JAISCR, 2011, Vol.1, No.4, pp. 301–314

MODELING & UNDERSTANDING ENVIRONMENT USING SEMANTIC AGENTS

Sbastien Dourlens and Amar Ramdane-Cherif

LISV, University of Versailles - St Quentin,
Centre Universitaire de Technologie, 10/12 Avenue de l'Europe - 78140 Velizy, France

Abstract

Understanding environment is a very complex task. Modeling and development of components of the environment in a symbolic way permit to simplify and clarify functional parts of components and interactions. In order to have a complete system specification, a rigorous behavior description is needed. Different levels of behavior abstraction are taking into account. The objective of our semantic modeling is to enhance architectural design and reduce complexity. It permits to agents to understand the environment, manage events and adapt the architecture. All the concepts of the environment like the component models and their behavior models are stored under event frames written in knowledge representation language. We present in this document, a generalized meta-model of behavioral aspects, that indexes the various environment behaviors in three ontologies. We have fully linked abstraction level with modeling and execution of scenarios. We show how software semantic agents can be modeled to build any interactive architecture.

1 Introduction

Complex systems require expressive notations for representing their software architecture. In this field, Component and Domain based Software Engineering has now emerged as a discipline for systems development. Robotic systems or software systems and their environment can often be seen as a very complex architecture in reality. Software engineering needs systems comprehension, design, evolution, reuse, analysis, construction, deployment and automatic reconfiguration. As a definition, environment is a fully observable set of parts of a program or a system, or parts of human environment (house, city and planet) where agents, objects and other entities live inside. We also refer to the work of [1] and [2]. Our definition of understanding and meaning of behaviors is well explained in [3]. In Multi Agents Systems (MAS), agents are autonomous, heterogeneous and dynamic components. The organizational stance permits to deal with complexity and dynamism of component interactions. They are used in modeling and simulation [4, 5]. Agents need Agent Communication Language (ACL). ACLs allow for a more anthropomorphic description of the interactive roles and protocols than connectors. Secondly, they allow for more flexibility, as the equivalent to protocols of connector which can be derived from the communicative action semantics. We propose in this paper a kind of new ADL/ACL based on an environment knowledge representation language (EKRL). ADL and ACL are not oriented to all behavioral aspects of the environment. EKRL is used to define and connect components in order to build, transfer, store, and query events in a knowledge base. EKRL gives a description of behavior and modeling entities including the system or the multiple parts of the system in case of pervasive architecture, ubiquitous network or ambient intelligence.

This work is a new contribution to Artificial Intelligence, in particular, the memory structure and the EKRL presented in this paper. It brings to agent the means to reason about their tasks and environment as human do, with words and by focusing on different levels of abstraction of behavioral aspects of entities living or moving in human environment. This work permits to simplify or reduce the percepts of environment to reason on it (i.e. extracting the meaning using models), to learn by storing new concepts, classes and instances from other specific domain ontologies, and using event models (also described with words) to program the agent. Moreover, our generic agent with its memory may be distributed.

After a state of the art on modeling languages and behavior specifications, we present our environment representation in terms of behaviors, scenarios and any other information stored and exchanged in EKRL.

2 Related Work

In this section, we focus on the value of specifying the behavioral aspects of architectural elements. Also, we explore behavioral aspects in existing main languages.

2.1 Modeling Behavior with Languages

Modeling architecture, components, interaction, and evolution with languages close to natural language simplifies the tasks of programming and using in product engineering for humans. It permits to follow the tendency of semantic web designers to share and reuse exploitable structured knowledge for search engines or other web services (Figure 1). Another reason to use semantic languages is the advanced researches done on propositional and modal logics to be integrated in MAS [6].

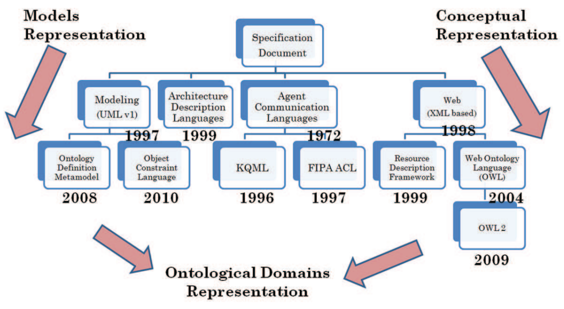

Models Representation
Conceptual Representation
Specification Document
Modeling (UML v1)
1997
Architecture Description Languages
1999
Agent Communication Languages
1972
Web (XML based)
1998
Ontology Definition Metamodel
2008
Object Constraint Language
2010
KQML
1996
FIPA ACL
1997
Resource Description Framework
1999
Web Ontology Language (OWL)
2004
OWL 2
2009
Ontological Domains Representation

Figure 1. Models & Concepts Representation

A good comparison of ADLs has been done in [7]. ACLs for specifying behaviors of agent components are well presented in [8, 9] and [10]. Web semantic languages like RDF[1] and OWL[2] are used to specify behaviors at different parts of the architecture or components bringing disambiguation. Knowledge representation language (KRL) with ontologies [11], as shown in this paper, is more powerful and explicit to model, specify and validate architecture behavior.

