JAMOCHAAGENT
A Rule-based Programmable Agent

Uta Christoph, Karl-Heinz Krempels and Alexander Wilden
Chair of Computer Science 4, RWTH Aachen University, Aachen, Germany

Keywords: Agent programming, Speech-act consumption, Speech-act production, Rule-based system.

Abstract: Agent programming in compliance with standardized interaction mechanisms is a challenging task in agent based application development. This results from standard-compliant agent development frameworks that provide support for the interaction mechanisms on one hand and non-standardized high level programming inference machines for knowledge processing on the other hand.
We propose an approach to automated speech act consumption and production within MAS by mapping agent behaviour onto a rule-based system. We show how speech acts have to be transformed, resolved and retransformed into the appropriate reply act to comply with predefined/given interaction protocols. This shift of behaviour definition onto a rule-based system allows for a convenient adaptation of agent behavior at runtime without the necessity of time consuming recompilation.

1 INTRODUCTION

Communication in third generation MAS (multi-agent systems) is mostly based on interaction protocols and speech acts recommended by FIPA (Foundation for Intelligent Physical Agents) and existing semantic languages (Botello et al., 2001), such as KIF, Prolog, FIPA Semantic Language, ebXML etc. Within the content of the messages constituting an interaction protocol, ontologies are used for the specification of domain and task oriented terms and relations. The ontology design process is already well supported by different development and maintenance tools, like e.g. Protégé, which also afford a convenient export of ontologies for direct use in existing agents, agent systems and their development process.

The appropriate continuative step is the automated semantic evaluation of speech acts and the used interaction protocols by agents themselves. This presumes the ability of an agent to automatically process the content of a message (speech act) represented by a content language and an ontology. Furthermore, it presumes that an agent is able to act according to its internal state (which might be influenced by the evaluation of a message content), e.g to react (properly) to a speech act. In this paper an approach for the automated evaluation of agent speech acts with the help of a rule based inference machine is introduced.

The remainder of this paper is outlined as follows: Section 2 gives an overview of FIPA-compliant intelligent agent programming, existing drawbacks in present approaches, and states requirements that must be met to resolve such drawbacks. This is followed by the introduction of our approach in section 4 and a detailed description of the *JamochaAgent* architecture in section 5. Section 6 specifies a formalization of the transformations between speech acts and rule-based systems and depicts some chosen speech acts in detail. In section 7 an example illustrates the transformations. We finish this paper with a summary of our results and some conclusions for future work in section 8.

2 STATE OF THE ART

The implementation of intelligent agents supporting FIPA-compliant interaction protocols(FIPA, 2002b) has to consider complex speech acts, the used content language, as well as embedded ontological concepts. In most FIPA-compliant agent systems Java is used as programming language and FIPA-SL (semantic language) (FIPA, 2002e) as the content language. Ontologies are usually designed and maintained with ontology editing tools, e.g. Protégé, and exported to a target language that can be handled by the agent system, e.g. Java code generated with Protégé's Bean Generator plugin. Yet the challenge

Christoph U., Krempels K. and Wilden A. (2009).
JAMOCHAAGENT - A Rule-based Programmable Agent.
In *Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 447-454
DOI: 10.5220/0001558204470454
Copyright © SciTePress

for developers is to manually create message content for agent communication in Java and in the process respecting the structure of the given ontology as well as the FIPA-SL grammar rules. Thus the construction of a simple query becomes a very extensive and time-consuming job, due to the need to define the used terms at all FIPA-SL grammar levels, to instantiate them, and to initialize them with the according sub-terms manually. (To get an idea of this task consider a bottom-up approach on the FIPA-SL grammar (FIPA, 2002e)[pages 2-4]: *VariableIdentifier*, *Variable*, *Wff.*, etc.)

For intelligent agents in real world scenarios the capability to use complex queries is essential. Assuming that these queries are of a higher complexity than the one in a trivial case, where one or two variables in a boolean logic domain range have to be handled. Additionally, the used ontological concepts have to be instantiated, their slots have to be initialized with corresponding values, and finally the new concept instance must be integrated in the described FIPA-SL term. An example for this can be consulted in (Caire, 2002)[pages 20-21].

Another point to be considered is that a sent message also has to be checked syntactically and semantically on the receiving side. For that purpose the message is passed to a parser for syntax-checking first and then the used ontological concepts must be extracted by hand. E.g. if the received message is a query then the used operators have to be mapped to corresponding operators of the programming language and if necessary the used logic as well. If the received message is a request for a specific action, then the term specified in the used ontology for this agent action needs to be bound, e.g to a JAVA method that executes this action.

