BRAIN ARCHITECTURE AND REASONING OF INTELLIGENT AGENTS IN MAS

Ondřej Kohut[1] , Michal Košinár[2], Ondřej Takács[3]

Katedra Informatiky, FEI, VŠB – Technical University Ostrava, 17. listopadu 15/2172, 708 33, Ostrava - Poruba, Czech Republic

[1]ondrej.kohut.fei@vsb.cz
[2]michal.kosinar.st@vsb.cz
[3]ondrej.takacs.st@vsb.cz

Abstrakt. Multi-agentní systém je systémem autonomních, inteligentních, avšak zdrojově omezených agentů. Jednotliví agenti musí být schopni činit vlastní rozhodnutí, a to na základě aktivit ostatních agentů a událostí v rámci systému jakož i jeho prostředí. K dosažení tohoto cíle musíme navrhnout „mozek“ agentů. V tomto článku prezentujeme základní návrh architektury mozku a popisujeme jeho funkci. Především budeme ilustrovat datové rozhraní mezi agentem-pozorovatelem a GIS systémem. V GIS jsou data uložena v „absolutní“ formě a agent je musí transformovat do tvaru relativního vzhledem k pozorovateli. Jako případová studie je použit model dopravního systému. Popisujeme agentovo vidění a jeho správu agentovým mozkem, a principy, jimiž se řídí usuzování agentů. Mozek je implementován v jazyce JAVA, rozhodování je realizováno programovacím jazykem Prolog s použitím pravidel Prologu. Usuzování agentů je ilustrováno algoritmy řešícími problematiku křižovatek. V článku se rovněž zabýváme využitím Transparentní intensionální logiky (TIL), přesněji její softwarové varianty TIL-Script, pro účely komunikace, získávání znalostí a rozhodování na základě dat obdržených z GIS.

Klíčová slova: MAS, mozek, GIS, rozhodování, Transparentní intensionální logika, TIL-Script.

Abstract. Multi-agent system is a system of autonomous, intelligent but resource-bounded agents. Particular agents have to be able to make decisions on their own, based on the activities of other agents and events within the system as well as its environment. To this end we design agents' "brain". In this paper we present a proposal of the brain architecture and describe its functioning. We concentrate on the data interface between an agent-observer and the GIS system. GIS data are stored in an "absolute" form, and an agent has to transform particular data to the form relative to the observer. As a case study, a traffic model is used. We describe the visibility of an agent and its management by an agent brain. The principles of agent's reasoning are explained and illustrated by an example of the algorithm for crossroad solving. The brain is implemented in JAVA language, decision making is realized by Prolog programming language using Prolog rules. Finally we deal with the integration of Transparent intesional logic (TIL), or more exactly of its software variant TIL-Script, into the agent's brain for purposes of communication, data mining and reasoning based on GIS data.

Keywords: MAS, brain, GIS, reasoning, Transparent intensional logic, TIL-Script.

1 Introduction

Multi-agent system is a system of autonomous, intelligent, but resource-bounded agents. Agents communicate with each other, but make decisions on their own. Reasoning of agents is driven by the activities of other agents and events within the system as well as its environment. The paper is organised as follow. In the next Section 2 we propose the architecture of agent's brain. Section 3 deals with Prolog brain unit and the respective algorithm of reasoning. Finally, in Section 4 we briefly introduce the TIL-Script language and its use in a multi-agent system.

2 The architecture of agent's brain

Agent's brain consists of several parts. First of all, each agent must have a memory, or rather an *Internal Knowledge Base* to store data and knowledge he is "born" with or learns during the life-cycle. It is available to all the other parts of the brain. Second, in order to implement some intelligence, agents must be equipped with one or more units capable of inferring other facts from those stored in the knowledge base. These units make use of different logic systems. The *Prolog Brain Unit*, which is being tested now, makes use of the first-order logic programming in Prolog. The *TIL Brain Unit*, which is under development, makes use of the higher-order expressive system of Transparent Intensional Logic, namely its computational variant, the TIL-Script language. The proposed brain architecture is open to other technologies like JADEX, JESS, etc. In addition to the units that are designed primarily as inference machines, the *Auxiliary Computational Module* provides the functions of mathematical computing (graph algorithms, numeric methods, etc.). The choice of a proper inference unit to make a decision is within the competence of the *Brain Interface*. The schema of the brain architecture is illustrated by fig. 1.

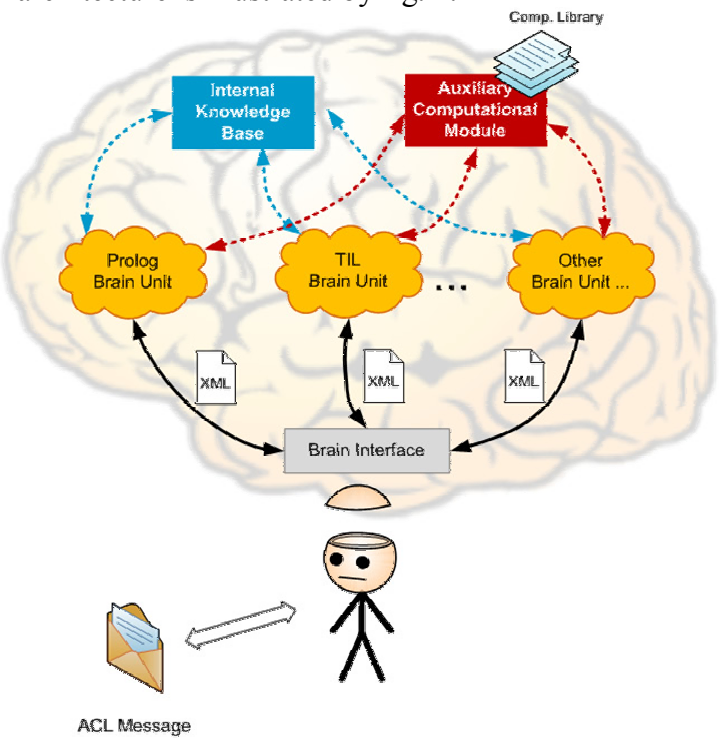


Fig. 1. Schema of the agent's brain.