2.2 Comparing RDF, OWL, UML and OCL

RDF is a set of concepts or a language used for describing knowledge. OWL is a set of concepts or language for modeling that knowledge. UML is a semi formal language for class-based modeling and OWL comprise some constituents that are similar in many respects like classes, associations, properties, packages, types, generalization and instances [12]. Despite of the similarities, both approaches present restrictions that can be overcome by integration. On one hand, a key limitation of UML class-based modeling is that it allows only static specification of specialization and generalization of classes and relationships, whereas OWL provides mechanisms to define these as dynamic. It means that OWL permits recognition of generalization and specialization between classes as well as class membership of objects based on conditions imposed on properties of class definitions.

Among ontology languages, Web Ontology Language (OWL) is the most used for semantic web applications. OWL offers a more expressive and extensible manner of modeling data and provides versatile ways to describe classes and, based on such descriptions, it allows type inference. Indeed, OWL provides various means for describing classes, which may also be nested into each other such that explicit typing is not compulsory. One may denote a class by a class identifier, an exhaustive enumeration of individuals, property restrictions, an intersection of class descriptions, a union of class descriptions, or the complement of a class description. OWL provides important features complementary to UML and OCL/MOF [13] that would improve software modeling: it allows differ-

[1]RDF: Resource Description Framework http://www.w3.org/RDF/

[2]OWL 2 : Ontology Web Language version 2 http://www.w3.org/TR/2009/WD-owl2-overview-20090327

ent manners of describing classes; it handles these descriptions as first-class entities; it provides additional constructs like transitive closure for properties; and it enables dynamic classification of objects based upon class descriptions. OWL ontologies can be operated on by reasoners providing services like consistency checking, concept satisfiability, instance classification and concept classification. Specification of OWL2 contains several sublanguages, or profiles [14], which offer increased tractability at the expense of expressivity [15]. Ontologies and web service architecture are now used to design and adapt MAS [16] bringing interoperability [17]. EKRL can easily integrate OWL.

2.3 Modeling Behaviors

Specifying behavior is a way to add semantic details to structural elements and their interactions that are time related and have other contextual characteristics [18]. It is common to define a model of Meta for the four levels of abstraction of the behaviors (Table 1).

Table 1. Behavior Layers. Lower abstraction level is 1, higher abstraction level is 4.

Level	Behavior Layer
1	Component Model
2	Static Behavioral Model
3	Dynamic Behavioral Model
4	Architectural Model

In meta-modeling, the four layers are defined:

1. A component of the environment or system can be split in two main parts: Behavioral content and Interaction connectors with other components.

2. Static behavior view captures the functionality of a component in a discrete manner, during the system's execution. It represents several states of architectural elements during the system execution. Its description is focused generally on pre- and post conditions specification. It refers to component programs with sequential states.

3. Dynamic behavior view provides, contrary to the static one, a continuous view of how an architectural entity arrives in various states presented in static view during its execution. Its description is based, usually, on state machines. It refers to complex scenarios where component's behavior will join or split.

4. Interaction behavior view provides the external view of the architectural entities, how they interact with and how to reconfigure them in the system using local or global policies.

Most of the current languages are modeling the levels 1 and 2 of software engineering and more often level 3 in system engineering or software parallel programming. Most of languages use network connectors to manage input and output messages. Few are managing behaviors and architecture changes.

2.4 Modeling Scenarios

A scenario can be expressed by a list of states and actions like in state graph used for execution, simulation and multimodal interaction to manage events. Most of the time, these formal graphs are Petri nets, discrete Markov decision processes (MDP) or continuous state space models. We consider that events are messages transmitted to particular components or broadcasted across environment. These messages (called events) are changes of the environment that must be stored in a knowledge base giving us current state, last states and actions related to these events.

Table 2. Scenario layers. Lower abstraction level is 1, higher abstraction level is 4.