This again is a tedious job that in the field of artifical intelligence usually is done by inference machines and not by developers. One possibility to simplify this process is to create templates for message contents with the ontology design tool Protégé and the Bean Generator plugin. Though this is not a very sophisticated solution because intelligent agents should be able to generate queries and requests at runtime.

3 EXISTING APPROACHES

The following approaches resolve the necessity of implementing each speech act manually for FIPA-compliant interactions.

The JADE Semantics Add-on (Bellifemine et al., 2007) provides basic support for FIPA-SL processing and interpretation. An incoming message is forwarded to an SL parser first and then the resulting terms are evaluated with an embedded knowledge base (KB) by syntactical term substitution. This however seems to be the main disadvantage of this approach as it can result in overflowing work memory for large amounts of terms.

In Jadex (Pokahr et al., 2005) incoming messages are handled as events according to their speech act type. For each speech act type a corresponding procedure is required that processes the speech act with respect to its content. The Jadex approach also presumes the separate programming of procedures in Java and their postponed configuration with the help of files, written in XML and a Java-like syntax. In our opinion this language mismatch is the weak point of this approach because it assumes that a developer must know more than one programing language and paradigm.

Zeus (Nwana et al., 1999) is another agent development framework that is not fully FIPA-compliant. Each incoming message is processed by a message handler and is then dispatched to the relevant components of the agent, e.g. action plans. All actions have to be programmed by the agent developer in advance and must be loaded to the action base of the agent before the execution is started.

In all these approaches one major drawback remains. Namely the fact that agents and their behavior have to be programmed and compiled completely before their execution. A reconfiguration of an agent behavior at runtime is possible only in a very limited way, e.g. by replacing its behavior object by another one. This requires a modification of the behavior program and its recompilation, and therefore only shifts the inflexibility from agent to behavior level. This drawback has a high impact on the development time of agent based applications, due to time consuming testing and evaluation phases and consequential restart cycles of the whole application.

3.1 Requirements

For the development of a model which resolves this problem the following requirements must be taken into account.

3.1.1 Standardized Interaction and Communication Modes

The usage of standards is a requirement for interoperability in all fields of software development and significant for the success of an application. It provides for using different components side by side or conjointly. Especially in agent technology in its role as intelligent middleware standardized interfaces are the key for successful system and service integration. Distributed agents which offer and use varying services need a well-defined interaction layer in order to solve problems in cooperative mode.

3.1.2 User-friendly Programming Interface

A user-friendly programming interface is demanded in order to reduce the agent development effort. The agent should be programmable in a higher level programming language at runtime without the need of recompiling the source code. For this purpose a rule-based systems is appropriate. To combine the standardized communication and interaction modes with this approach the rule-based programming paradigm the rule-based system must be integrated in the agent framework.

3.1.3 Consolidation of Requirements

The consolidation of standards and a high level programming interface promise to resolve the remaining drawback. Thus in our approach we try to integrate both requirements into one intelligent software component, the *JamochaAgent*. In this way *JamochaAgent* offers the developers of agent-based applications the advantage of convenient declarative programming at runtime and at the same time ensures interoperability with existing and well established systems.

4 APPROACH

To meet the above claimed requirements the following standard and rule-based system are used in this approach. The FIPA standard is already prevalent in agent technology and has been admitted as a new chapter of the IEEE computer society. Thus the approach introduced here is based on the FIPA interaction and service models. Particularly the agent communication language (ACL) (FIPA, 2002a) which is based on speech acts, the content language FIPA-SL and the interaction protocols provided by FIPA are the essentially considered elements. Therefore, the agent framework JADE which supports these models is chosen as the preferred agent development platform.

As programming interface for the rule-based system CLIPS (C Language Integrated Production System) (ASD, 2007) is chosen here, due to its user-friendliness and the convenient way to describe the behavior of an agent by rules and facts in a declarative manner. Additionally it provides numerous well established libraries for solving problems in the AI field.