A query sent to the brain is encoded in the XML format. The Brain Interface selects an appropriate inference unit to which the XML-query is assigned. The respective unit performs the necessary deductions, and the resulting answer is returned to a sender *via* the Brain Interface (encoded in the XML format again). The choice of proper inference unit is made in compliance with the pre-defined map:

$$f: P \rightarrow U,$$

where P corresponds to a set of problems and U is a set of inference units. The XML pack with a query contains also the unique name of problem. An example of this map is shown in Table 1.

Table 1. An example of the unit-choosing map.

Problem	Inference Unit
driving	Prolog BU
card playing	Prolog BU
communication	TIL BU

3 Prolog Brain Unit

In this chapter we discuss the Prolog Brain Unit that has been already implemented.
The Unit consists of the two main parts: the *PBU interface* and the *Prolog Engine*. The former communicates with the *Internal Knowledge Base*, and the latter infers particular consequences of the known facts using Prolog rules. The schema of the Prolog Brain Unit is shown in Fig. 2.

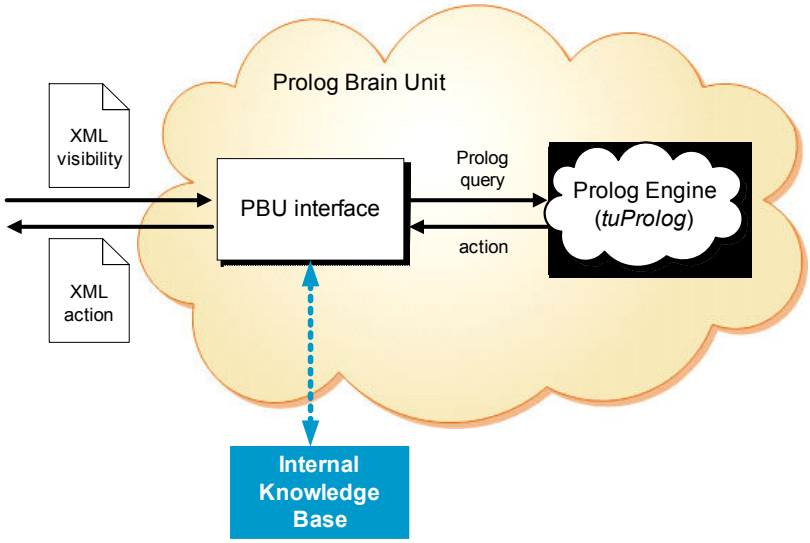


Fig. 2. Schema of the Prolog Brain Unit.

3.1 PBU interface – Data transformations

The aim of the PBU interface is to transform absolute data written in XML to the form relative to the observer, generating thus a query for the Prolog engine. As a case study we use the model of a traffic system. The infrastructure described in XML visibility contains roads and crossroads. Every road is divided into road-elements and each road-element consists of traffic lanes. These elements are defined absolutely, and they form a static infrastructure. The orientation of each road is determined by the beginning and ending crossroad. Road-elements are oriented by the order of junctions. The lane orientation is derived from the road orientation. Lanes in each road-element are numbered from left to right by integers 0,…,*n*. Intersections are described by the list of crossing roads and their azimuths. The infrastructure is inhabited by agents and objects.

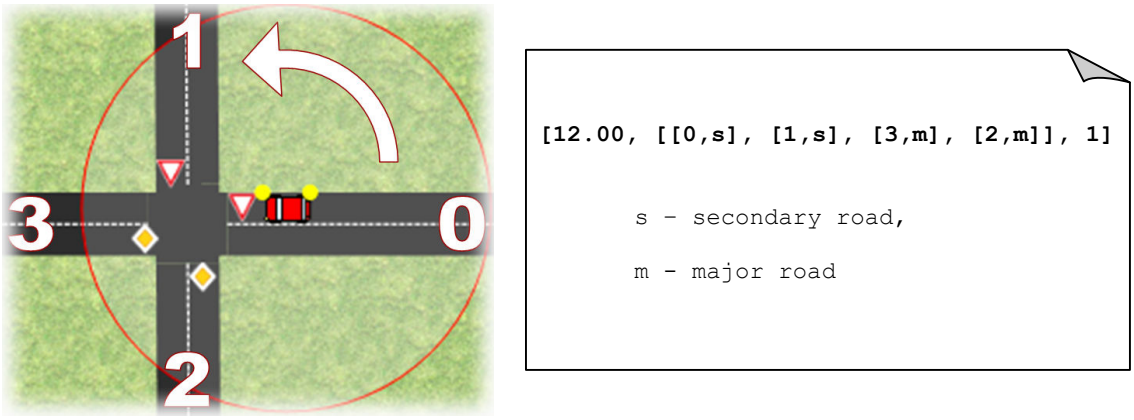
It is important for an agent to know whether the orientation of a lane is consistent with the direction he/she follows, which lanes are on the left or right side, the distances of infrastructure objects with respect to the agent, etc. In addition, an agent-observer has to follow the other agents, namely their position, (congruent or reverse) direction of moving, etc.