Level	Scenario Layer	Behavior
1	Elementary action s	Simple
2	Composed actions or simple scenario	Sequential or Static
3	Complex scenario or composed scenario	Complex or Dynamic
4	Scenario Management	Architectural

Table 2 is deeply linked to Behavior table in the third column. In meta-modeling,

1. Simple behavior concerns elementary actions realized by a structural or physical component.

2. Static behavior represents a composition of elementary actions of the level 1. It permits to design the behavior of components of the system and the rules of sequential execution of tasks.

3. Dynamic behavior implies the knowledge and

actions are related to the evolution of the system in space, time and other contextual roles. It refers to several composed actions in a complex scenario or the change of composed actions under different conditions.

4. Architectural or Interaction behavior contains meta-scenarios to apply on components as in an external view of the system. We can change objectives online to adapt the system to the environment and generate a reorganization of the architecture.

3 Environment Knowledge Representation Language

To represent what is happening in the environment during simulation or execution, in terms of meaning or interpretation, the principle presented in this section is to store and fuse perceived events by respecting the previous models of behaviors and scenarios defined in the two previous sections. A formal grammar is required to achieve this task. *Meta-concept* and *Concept* are parts of Meta ontology (Figure 3) and Concepts ontology (Figure 4). Models ontology contains event models and event instances of a model encoded in EKRL.

3.1 EKRL Grammar of Models

EKRL can be denoted *L(P(RA,. . . RA))* where *L* is our event language, *P* is a predicate (*event* name representing an action, also called open formulas in formal logic), *R is* a *role* and *A* are arguments *args* of a role. List of *R* and *A* forms a *frame*. Figure 2 presents EKRL grammar. EKRL can be integrated to a knowledge base with an inference engine. It brings expressivity, removes ambiguities of natural language and permits model checking with its formal logic.

```
event:  rootpredicate ":" predicate frame
frame: frame role args
role:    Meta-concept
args:    NIL | args arg
arg:     Concept | string | number | operator "("
args ")"
operator: Meta-concept
```

Figure 2. EKRL Grammar of an Event Model

In our case, a model corresponds to an event predicate. EKRL past events (facts), event models and query models will be stored in the Models Ontology (Figure 6). Ontologies are made of generalization and specialization relations between concept or event nodes in tree structures. Each model is an event, atomic or not. Events ontology respects behavioral levels.

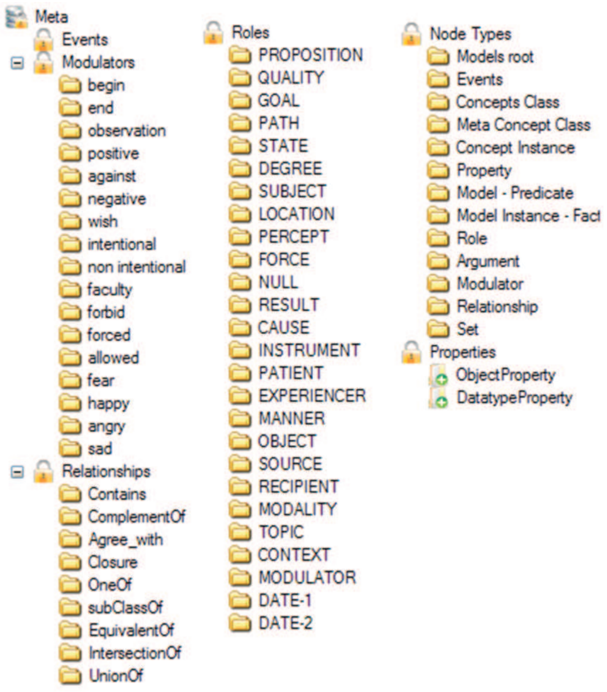


Figure 3. Meta Ontology

Root-predicates are parent predicates, directly located under the Models root of the tree. They permits to improve event search. A predicate is an event model. A fact is a past event in a frame containing roles from Meta ontology and arguments. Argument may be operator (or relationship) from Meta Ontology and concepts from the Concepts ontology. They are stored in knowledge base under its predicate. A future action or scenario to execute is also an instance of event models. Date roles indicate if fact is from past, present or future.