JamochaAgent inherits FIPA-compliant communication capabilities from the agent component provided by the JADE framework and serves therein only as a communication interface to other agents. The interaction among agents consisting of consecutive speech acts is described by FIPA interaction protocols (FIPA-SC00001G, 2002) (FIPA-SC00023K, 2004) which are mapped onto the rule-based system to allow for interaction protocol adaptations at runtime. Incoming speech acts are consumed and outgoing speech acts are produced by the rule engine in rule-based manner. This also allows for the adaptation of agent behavior at runtime.

5 ARCHITECTURE

Figure 1 gives an overview of the *JamochaAgent* architecture. At first the functionality of the internal components of the *JamochaAgent* is introduced, then the operation modes of the agent are discussed. Therein the consumption and production of speech acts are shown in Figure 2 and Figure 3.

Incoming message queue containing all the incoming ACL messages of the agent.

Outgoing message queue containing all the outgoing ACL messages of the agent.

Rule-based system consisting of a KB, a rule based inference engine. The KB contains a description of the agent world and the rule base a rule-based description of the agents behavior and a rule-based definition of the FIPA interaction protocols. The inference engine applies the rules from the rule base to the KB to deduce the next action of the agent.

FIPA-SL to CLIPS adaptor transformes an ACL message from the incoming message queue represented in FIPA-SL content language into the equivalent CLIPS programming language representation of the rule-based system.

CLIPS to FIPA-SL adaptor transforms propositions of the CLIPS programming language describing a speech act into appropriate FIPA-SL representation and generates an ACL speech act that is added to the outgoing message queue of the agent.

User interface enables the user to inspect the content of the KB, the content of the rule base, and the processing mode of the inference engine with the help of a graphical user interface or a shell with a command line interface.

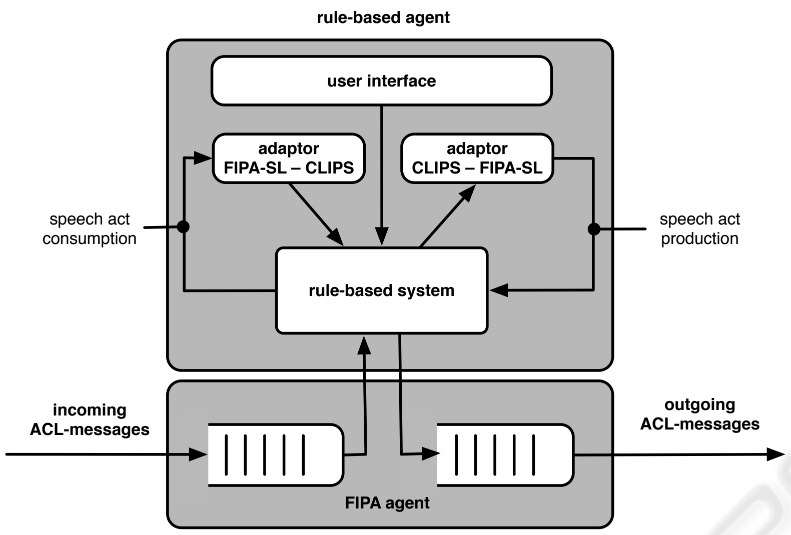
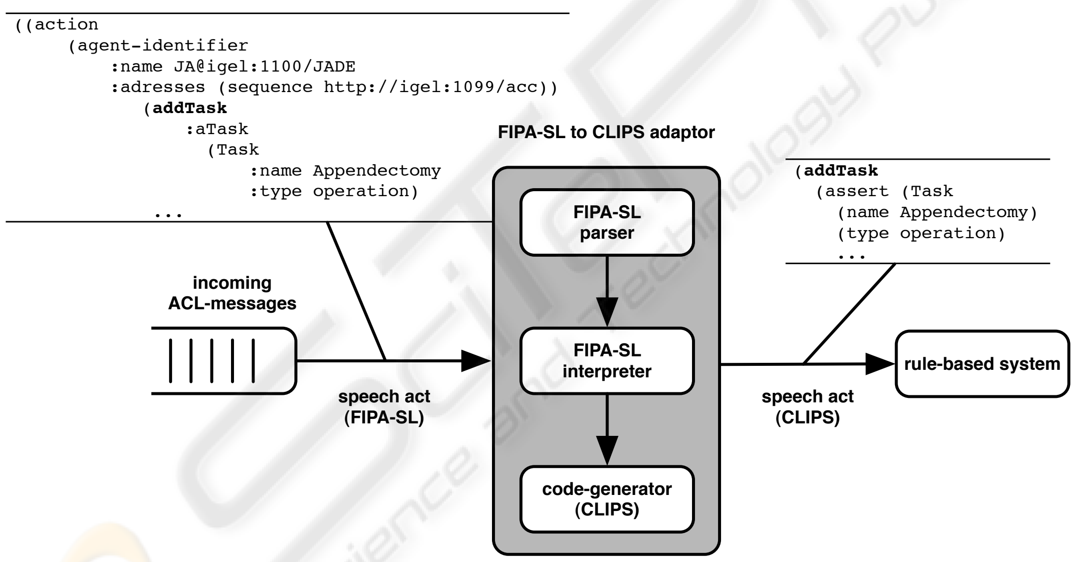