Data pre-processing consists of transforming absolute data to the values relative to an agent. It concerns above all the orientation (agent knows which direction is congruent with his/her own), renumbering of lanes and enumeration of the road-elements so that the agent may know which elements are ahead and behind him. Some of these transformations are described in [5].

Now we are going to describe the management of agents' behaviour when passing a crossroad. Intersection is defined in a Prolog-query as follows:

[*distance*, *road-element list*, *target road-element*],

where *distance* is the distance between the agent and the crossroad, *road-element list* contains the counter-clockwise list of crossing road-elements marked as major (*m*) or secondary (*s*) road. The list begins with the road-element in which the agent is situated followed by the elements ordered according to the azimuths of particular roads entering the intersection. *Target road-element* is the first road-element of a target road behind the crossroad while the target road corresponds to the next desired road according to an agent's itinerary. For example, the crossroad figured on Fig. 3 is described by [12.00, [[0,*s*],[1,*s*],[3,*m*],[2,*m*]],1].



(a) (b)

Fig. 3. An example of crossroad: a) real situation; b) corresponding description.

Detection of change direction

The only way the agents demonstrate their intentions is the use of trafficators. Thus it is necessary to determine whether the road behind a crossroad leads to the left, to the right or

straight ahead. The shape of the crossroad is described by the four-element list [CURRENT, RIGHT, STRAIGHT, LEFT]. When an agent is approaching the crossroad of an X shape, the solution is trivial. The list is filled by all the four road-elements in the counter-clockwise order. If there are only three roads crossing, one of the elements is left empty [].

If the deviation of two roads is similar[2] then the first and the second road are considered as the left and the right, respectively, regardless the real deviation value (see Fig. 4). Otherwise we must segment the direction circle into three areas representing left, right and straight directions (see Fig. 5a). The directions are determined according to the following rules (see Fig. 5b):

Both roads lay in the left region → the right one is set as *straight*
Both roads lay in the right region → the left one is set as *straight*
Both roads lay in the straight region → the left one is set as *left* and the right one as *right*

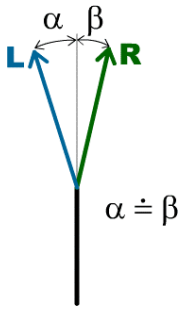
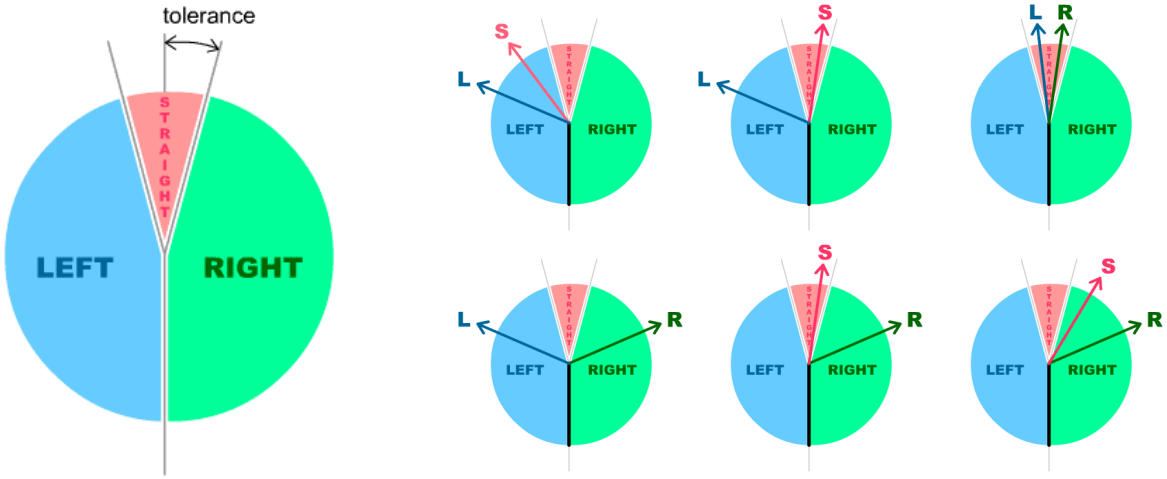


Fig. 4. Detection of change direction according to deviation similarity.



(a) (b)

Fig. 5. Detection of change direction according to the segment classification: a) segmentation; b) possible situations.

3.2 Prolog engine; reasoning in Prolog

In this section we are going to describe the algorithms for solving various traffic situations. We use *tuProlog*[3] open source implementation written in Java Language. Agents reasoning consists of (a) the selection of an appropriate algorithm that is best suited for the current

[2] Similarity is determined with respect to a given limit, e.g. 10 degrees.
3 http://www.alice.unibo.it:8080/tuProlog/

problem, (b) execution of the selected algorithm, and (c) output the result, i.e., the intended action, to the PBU interface.

Selection of an algorithm
Based on the visibility and a current situation, a suitable algorithm is chosen. Below we will adduce some examples of various algorithms solving the crossroad passing together with the conditions of their using.