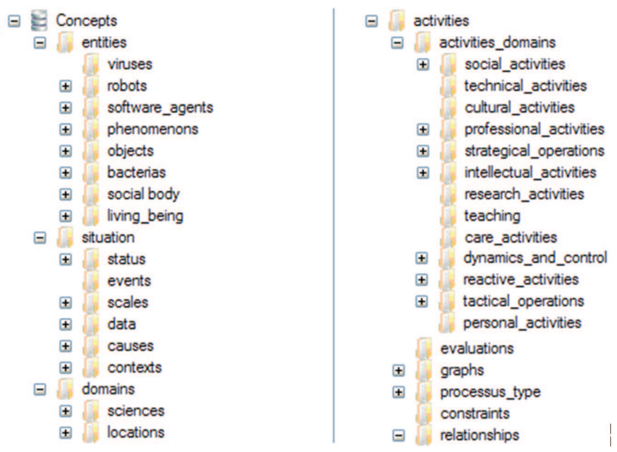


Figure 4. Concepts Ontology (sample)

3.2 Modeling Entities in EKRL

Environment is a set of entities like human, agents, robots or objects. They are linked to different attributes. For instance, *cars* is a concept class with *subClassOf(terrestrial transport)* relationship like *trains, buses* or *trucks* classes. Instances of concept *trains* can be *Orient Express*. Metaconcepts are used to build all ontologies (nodes properties and relationships). The root of the Concept ontology is named *Thing*. Each node of ontologies contains a unique reference (RDF *about* term), a comment (RDF *comment* term) and relationships like *owl:complementOf, owl:oneof, owl:unionOf, rdfs:subClassOf, rdfs:EquivalentOf or owl:intersectionOf*. These give us the possibility to directly insert OWL files into our Concepts ontology. Meta and Concept ontologies are not bounded to external entities in the environment but they also contain internal parts of the designed system interacting with the environment.

One key idea must be highlighted here. Meta and Concepts ontologies are built with OWL v2 formalism but we have limited them to a tree structure and not a complex graph to ensure a fast search browsing of concepts and instances and especially ensure graph closure. Direct relationships between a class and a subclass are only allowed. So now questions are: - Where are others more complex OWL relationships? and - Where are the logical rules to infer the properties of these concepts? The answer is in the section 3.3, they are parts of the facts description and they are simply EKRL event models.

To improve later the system and manage fuzziness, we have added some values to each node. More precisely, scaled values attached to concepts of some nodes: *access control list* for rights, *rank* for ordering, *strength* for % of truth and *status* for active/not active.

3.3 Modeling Behaviors in EKRL

In the previous section, we showed concepts ontology is a tree structure. The same relationships can exist between models. That's why we design our Models ontology exactly like the Concepts ontology. The main difference is that the node structure of models respects the EKRL grammar. So they are much complex and each role into the model contains a formula on concepts (case of an event instance) or other models (case of a rule model). Figure 5 presents how concepts and models instances, metaconcepts and concepts are attached to the Models ontology.

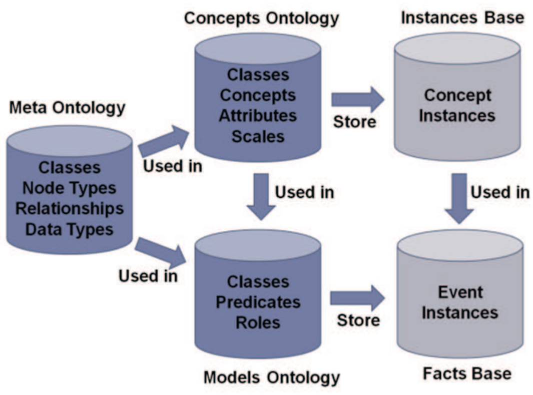


Figure 5. Meta, Concepts & Models ontologies

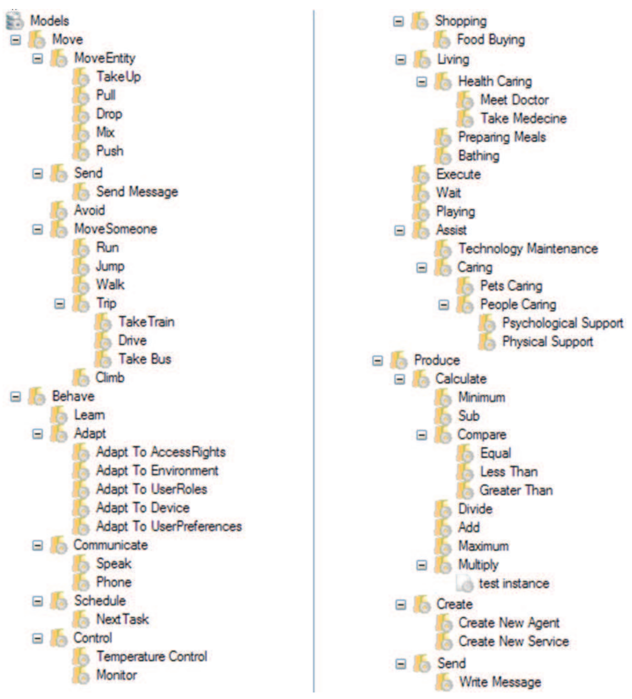


Figure 6. Models Ontology (sample)

Behaviors are modeled by a list of events. All event models and past events (model instances called "facts") are stored in the Models ontology (Figure 6). EKRL root-predicates correspond to all possible events happening to components. Under these root predicates, subclasses of predicates will stand to describe more refined event models respecting the subsumption relation of the Models ontology.