Figure 1: Architecture JamochaAgent.


```
((action
     (agent-identifier
         :name JA@igel:1100/JADE
         :adresses (sequence http://igel:1099/acc))
            (addTask
                :aTask
                  (Task
                      :name Appendectomy
                      :type operation)
            ...
```

```
(addTask
  (assert (Task
    (name Appendectomy)
    (type operation)
    ...
```

Figure 2: ACL processing.

A *JamochaAgent* consumes an incoming message with respect to the used interaction protocol by asserting a new fact into the KB of the agent. The new fact activates a rule that is fired. The execution of this rule transforms the content of the ACL message given in FIPA-SL to the CLIPS language with the help of the FIPA-SL to CLIPS adaptor and saves the result in a special slot of the same fact. This activates another rule that executes the CLIPS code produced in the rule-based system. Figure 2 shows the architecture of the FIPA-SL to CLIPS adaptor, that is composed of a FIPA-SL parser, a FIPA-SL interpreter, and a CLIPS code generator. The content of an ACL message given in FIPA-SL is processed by the FIPA-SL parser. The resulting syntax tree is used by the FIPA-SL interpreter to configure descriptions of the syntactical elements of the CLIPS language. From these descriptions the CLIPS code generator will produce the CLIPS code semantically equivalent to the FIPA-SL content.

The architecture of the CLIPS to FIPA-SL adator is shown in Figure 3. The adaptor consists of speech act templates and a FIPA-SL code generator. An acting agent writes a proposition describing its action

Table 1: Semantic characterization elements of $\mathcal{L}_{SL}$.

Element	**Description**
$Action$	An action that is intended to be executed by the receiving agent.
$ConstraintCondition$	A condition that has to be met in order to execute an action.
$Proposition$	An expression that describes one or more elements of the underlying ontology.
$ReferentialExpression$	An expression that describes elements of the underlying ontology and additionally holds a reference variable defining a specific scope.
$speechact$	An additional speech act embedded in the given speech act.

Table 2: Semantic characterization elements of $\mathcal{L}_{KB}$.

Element	**Description**
$Fact$	A fact in the underlying KB.
$FunctionCall_I$	(*I* for *Immediate*) A function call that will be executed in the underlying KB.
$FunctionCall_{OD}$	(*OD* for *On Demand*) A function call that can be executed in the underlying KB, but not immediatly. Mostly it only has informative purpose and is stored in string representation.
$Rule$	A rule that can trigger one or more actions in the underlying KB.
$ConceptSet$	A simple character string that can be interpreted in the underlying ontology in a reasonable way.

(query, request, inform, etc.) in a new fact that is based on the corresponding speech act template. The new fact activates a rule that fires and transforms the CLIPS proposition with respect to the chosen template (speech act type) to FIPA-SL content. After the transformation of the agent action into the FIPA-SL representation, this is embedded in a FIPA-ACL message and forwarded to the outgoing message queue of the agent.

6 MAPPING SPEECH ACTS ONTO RULE-BASED SYSTEMS

The semantics of FIPA-SL messages need to be preserved when mapping speech acts onto a rule-based system. Therefore it is necessary to define a transformation from FIPA-SL representation to another knowledge representation (in this approach: the CLIPS language). The following definitions are given to formalize this transformation:

Definition 1. *Let $\mathcal{L}_{SL}$ be the language of valid SL expressions and $\mathcal{L}_{KB}$ the language of valid expressions of a KB.*

Then $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB}} : \mathcal{L}_{SL} \to \mathcal{L}_{KB}$ is a transformation, that maps an element of $\mathcal{L}_{SL}$ onto appropriate elements in $\mathcal{L}_{KB}$.

The semantic characterization elements of the languages $\mathcal{L}_{SL}$ and $\mathcal{L}_{KB}$ are listed and shortly described in the tables 1 and 2 respectively.