The algorithm for driving on an empty road
This is the easiest traffic situation. It is used, when there are no obstacles on agent's traffic lane or on the lane right of him. The agent just aims at staying in the traffic lane to the right, and under the highest allowed speed. Thus there are only four actions an agent can perform: speed up, speed down, don't change speed and change lane right.
The agent would speed up, when the allowed speed is higher than his current speed.
The agent would speed down, when the allowed speed is lower than his current speed.
The agent would don't change speed, when the allowed speed is approximately the same as his current speed.
The agent would change lane right, if he is not in the lane most to the right.

The algorithm for driving on a road together with some other agents or obstacles
This algorithm is used when there are other agents or obstacles on agent's traffic lane or on the lane right to him. In these situations the agent tries to avoid collision with the objects on the road while maintaining the highest allowed speed and not restricting other agents on the lane right to him. The agent has to check the lane in front of him, and the lane to the right.
In this algorithm the conception *safety distance* is used. The safety distance is defined as the distance within which the agent can safely avoid collision without changing direction. It is determined by agent's current speed. The other limiting factors are not considered for the sake of simplicity. Current safety distance of an agent equals to double of his current speed. In case that there is another agent moving in an opposite direction, the safety distance equals to double of sum of both agents' speed. In case there is another agent behind the agent, the safety distance is double of the other agent speed.

The agent would speed up, when the allowed speed is higher than his current speed and the safety distance to another object in front of him is greater than the current distance to the object and the agent can change lane to the right.

The agent would speed down when the allowed speed is lower than his current speed or when there is another agent in front of him and the safety distance is lower than his current distance and he can't change lane to the right.

The agent would not change the speed, when the allowed speed is approximately the same as his current speed and he cannot change lane to the right and the safety distance to the other object before him is approximately the same as the current distance to the object.

The agent would change lane right if the safety distance to each object on the right lane is greater than the current distance to the object.
Two algorithms for solving crossroads are described. Both are independent of the shape of a crossroad and use the procedure which decides whether the two agents are *crossing their routes* or not. This can be influenced by these factors: relative positions of agents, and their intensions. Table 2 enumerates the cases when two agents cross their routes. In other cases they do not cross.

Table 2. Crossing routes of two agents. Rows are relative positions of agents, columns are intentions of the first agent, values are intentions of the second agent

	going left	**going ahead**	**going right**
to the left	ahead	ahead; right	X
against	ahead; right	left	left
To the right	left; ahead	left; ahead	ahead

Algorithm for solving crossroads without priorities

This algorithm is used when agent's distance to the nearest crossroad (without priorities) is lower then the safety distance. In such a situation the agent follows only two rules:

- give way to the agent to the right,
- when turning left, give way to other agents in the opposite direction.

The agent would slow down if he crosses the route of the other agent to his right. He will also slow down if he is turning left and crosses the route of the other agent in the opposite direction.

In the other situations the agent would keep going ahead.

Algorithm for solving crossroads with specified priorities

This algorithm is used when agent's distance to the nearest crossroads is lower then the safety distance and there are priorities specified. The decisions the agent has to made are based mainly on his position and intention, which determines three basic situations:

The situation when the agent is on the main road and his intention is to go ahead on the next main road is simple, because other agents must give way to him and he can go ahead through the intersection without checking other conditions.

The situation when the agent is on the main road and his intention is to proceed on a minor road is also easy to solve. The agent would only check if he crosses the route of some other agents on another main road. If he does, he must slow down, otherwise he can go ahead.

The last situation obtains when the agent is coming from a minor road. Here will the agent first give way to other agents on main roads, and then apply the algorithm for solving crossroads without priorities. More precisely, the agent will slow down, if he crosses routes of any agents on main roads. If he does not, he applies the algorithm for solving crossroads without specified priorities described above. If he crosses the route of some of these agents, he must slow down.

4 Introduction to TIL-Script

Agents in a multi-agent system are autonomous, but have to communicate with each other in order to achieve their goals. Thus communication is a key topic in the area of MAS. Current FIPA standard languages designed for agents' communication are based on the first-order logic enriched by higher-order construals whenever the first-order framework is too narrow to meet the demands of a smooth communication. However, these extensions are well-defined syntactically while their semantics is often rather sketchy, which may lead to inconsistencies.

We propose the TIL-Script language that is based on Transparent intensional logic (TIL). TIL is an expressive logical system primarily designed for the logical analysis of natural languages. Due to its procedural higher-order semantics, TIL-Script is well suited for the use as a content language of agent messages. Til-Script can also serve as a general semantic framework for the known formal languages. Moreover, it is a powerful declarative programming language that can easily be interconnected with the local as well as external knowledge bases of the system.

4.1 Transparent Intensional Logic

Transparent Intensional Logic (TIL) is a logical system founded by Prof. Pavel Tichý. It is a higher-order system primarily designed for the logical analysis of natural language. As an expressive semantic tool it has a great potential for using in artificial intelligence and in general whenever and wherever the humans need to communicate with the computers. More on the role of logic in artificial intelligence see [7].

Due to its rich and transparent procedural semantics, all the semantically salient features are explicitly present in the language of TIL constructions. It includes explicit intensionalisation and temporalisation and hyper-intensional level of algorithmically structured procedures (known as TIL constructions), which of a particular importance in the logic of attitudes, epistemic logic and the area of knowledge representation and acquisition.

In what follows we are going to briefly introduce the main notions of TIL. For details, see [1,2].