Table 3. Root predicates

Name	Description
Exist	New component appears or exist
Receive	Sense or receive
Own	Membership relations between entities or objects
Move	All simple moves, exchanges or transfers (even virtual like bank transfer)
Produce	Transformation or build process
Behave	All complex moves, adaptation &interaction schemes
Experience	Use or experiment, evaluate & measure

In Table 3, we give a description of the root-predicates. Exist, Own and Receive are root-predicates corresponding to scenario level 1 (simple behavior). For example, *Exist:EntityPosition* is an event model indicating a change in the position of an entity. Move and Produce are root-predicates corresponding to scenario level 2 (sequential behavior in section 2.4). *Move:RobotWalk* is another event model, it is a composition of several memorized past facts under *Exist:EntityPosition* event model. *Behave* and *Experience* are root-predicates at level 3 modeling complex or dynamic behaviors.

Under these root-predicates are all specialization predicates formed from the root-predicate and at different levels of the tree. Sub-predicates are specialized from their parent predicates. So parent predicates subsume children classes and children predicates specialize parent predicates.

The structure of our ontology itself realizes the meaning function.In our example in Figure 7, Tables 4 and 5, we note that *Behave:Gym* represents a more complex scenario using *Move:Walk* and *Move:Jump* level 2 events. *coord* is an AND operator and concerns a sequence of events.

Behave is not limited to human or robot entities so any agent or processes may have a specific behavior of level 3 (section 2.4). *Behave* can also be used for architectural changes by replacing or disabling components of the level 4. Level 4 also corresponds to the terms in the Meta ontology and permits to refine the ontologies. Any other root predicate may be employed to any kind of events while the integrity of all behavioral events in Models ontology is respected. We have only presented on the figure the root predicates and predicates that we have been used in our agents.

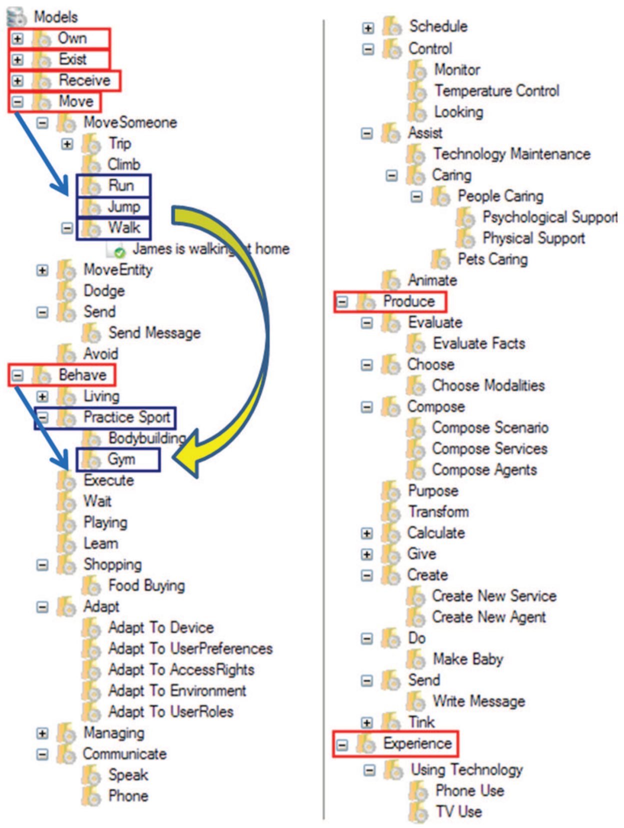


Figure 7. Gym Composite Model

Our EKRL matches well behavioral levels since the models ontology is partitioned in sub trees of root predicates for a fast querying and reasoning system. EKRL Formal grammar is used to generate and describe components and events. EKRL Formal language is used to query facts (instances) and recognize scenario using meta operators and concepts in a role of an event model. Roles are like properties in OWL. Roles in our EKRL describe an event happening on these entities or objects in all possible contexts: temporal, spatial, acoustic, visual, danger, medical, emergency and so on. Examples of roles are SUBJECT (who or what is concerned), OBJECT (objective, goal), CONTEXT, CONTENT (values or string), LOCATION (space), DATE (date and time), and so on. Following examples show that the granularity in multiple contexts is taken into account. *Gym* is a predicate (i.e. a meaning) of a higher level of behavior in this example; Walk and Jump are middle level. The key idea is that *Gym* model is a rule where precondition is a formula on past events and postcondition is *Gym* composed instance.