The mapping of the SL elements varies for each speech act (see (FIPA-SC00037J, 2002)). Thus the set of all FIPA-compliant speech acts is defined first and based on that the transformation of single elements for each invidual speech act is defined.

Definition 2. *S is the set of all speech acts defined in the FIPA standards (FIPA-SC00037J, 2002).*

In the following paragraphs the transformation $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB}}$ for all speech acts in S is defined separately. For this reason $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB}}$ is extended by an index $s \in S$ for each speech act.

Definition 3. *Let $s \in S$.*

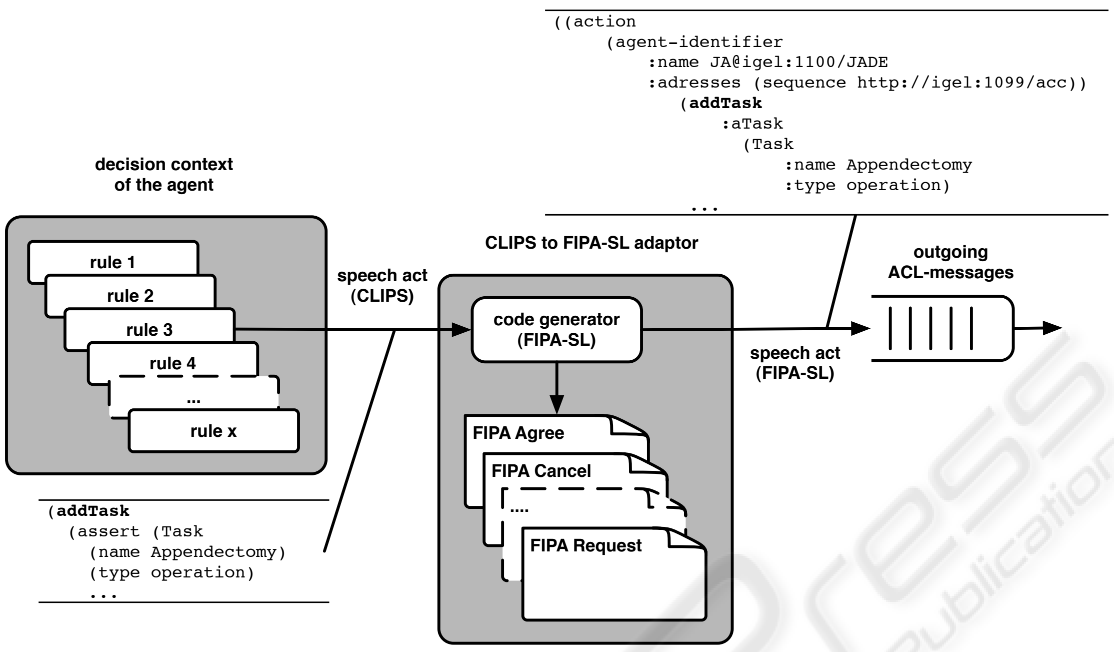
Then $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},s}$ is the transformation $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB}}$ that only maps exactly the elements of $\mathcal{L}_{SL}$ onto elements of $\mathcal{L}_{KB}$ that are used in the speech act s.

Due to limited space only a selection of speech act transformations is given here. A complete list of defined speech acts will be published in a technical paper. The selection discussed here covers all speech acts which can occur in a FIPA-query (FIPA, 2002c) and FIPA-request interaction protocol (FIPA, 2002d).

6.1 Agree

An *agree* speech act informs the receiver that the sender agrees to perform some action the receiver previously requested him to do. This speech act consists of an $Action$ the sender is going to execute and a $Proposition$ that gives one or more conditions that must fulfilled prior to the execution. The transformation steps for $s = agree$ are defined as follows:

- $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},agree} : Action \mapsto FunctionCall_I$



```
((action
     (agent-identifier
         :name JA@igel:1100/JADE
         :adresses (sequence http://igel:1099/acc))
            (addTask
                :aTask
                  (Task
                      :name Appendectomy
                      :type operation)
              ...
```

```
(addTask
  (assert (Task
    (name Appendectomy)
    (type operation)
    ...
```

Figure 3: ACL production.

- $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},agree} : Proposition \mapsto ConceptSet$

The information embedded in an *agree* speech act is just added to the KB and can be interpreted using the underlying ontology.