4.2 Basic Philosophy

The key notion of TIL is that of a *construction.* It is an algorithmically structured procedure, or instruction on how to arrive at a less structured entity. Constructions are assigned to expressions as their meanings. There is an analogy between the notion of construction and that of a formula of formal calculi. What is encoded by a particular formula (or a λ-term of the TIL language) is just a construction of the model of the formula. However, whereas the formulas of a formal language are mere sequences of symbols that have to be *interpreted* in order to equip them with meaning, TIL constructions are just those meaning. Particular linguistic terms serve only to their encoding.

4.3 Types of order 1

From the formal point of view, TIL is a partial, hyper-intensional *typed* λ-calculus. Thus all the entities (including constructions) receive a *type*. Types are collections of member objects. For a type '*a*' we call its members '*a*-objects'.

Types in TIL arise from a type *base*. The base is a (finite) collection of non-empty sets— *atomic types* of order 1. For the purposes of natural language analysis, *epistemic base* is used. Its members are specified in Table 3.

Notation	Description
o	Truth values: True and False.
ι	Individuals: simple objects — the 'lowest-level bearers of properties'.
τ	Time points. In TIL these are modelled as real numbers. This type is just the set of real numbers.
ω	Possible worlds. Collection of all the logically possible states of the world.

Table 3. Epistemic type base.

Molecular types are defined as functional closures of atomic types. If $a, b_1,...,b_n$ are types of order 1, the collection of all (including partial) functions/mappings from $b_1, ...,b_n$ to a is a *type of order 1*: denoted $(a\ b_1, ...,b_n)$. Next we are going to define types of order n that include constructions. But first, constructions have to be defined.

4.4 Constructions

Constructions are the elementary building stones of TIL. Depending on a valuation v, any construction v-constructs an object of some type, or is v-improper (fails to v-construct anything). TIL is a logic of partial functions.
There are four kinds of constructions:

Trivialization – ${}^0a$
is an elementary construction constructing the object a without the mediation of any other construction.
Variable – x
is a construction ("x" is just a name) that constructs an entity dependently on valuation— it v-constructs.
Composition – $[F\ C_1...C_n]$
is an instruction of the functional application: If F v-constructs a function f of type $(a\ b_1...b_n)$ and each C_i v-constructs c_i of type b_i, the Composition v-constructs the value of f at $<c_1...c_n>$, if any; otherwise the Composition is v-improper.
Closure – $[\lambda x_1...x_n\ C]$
If variables x_i range over b_i and C v-constructs an object of type a, the Closure v-constructs the following function f: let v' be a valuation that associates x_i with B_i and is identical v otherwise. Then f is undefined on $B_1,...,B_n$ if C is v' improper, otherwise the value of f on $B_1,...,B_n$ is what is v'-constructed by C.

4.5 Higher-order types

Each construction is of some order. The order of a construction is the highest order of the types involved in the construction. Basic type of a higher order i ($i \geq 2$) is the type $*_i$ — the collection of all constructions of order i. Molecular types of order i are functional closures of the types of order i, see above.

4.6 Multiagent Systems and Communication

Technologies based on agents are relatively new and very promising. A big amount of their applications can be found in artificial intelligence and large computer systems. A roadmap of this approach is presented in [6]. In this paper we will focus on the communication in MAS and particularly on content languages.

Basic standards for MAS are given by FIPA[4] (The Foundation for Intelligent Physical Agents, see [3,4]). According to it the basic unit of communication is a *message*. It can take an arbitrary form but it is supposed to have a structure containing several attributes. *Content* of the message is one of these attributes.

From the point of view of communication, the most important attributes are:

Performative
denotes a type of the message – its communicative act. Basic performatives are:

- Query
- Inform
- Request

Content
carries the semantic of the message. It can be encoded in any suitable language.

Ontology
is a vocabulary of domain specific terms. These (and only these) terms can be used in the content of the message.

4.7 FIPA SL

One of the objectives of this paper is to propose a new content language for multiagent systems. First we will briefly discuss the existing standard – FIPA SL.

FIPA SL (Semantic Language) is the only FIPA candidate content language marked as '*standard'*. It is based on the language of first order logic (FOL), but it extends its capabilities. One of advantages of this approach is that FOL is a well-known logic and it is well elaborated. But there are disadvantages too. First, FOL is a mathematical logic. Its development was motivated mainly by mathematics and FOL is really good for describing algebraic structures. But this is usually not the way the agents communicate a message.

For utilization in multiagent systems FOL needed to be extended. Formulas of FOL express only assertions. But also queries and requests are valid messages. So SL defines so-called *identifying expressions*. Moreover, SL is capable of specifying propositional attitudes of agents to other assertions like "*John believes that it is raining.*" However, the *content* of the embedded-believed clause is taken only as a *syntactic* object.

Syntactically is the SL well-defined. But there is no proper specification of semantics; one can only consult the section "*Notes on FIPA SL Semantics*" which is (as it says) just notes. The standard counts upon well-known semantics of FOL, but because of numerous extensions it is not applicable.

This lack of semantics can have unpleasant consequences. Two agents relying completely on the standard can understand the same message in a different way, which may yield fatal misunderstandings and inconsistencies. This is in conflict with the name – "*Semantic* Language".