Table 4. Event Models

Move:Walk Subject: human_beings Location: Living room Context: Indoor activities Date: date time
Move:Jump Subject: human_beings Location: Living room Context: Indoor activities Date: date time
Behave:Gym Object: coord(Move:Walk,Move:Jump) Subject: human_beings Location: Living room Context: Indoor activities Date1: date time start Date2: date time end

Table 5. Fact (Instance of Gym Model)

Behave:Gym Object:coord(Move:Walk,Move:Jump) Subject: John Location: Living room Context: Indoor activities Date: 20/07/2010 17:00 Date: 20/07/2010 17:45

4 Environment Modeling Tool

4.1 Agents' Architecture to Model Environment

We consider all components of the architecture as autonomous agents working in parallel. Agents (Figure 8) are pieces of software with communication ability in a distributed network.

Agents may be very simple, reactive or complex programs; they may perceive and translate sensor values in EKRL and even control hardware parts like drivers. Our aim is to define generic agents to be used in any ambient, ubiquitous or pervasive architecture in order to solve interaction with the environment issues. Agents for interaction must realize preconditions and postconditions of a multimodal decision process. Accordingly, two families of agents are defined: - Fusion agents to build the precondition under the form of composed events with events coming from sensors or other agents and, - Fission agents to reduce high level events to simple orders to be executed by the hardware actuators.

In a structural point of view, the first layer of fusion agents will be able to produce composed events of level 2 (Figure 3) only by using level 1 events. Other layers will use only level 2 events to produce level 2 and 3 events. Agents change their state in a required time.

Figure 9 presents our cognitive memory, into an agent, connected to the environment. Each semantic agent will have a memory designed to store and transfer KRL messages called events. This memory is the knowledge base explained in the previous sections. It contains our Meta, Concept and Models ontologies. In Figure 9, we deliberately split memory into ontologies (darkgray) and facts instances (lightgray) to indicate the difference between *Refine, Use, Store* and *Query* operations on messages.. These actions are made of query or response messages. Depending on if the agent type is a fusion agent or a fission agent, as defined above and last events permit to compose a new event. This memory will return an action, an order or an answer to a query.

The architecture can be designed, executed or verified by following the sequence of events stored in the knowledge base. It means that an agent memory completely models agent and its environment behaviors and all other parts of this agent is fully generic code. Model checking can be applied only on ontologies and, in particular, on models inserted in memory. Uncertainty in observation and action depends on sensor and hardware actuator accuracy, on awareness (observability) and on the ability of sensors to give its uncertainty value. Different fusion levels correct this uncertainty but it deeply depends on hardware parts design. It depends also of the accuracy of event models (in roles) designed and stored in Models ontology.

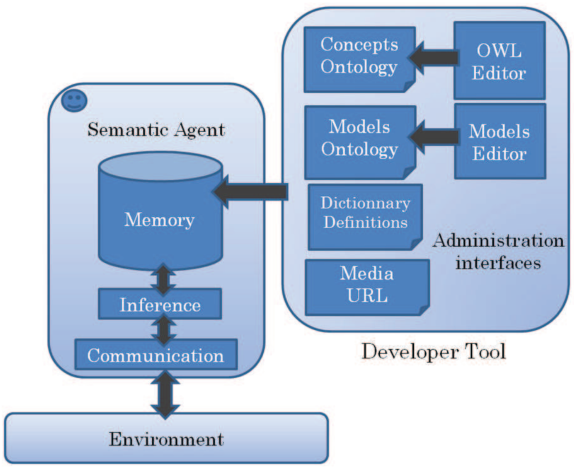


Figure 8. Semantic Agent

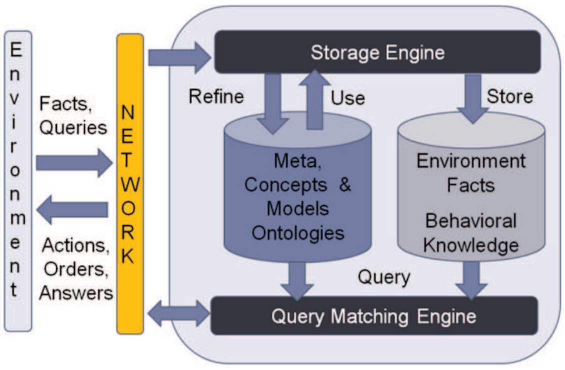


Figure 9. Storage and Querying the agent memory

4.2 Exchanged and Stored Information

Interaction Architecture can be defined with agent components and event messages, we will define the layers of exchanged and stored information (Table 6).