6.2 Failure

A *failure* speech act indicates that the sender tried to perform some action but failed during the execution. It consists of an *Action* the sender tried to perform and a *Proposition* denoting the cause for the failure. The transformation steps for $s = failure$ are defined as follows:

- $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},failure} : Action \mapsto FunctionCall_I$
- $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},failure} : Proposition \mapsto ConceptSet$

The information embedded in an *failure* speech act is just added to the KB and can be interpreted using the underlying ontology.

6.3 Inform / Inform-if / Inform-ref

An *inform* speech act tells the receiver that the embedded statement is true or not. *inform-if* and *inform-ref* are just so called macro speech acts that further describe the content. *inform-if* gives information about the truth value of a statement and *inform-ref* provides one or more items that fulfill a given condition. In this section those three speech acts will be treated combined as *inform*.

The *inform* speech act just consists of one *Proposition* that is regarded as fulfilled and send to the receiver for informational purpose. The transformation step for $s = inform$ is defined as follows:

- $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},inform} : Proposition \mapsto ConceptSet$

The information embedded in an *inform* speech act is just added to the KB and can be interpreted using the underlying ontology.

6.4 Not-understood

A *not-understood* speech act states that the sender does not understand an action performed by the receiver or that he could not interpret a previous message of the receiver. The latter can happen because of an unknown ontology for example. This speech act consists of an *Action* that was not understood and a *Proposition* giving the cause for the not understanding. The transformation steps for $s = not-understood$ are defined as follows:

- $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},not-understood} : Action \mapsto FunctionCall_I$
- $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},not-understood} : Proposition \mapsto ConceptSet$

The information embedded in an *not-understood* speechact is just added to the KB and can be interpreted using the underlying ontology.

6.5 Query-if

With a *query-if* speech act the sender can ask the receiver if a statement is true or false. The *query-if* speech act only contains a *Proposition* that should be checked for its validity by the receiver. The transformation step for $s = query - if$ is defined as follows:

- $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},query-if} : Proposition \mapsto Rule_{True}, Rule_{False}$

The embedded *Proposition* is mapped onto two rules. $Rule_{True}$ fires only if the statement is true, $Rule_{False}$ fires if the statement is false, so for each option the corresponding interaction protocol speechact can be triggered.

6.6 Query-ref

With a *query-ref* speech act the sender asks the receiver to supply one or more entities that validate the embedded proposition. This speech act only consists of one $ReferentialExpression$ for which the receiver should find valid entities. The transformation step for $s = query - ref$ is defined as follows:

- $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},query-ref} : ReferentialExpression \mapsto Rule_{True}, Rule_{False}$

The embedded $ReferentialExpression$ is mapped onto two rules in the KB. $Rule_{True}$ fires exactly for those facts that satisfy the conditions of the $ReferentialExpression$. $Rule_{False}$ fires if the $ReferentialExpression$ references an empty set.

6.7 Refuse

Using a *refuse* speech act the sender informs the receiver that he refuses to perform an action that he has been asked to do before. This speech act consists of the *Action* which the sender refuses to execute and a *Proposition* giving the cause for the refusal. The transformation steps for $s = refuse$ are defined as follows:

- $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},refuse} : Action \mapsto FunctionCall_I$
- $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},refuse} : Proposition \mapsto ConceptSet$

The information embedded in a *refuse* speech act is just added to the KB and can be interpreted using the underlying ontology.

6.8 Request

With a *request* speech act the sender informs the receiver that he wants him to perform some action. This speech act contains an *Action* that should be executed by the receiver. The transformation step for $s = request$ is defined as follows:

- $\tau_{\mathcal{L}_{SL},\mathcal{L}_{KB},request} : Action \mapsto FunctionCall_I$

After mapping the *Action* of a *request* speech act onto a function of the KB it is executed in its context.

The reverse mapping of speech acts from CLIPS to FIPA-SL representation is another complex part of this approach and will therefore be discussed soon in another paper with appropriate extent.

7 EXAMPLE

The following example shows how a received *query* speech act is processed by the agent.

Assume the *JamochaAgent* features some KB concerning cars whose appropriate ontology is called *cars*. Items in this ontology are cars having a name, a color and a certain horsepower. A query to this ontology can look like this:

```
(query-ref
 :sender (agent-identifier :name ↩
     buyer@nocturna:1099/JADE :addresses ↩
     (sequence http://nocturna:7778/acc))
 :receiver (set (agent-identifier :name ↩
     seller@nocturna:1099/JADE))
 :ontology cars
 :content "((all ?x
    (Car    (name ?x)   (color green)   ( ↩
        horsepower 150))))"
 :language fipa-sl
 :protocol fipa-query)
```

Code 1: Example query.