[4] http://fipa.org/

4.8 The TIL-Script Language

TIL is well-suited for the utilization as a content language in multi-agent systems. Its main advantages are:

Semantic nature

TIL constructions are objects *sui generic.* Thus we do not have to encounter the problems of syntacticism like those of translation and inconsistencies stemming from the need od 'disquoting'.

High expressibility

The expressive power of TIL is really high. TIL is capable of analyzing all the semantic features of natural languages.

Its purpose

Unlike mathematical logics, TIL has been intended to be a tool for logical analysis of communication. Primarily it was designed for natural languages, but this gives it the great potential even in other areas.

But TIL, due to its notation, is not plausible as a computerised content language. That is why we are going to define a computational variant of TIL, namely the TIL-Script language. The reasons are:

1. The notation of the language of constructions is not fully standardized.
2. It is not possible to encode the language of constructions using only ASCII characters. It contains Greek letters, superscripts and subscripts.
3. TIL does not specify an interface to ontologies. Any content language must be able to use concepts of a specific domain. These concepts are defined in ontologies.
4. We need one standardized type base. The epistemic type base is not the best for multi-agent systems, because it lacks some very common types like integers, strings, lists and tuples/sequences. True, lists and tuples can be defined as molecular, functional types, but for the need of convenience we define them as standard basic types.

Type Base

The type base of TIL-Script is an extended epistemic type base. To make it more convenient we add some types common in informatics (see table 4). The type of actions is added too.

TIL-Script Type	TIL Equivalent	Description
`o, bool`	o	Type of truth values.
`i, indiv`	ι	Type of individuals.
`t, date`	τ	Type of time points.
`w, world`	o	Type of possible world states.
`n, nat`		Natural number type.
`t, real`	τ	Real number type.

Table 4. TIL-Script Type Base.

Molecular Types

Like in TIL, molecular TIL-Script types are collections of functions on defined types. More than that, TIL-Script has also sequence types. Sequence type is a collection of (finite) sequences of particular type. The following Table 5 shows the notation for various molecular types of TIL-Script.

TIL-Script Type	TIL Equivalent	Description
`(oi)`	*(οι)*	Type of characteristic functions from individuals to truth values.
`(oti)`	*(οτι)*	Type of functions from time points and individuals to truth values.
`i@tw`	$\iota_{\tau\omega}$	Type of individuals in intension (individual offices).
`List(i|i)`		Type of list of individuals.
`*1`	$*_1$	Type of constructions of order 1.

Table 5. TIL-Script Composite Types.

Constructions in TIL-Script

There are four standard kinds of constructions in TIL. All of them have its equivalent in the TIL-Script language.

Constructions in TIL-Script are written in nearly the same way as in the standard TIL notation. The following Table 6 shows the conversion between TIL and TIL-Script.

Description	TIL Notation	TIL-Script Syntax
Brackets	*[,]*	[,]
Trivialization	^0C	'C
Variable	*x*	x or x:[Type]
Abstraction	$\lambda x_1 \ldots x_n\ C$	\x1 \x2
Application on a world state and a time	C_{wt}	C@w,t

Table 6. TIL-Script Composite Types.

Examples

Now we are going to present some examples of sentences of natural language analyzed in TIL and its transcription using the TIL-Script language.

The successor function.

TIL analysis: $\lambda x[^0+ x\ ^0 1]$

TIL types: $x/*_1 \rightarrow \tau$; $+/(\tau\tau\tau)$; $1/\tau$; *CR*, *VaclavKlaus*/ι.

TIL-Script analysis transcription: `\x:Nat ['+ x '1]`

The president of the Czech Republic is Vaclav Klaus.

TIL analysis: $\lambda w \lambda t\ [^0= \lambda w \lambda t\ [^0 President\_of_{wt}\ ^0 CR]_{wt}\ ^0 VaclavKlaus];$

TIL types: $w/*_1 \rightarrow \omega$; $t/*_1 \rightarrow \tau$; $=/(o\iota\iota)$; *President_of* $/(\iota\iota)_{\tau\omega}$; *CR*, *VaclavKlaus*/ι.

TIL-Script transcription:

```
\w \t ['= [\w \t ['President@w,t 'CR]]@w,t 'VaclavKlaus].
```

4.9 Knowledge and ontologies for TIL-Script

Any content language is tightly related to ontologies. All concepts used or mentioned by a content language must be defined in an ontology. And vice versa, the content language must be able to use any concept from the ontology.

FIPA definition of ontology is rather vague. It just says that ontology provides a vocabulary of domain specific concepts and relations between them. This leads to diversity in implementations. Usually, ontotology takes a frame-like structure, which is well suitable for the FIPA SL language supported by the developer frameworks like Jade.[5]

To specify ontology for TIL-Script, we use either a knowledge base to store the TIL-Script description of particular entity types and names, or the built-in definition of entities, as illustrated by an example below (Table 7). The latest trend is to use the well-proven technologies of semantic web. In particular, the OWL language for defining ontologies is well-standardised and defined. However, the OWL support by the implementation tools for multi-agent systems is still a work-in-progress. We are ready to support knowledge extraction from OWL to TIL-Script, which is the subject of the ongoing research.

TIL Notation	TIL-Script Syntax
Charles, *Peter*/ι;	`Charles/Indiv; Peter/i;`
Function/(ττ);	`Function/(Real Real);`
Property /(οι)$_{τω}$;	`Property/(o i)@tw;`
Know/(οι*$_n$)$_{τω}$;	`Know/(o i *)@tw;`
C/*$_n$; *Improper*/(ο*$_n$);	`C/*; Improper/(o *);`

Table 7. Declaration of entities in TIL and TIL-Script.