Description of these levels of information:

1. Networking packets and SOAP web services protocol for interoperability. This layer implies a web or TCP-IP network and is useful for ambient and pervasive architecture but it is more related to physical transport of information. These messages are identical to electronic mails with FROM, TO, SUBJECT and CONTENT fields.

2. EKRL Events are used in communication between components. They are textual messages in the CONTENT field of the first layer. This

content is under the form of a EKRL grammar like in the sample (Figure 12).

3. KRL concepts and models in our two ontologies permit the events storage like facts (past events) and models of events, models of query and models of scenarios.

4. Meta concepts are stored in a Meta ontology used to build Concepts and Models ontologies.

All agents are conceived to integrate these 4 layers.

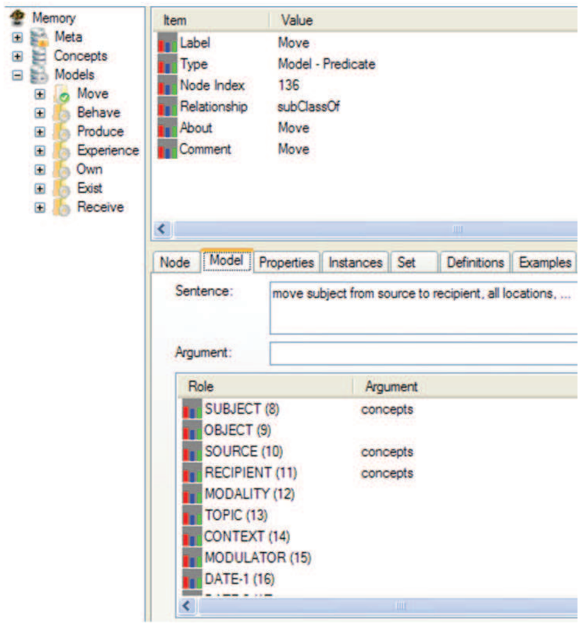
Table 6. Information layers. Lower abstraction level is 1, higher abstraction level is 4.)

Level	Information Layer
1	Networking
2	Events Communication
3	Events Storage
4	Meta Information

4.3 Agent Memory Editor

To implement our approach, we have developed an Agent Memory Editor (Figure 10). It permits to any designer to model agents and environment behaviors. Memory or knowledge base of any agent contains concepts and behavioral models into the ontologies.

Models ontology is organized by concepts classes with subsumption relationships or more generalized models (root predicates) to more specified models. Our agent memory editor permits to modify Meta terms, concepts classes and instances, event models and instances. All ontologies have a common tree root called “Memory”. Each tree node is a term with a name in a natural language (several nodes for several spoken languages), a node type (for example *Model Predicate*), a relationship type (for example *subClassOf*), a unique reference term (equivalent to OWL *about*), and a free comment. If a node is an event model (Figure 11), we can add a natural language sentence telling what’s happened, then select a role and add one or several arguments combined with one or several operators. For models and concepts, user or machine can add instances online using memory’s inference engine (Figure 9). For concepts, user or machine can add properties and a set of concepts linked together independently of the tree structure (replacing complex transformation operations of the matching process). Definitions and examples are used to insert dictionary information to the node including gender and phonetic. We may attach multiple media URI to a node too.



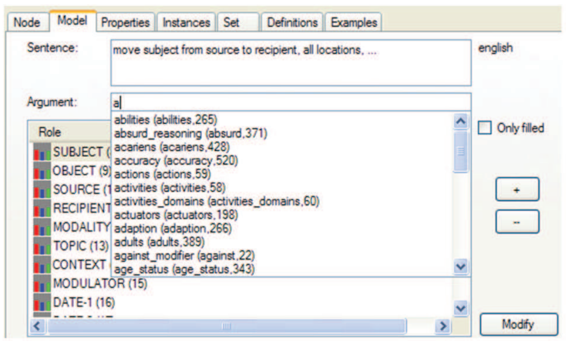
Item	Value
Label	Move
Type	Model - Predicate
Node Index	136
Relationship	subClassOf
About	Move
Comment	Move

Sentence: move subject from source to recipient, all locations, ...