The ACL message is asserted as fact in the rule engine. The content of the speech act is automatically transformed into the following CLIPS code by the rule corresponding to the *query-ref* speech act:

```
(bind ?*queryRef-temp* (create$))
 (defrule query-ref
   (Car
     (name ?x)
     (color green)
     (horsepower 150)
   )
   =>
   (bind ?*queryRef-temp* (insert-list$ ?* ↩
       queryRef-temp* 1 ?x))
 )
 (fire)
 (undefrule "query-ref")
 (assert
   (agent-queryRef-result
     (message <fact of the query-ref message ↩
         >)
     (refOp all)
```

```
      (items ?*queryRef-temp*)
    )
)
```

Code 2: Example query as CLIPS code.

The *agent-queryRef-result* fact activates and fires another rule that will process the CLIPS code. It generates an answer with all the possible items matching the query. The buyer agent then would receive a list of the names of all green cars with 150 hp. More details regarding the *JamochaAgent* examples are provided in the web[1].

8 CONCLUSIONS AND OUTLOOK

Combining knowledge processing with standardized interaction mechanisms into one software component enables agent application developers to program their problem solving methods (algorithms, heuristics) in a convenient manner. This is provided by the declarative programming language CLIPS and the well integrated (hiding technical details from the developer) speech act production and consumption mechanisms implemented in the presented approach.

With the help of the developed intelligent software component *JamochaAgent* we speed up the development process of intelligent agent-based applications (Krempels and Panchenko, 2007) leading to shorter development cycles and a faster time to market. The main applications areas of the *JamochaAgent* are distributed agent-based applications in planning, coordination, and scheduling in the areas: health care, supply chain management, aviation, etc.

Future work will be focused on agent-based infrastructures for cloud computing in which the *JamochaAgent* is used as main component for distributed reasoning and knowledge processing.

ACKNOWLEDGEMENTS

This work is motivated by experiences made in the German priority research programme 1083 Intelligent Agents in Real-World Business Applications (2000-2008).

[1]http://tinyurl.com/JamochaAgent

REFERENCES

(2007). *CLIPS Reference Manual: Basic Programming Guide*, quicksilver beta edition.

Bellifemine, F. L., Caire, G., and Greenwood, D. (2007). *Developing Multi-Agent Systems with JADE*. Wiley.

Botello, L., Willmott, S., Zhang, T., and Dale, J. (2001). Multilingual agents: Ontologies, languages and abstractions. Technical report no. 01/362, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland).

Caire, G. (2002). JADE Tutorial - Application-defined Content Languages and Ontologies. Manual, TILAB.

FIPA (2002a). FIPA ACL Message Structure Specification. Standard, Foundation for Intelligent Physical Agents, http://www.fipa.org/.

FIPA (2002b). FIPA Iterated Contract Net Interaction Protocol Specification. Standard, Foundation for Intelligent Physical Agents, http://www.fipa.org/.

FIPA (2002c). FIPA Query Interaction Protocol Specification. Standard, Foundation for Intelligent Physical Agents, http://www.fipa.org/.

FIPA (2002d). FIPA Request Interaction Protocol Specification. Standard, Foundation for Intelligent Physical Agents, http://www.fipa.org/.

FIPA (2002e). FIPA SL Content Language Specification. Standard, Foundation for Intelligent Physical Agents, http://www.fipa.org/.

FIPA-SC00001G (2002). FIPA Abstract Architecture Specification.

FIPA-SC00023K (2004). FIPA Agent Management Specification.

FIPA-SC00037J (2002). FIPA Communicative Act Library Specification.

Krempels, K.-H. and Panchenko, A. (2007). KR-driven Development Process Integration. In Vendetti, J., Hopper, T., and Tudorache, T., editors, *Proc. of 10th Intl. Protégé Conference*, Budapest, Hungary.

Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C. (1999). Zeus: A toolkit and approach for building distributed multi-agent systems. In *Agents*, pages 360–361.

Pokahr, A., Braubach, L., and Lamersdorf, W. (2005). Jadex: A bdi reasoning engine. In Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 149–174. Springer.