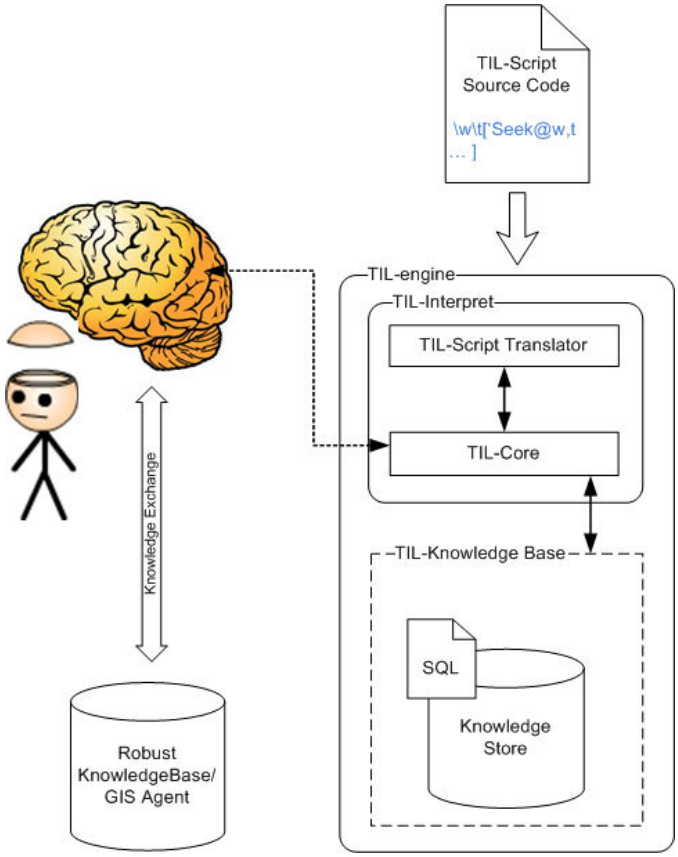


Fig. 6. The scheme of the TIL-Script usage.

[5] http://jade.cselt.it/

As illustrated by Figure 6 above, knowledge exchange between the TIL-Script Engine and the inner knowledge base of the agent's brain (which is likely to form agent's memory) is realized by a module within the TIL-Script Engine; to be more specific it is the TIL-Core unit that deals with knowledge exchange between the agent's memory (e.g. history of agent's visibility, messaging...), the external knowledge base (OWL ontologies transformation) and the TIL-Script format.

In the standard ontologies concepts are conceived as classes of so-called individuals. However, we have to be aware of the fact, that
(a) classes of individuals are not concepts; in TIL *concepts* are *(closed) constructions* of such classes, and
(b) the so-called 'individuals' of particular ontologies are objects of any (atomic as well as molecular) TIL-Script type of order 1; thus we must not confuse 'ontology individuals' with the TIL-Script elements of type Individual / i. For TIL-Script that means that any ontology concept (class), which members are of type α, is an object of type (oα) – a set of α -objects. Ontology individuals (members of classes) are directly α -objects.

The interface of TIL-Script to the knowledge base or an ontology is realised *via* TIL Trivialisation. You may trivialize any concept or individual defined in the agent's memory or the ontology in use. However, it is not possible to trivialize an unknown object, i.e., the object not defined by the ontology in use. Moreover, for the use in TIL-Script language, all the objects as well as ontology classes have to be equipped with TIL-Script types.

4.10 Knowledge acquisition from GIS

Geographic information systems (GIS) are generally used for gathering, analysis and visualization of information on the space aspects of real-world objects. The main advantage of GIS is the ability to relate different kind of information obtained from different sources of a spatial context. This ability enables agents to act in the real-world environment and make decisions based on its state. Agents usually dwell in the virtual world of a computer memory and they are not aware of their position, or of the objects and other agents around. GIS ontology enables agents to receive, manipulate and exchange spatial information.

Ontologies were developed to facilitate knowledge representation, sharing and exchanging. Spatial agents make use of geographic information as the input source for the decision-making mechanism. Situated agents are context-aware. They are aware of their position in space, actual speed, surrounding objects and relationships between them.

Agent actions are usually modelled by defining their behaviours. Each agent can perform the limited amount of predefined actions which can be combined into a more complex behaviors. Creating ontology is then divided into two steps.

1. Identification and definition of agent behaviors.
2. Creating behavior-specific ontology.

Agent can perceive their environment by sensors or geographic database and transform it into the symbolic representation of ontology. The basic idea of knowledge exchange has been illustrated by Figure 6 above.

4.11 Example

By way of conclusion we now present a simple scenario of communication of two agents in a situated multi-agent system using the TIL-Script language.

Scenario
The situation: there is lots of parking places and two agents, a driver and a dispatcher:

- **Driver** – an agent driving a car, who wants to park at the suitable parking place ('suitable for agent Driver' meaning the parking places in a close distance)
- **Dispatcher** – a dispatcher of the car parking.

The sketch of their communication:

1. **Driver:** I wish you search a suitable parking place for me.
   2.1. **Dispatcher:** Ok, I've found some parking places suitable for you.

Driver: Thank you.

   2.2. **Dispatcher:** Sorry, but I don't know what you mean by "suitable parking place", please refine.

