACCEPT_PROPOSAL) (remetente ?r) (destinatario ?s)
(conteudo ?c) ))
 (retract ?m)
)

;caso uma mensagem seja inserida na memória de trabalho após ter sido
;enviada pelo proprio agente remetente, então a mensagem é enviada e
;em seguida é limpa através do comando retract
(defrule enviaMensagem
 (templateAgente (nomeAgente ?n))
 ?m <- (mensagemACL (remetente ?n))
 =>
 (enviaMSG ?m)
 (retract ?m)
)

;*****************************************************************************;
;                                    Regras de Decisão                        ;
;*****************************************************************************;
(defrule regra1
    (mensagemACL (remetente ?rmt))
    (test (neq ?rmt (trataBeansBD CONTRIBUINTE VALORNF ?rmt VALORNF)))
     =>
    (enviaMSG "Sonegação de imposto")
    (printout t "Sonegação de imposto" crlf)
    (printout t "O Contribuinte emitiu um valor menor do que o real" crlf)
)
(defrule regra2
    (mensagemACL (remetente ?rmt))
    (mensagemACL (conteudo ?cont))
    (test (neq ?cont (trataBeansBD CONTRIBUINTE NF ?cont  NF)))
    (test (eq ?cont (trataBeansBD TOMADOR NF ?cont  NF)))

     =>
       (enviaMSG "Sonegação de servicos prestados")
```

```
    (printout t "Sonegação de servicos prestados" crlf)
    (printout t "O Contribuinte não declarou a Nota Fiscal" crlf)
)

(defrule regra3
    (mensagemACL (remetente ?rmt))
    (test (neq "SIM" (trataBeansBD NOTAFISCAL AUTORIZADA 'SIM' AUTORIZADA)))
     =>
       (enviaMSG "Documento Fiscal não autorizado ou inexistente")
    (printout t "Documento Fiscal não autorizado ou inexistente" crlf)
    (printout t "O Contribuinte emitiu notas sem Autorização para Impressão de Documentos
Fiscais" crlf)
)

(defrule regra4
    (mensagemACL (remetente ?rmt))
    (mensagemACL (conteudo ?cont))
    (test (eq ?cont (trataBeansBD TOMADOR NF ?cont NF)))
    (test (> 1 verDupliNF TOMADOR NF ?cont NF))
     =>
       (enviaMSG "Emissão de Nota Fiscal paralela ou em duplicata")
    (printout t "Emissão de Nota Fiscal paralela ou em duplicata" crlf)
    (printout t "O Contribuinte emitiu notas em duplicidade" crlf)
)

(defrule regra5
    (mensagemACL (remetente ?rmt))
    (mensagemACL (conteudo ?cont))
    (test (eq ?cont (trataBeansBD CONTRIBUINTE NF ?cont  NF)))
    (test (neq ?cont (trataBeansBD TOMADOR NF ?cont  NF)))
     =>
       (enviaMSG "Emissão de Nota Fiscal paralela ou em duplicata")
    (printout t "Emissão de Nota Fiscal paralela ou em duplicata" crlf)
    (printout t "O Contribuinte emitiu notas em duplicidade" crlf)
)
(defrule regra6
    (mensagemACL (remetente ?rmt))
    (mensagemACL (conteudo ?cont))
    (test (neq ?cont (trataBeansBD CONTRIBUINTE DMS ?cont IRF)))
    (test (eq  ?cont (trataBeansBD CONTRIBUINTE DMS ?cont IRF)))
     =>
       (enviaMSG "Nota Fiscal recebida com diferença de retenção de imposto na fonte -IRF")
    (printout t "Nota Fiscal recebida com diferença de retenção de imposto na fonte -IRF "
crlf)
    (printout t "O Tomador informou valor do IRF menor do que o declarado pelo Contribuinte ou
inexistente" crlf)
)

;exibe os fatos
(watch facts)
;exibe a agenda (work memory)
(watch all)
(reset)
(run)

**************************************************************************

package messagens;
import jade.core.AID;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.List;
import jess.Context;
import jess.JessException;
import jess.ValueVector;

public class TrataAID {

          /* hashTabAID armazena temporariamente nomes dos agentes
           * como chave e valor como AID para mapeamento
           */
          public Hashtable<String, AID> hashTabAID;

          /*recupera o AID dos agentes a partir do seus nomes*/
          public AID obtemAIDAgentes(String nomAgente) {
            AID result;
            result = (AID)hashTabAID.get(nomAgente);
```

```
            if (result == null){
              result = new AID(nomAgente, AID.ISLOCALNAME);
            }
            return result;
          }

          public List obtemListaAgentes(Context cntx, ValueVector lst) {
            ArrayList<AID> l = new ArrayList<AID>();
            for(int i = 0; i < lst.size(); i++){
              try{
                l.add(obtemAIDAgentes(lst.get(i).stringValue(cntx)));
              }
              catch(JessException je) {}
            }
            return l;
          }



          /**
           * Adiciona o AID na tabela hash
           * Se ja existir o AID, o metodo sobrescreve
           * o AID existente
           */
          public void adicionaAID(AID aid) {
                 hashTabAID.put(aid.getName(),aid);
          }

}

*****************************************************************************


*****************************************************************************


*****************************************************************************


*****************************************************************************


*****************************************************************************
```