Driver: The most suitable parking place for me is the one which is nearest to me and is cheaper than 2$.

Ontology
In order to analyze the communication using TIL-Script we need an ontology of the used concepts. The ontology is presented in Table 8.

Table 8. The ontology needed for this example.

Class	TIL-Script Type	Description
TheDriver	`i`	The Driver agent.
TheDispatcher	`i`	The Dispatcher agent.
TheParking	`i`	The parking place the driver wants the dispatcher to look for.
SuitableParkingFor	`(List i)@tw`	A list of suitable parking places (concrete individuals obtained from GIS) for agent *TheDriver*
Refine	`(o *1)`	A request to refine what the used construction means
Seek	`(o i List)@tw`	An action of seeking taking as arguments who (agent *Driver*) and what (list of suitable parking places)
Parking	`(oi)@tw`	The property of individual to be a parking place.
Price	`(ti)@tw`	The price of the concrete Parking place as an individual
TheLess	`(t(ot))`	Analytical function which returns the least number from the set of numbers.
TheNearest_to	`(oii)@t,w`	The function which returns a proposition whether the parking place is the nearest to *TheDriver* or not.

Communication

Now we can reconstruct the communication between the driver and the dispatcher precisely.

Example of a human agent requesting a suitable parking space and being asked by the dispatcher to specify the meaning of 'suitable parking space':

A human agent *TheDriver* asking for a suitable parking can simply formulate a message to the traffic dispatcher like this:

- "I am looking for a suitable parking place".

The content of the 'request-type' (FIPA-compliant) message in the TIL-Script is as follows:

- `\w\t['Seek@w,t 'TheDriver \w\t['Suitable_parking_for@w,t TheDriver]].`

The content of the reply message from the dispatcher can be, e.g., as follows:

- `\w\t[['Suitable_parking_for@w,t 'a] = List(p1,p2,p3)],`

which is translated into "The list of suitable parkings for *a = p1,p2,p3*", where *p1,p2,p3* are particular locations (currently GIS coordinates).

Or, the agent *(The Dispatcher)* may not know what 'suitable parking for *a*' means. In such a case the content of the reply message is a request for refining:

- `\w\t['Refine '[\w\t['Suitable_parking_for@w,t 'TheDriver]]]`

translated into "Refine the *concept* of Suitable parking for *TheDriver*". The agent *Driver* can then specify the parameters of the desired parking.

- `\w\t\x:Indiv['And ['Parking@w,t x] ['And ['< ['TheLess ['Price@w,t x]] '2] ['Nearest_to@w,t TheDriver x]]]`

5 Conclusion

The architecture of agent's brain is still being developed; in this paper we presented its current state. The Prolog brain unit has been described in details including the explanation of algorithms for decision making. The architecture and Prolog brain unit algorithms were tested both on simulated data and on the real GIS data. This architecture can be used as a base for creating agent framework that works with GIS data and makes use of the facilities provided by Prolog as well as the power of TIL.

The existing standards for communication in multi-agent systems are syntactically rather than semantically defined. This can slow down the progress in future research. As an alternative we propose the TIL-Script language which is based on the well-elaborated Transparent intensional logic. Thus TIL-Script is a semantically based language suitable for multi-agent systems.

The high expressive power of TIL-Script makes it an appropriate tool to adopt other logics and languages into its semantic framework as well, so that TIL-Script can be used as a general specification language. The TIL-Script semantic facilities make it possible to design the communication between humans and computer agents smooth and natural.

The TIL-Script language is being implemented and tested in multi-agent systems using the Python language and the framework Jadex[6].

[6] http://vsis-www.informatik.uni-hamburg.de/projects/jadex/

Acknowledgements. This research has been supported by the program "Information Society" of the Czech Academy of Sciences, project No. 1ET101940420 "Logic and Artificial Intelligence for multi-agent systems"

References

1. Duží, M., Jespersen, B., Müller, J.: Epistemic Closure and Inferable Knowledge. In *the Logica Yearbook 2004*. Ed. Libor Běhounek, Marta Bílková, Praha:Filosofia, 2005, Vol. 2004, 124-140, Filosofický ústav AV ČR, Praha, ISBN 80-7007-208-3.
2. Duží, M., Materna, P.: *Constructions,* <http://www.phil.muni.cz/fil/logika/til/>.
3. *FIPA: FIPA SL Content Language Specification* [online]. c2002 [cit. 2007-11-11]. Available from WWW: <http://www.fipa.org/specs/fipa00008/>.
4. *FIPA: FIPA Abstract Architecture Specification* [online]. c2002 [cit. 2007-11-11]. Available from WWW: <http://www.fipa.org/specs/fipa00008/>.
5. Kohut, O.: Brain for agents in multi-agent systems, In *WOFEX 2007*, Fakulty of elektrical engineering and computer science, VŠB – Technical University of Ostrava, 2007, p. 291-296
6. Luck, M., McBurney, P., Shehory, O., Willmott, S.: *Agent Technology: Computing as Interaction. A Roadmap for Agent Based Computing*. University of Southampton on behalf of AgentLink III, 2005.
7. Thomason, R.: *Logic and Artificial Intelligence*[online]. The Stanford Encyclopedia of Philosophy, Available from WWW: <http://plato.stanford.edu/archives/sum2005/entries/logic-ai/>.
8. TIL-Script Home Page [online]. c2005 [cit. 2007-11-11]. Available from WWW: <http://www.cs.vsb.cz/til/>.