Argument:

Role	Argument
SUBJECT (8)	concepts
OBJECT (9)	
SOURCE (10)	concepts
RECIPIENT (11)	concepts
MODALITY (12)	
TOPIC (13)	
CONTEXT (14)	
MODULATOR (15)	
DATE-1 (16)	

Figure 10. Agent Memory Models Editor



Sentence: move subject from source to recipient, all locations, ... english

Argument: a

abilities (abilities,265)
absurd_reasoning (absurd,371)
acariens (acariens,428)
accuracy (accuracy,520)
actions (actions,59)
activities (activities,58)
activities_domains (activities_domains,60)
actuators (actuators,198)
adaption (adaption,266)
adults (adults,389)
against_modifier (against,22)
age_status (age_status,343)

Role
SUBJECT (
OBJECT (9)
SOURCE (1
RECIPIENT
MODALITY
TOPIC (13)
CONTEXT
MODULATOR (15)
DATE-1 (16)

Figure 11. Model tab with auto-completion

5 Understanding Environments

5.1 Understanding Behaviors in KRL

Environment Behaviors at different levels can be expressed by querying events using temporal and location roles associated to a components group, activities and scenario contexts. Present roles in event models and facts define the possible contexts.

Matching between facts (past perceived events) and a context model permits to find a behavioral context. It is also the case for a matching between scenario models and facts. Context can be extended to any other role terms or concepts of argument. We may understand the environment in many ways and granularities by focusing (cognitive awareness) to a particular event or a long-term and complex scenario.

5.2 Interaction Analysis

Interaction refers to *Fusion* and *Fission* processes realized by semantic agents. Behavior abstraction layers L1 to L4 contain composed events done by fusion agents. Figure 12 shows the quantity of events in each layer. Generally, fusion agents will build new events from layer L2 to Layer L4. L1 contain a lot of simple events most of the time generated by sensors while L4 contain few events with higher level meanings. Complexity of queries will increase from layer Q1 to layer Q4.

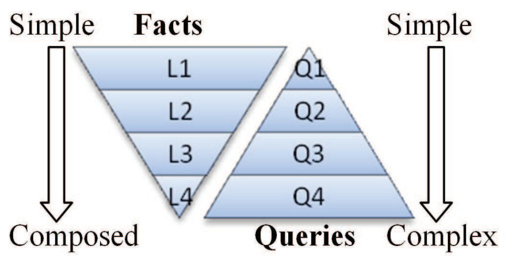


Figure 12. Queries complexity

Events retrieval time increases for queries containing lots of roles with a composition of operators, concepts or events but once an event is composed by another agent, it's our case, access to facts is more simple (linear search complexity) and fast because less facts in the fusion case. In reality, events of different types will be distributed on agents of different abstraction layers decreasing the complexity and quantity of events to manage during a query to their memory.

5.3 Inference Engines

Researchers are looking for mechanisms and models to formalize and reason with domain knowledge using logic and logical inference. Reasoners like RACER[3], KAON2[4], Pellet[5], JESS[6], Fact++[7], JENA[8], Hoolet[9] emerge but they are based on the weak "inference by inheritance" reasoning paradigm because they can only solve, in practice, the most common classification "subsumption" problems. The reasoner performs model checking such that entailments of the Tarski-style model theory are fulfilled. Languages like the Semantic Web Rule Language (SWRL[10]), based approximately on extensions of the inferential properties of Horn clauses and Unary/Binary Datalog to deal with RDF/OWL data structures, appear to be for now as quite limited with respect to the range of their possible applications (web semantic and web services) and complicated to be used in practice.

Due to the main key idea of concepts and models ontology building, we were to make our own inference engine. Our matching process lists all corresponding facts with these query models (a query model is an event model with some roles duly filled) and in case of found operators into a role, it recursively evaluates or calculates it. The list of events can be simple facts or complex scenario. Matching process will then recognize it or build a new composite event that will be sent to other agents. We developed our own inference engine for four reasons:

1. The process may realize an *n*-ary matching and a unification process, building a new event by using query models working on all roles and arguments of facts stored only under a predicate limiting the search path (a tree search).

2. Formula on arguments in a role of our event models can recursively refers to other event models. Meaning that in basic condition of a query model, we use a second order logic and in extreme condition n-order logic but only on

[3] *http://www.racer-systems.com/*
[4] *http://kaon2.semanticweb.org*
[5] *http://clarkparsia.com/pellet*
[6] *http://www.jessrules.com*
[7] *http://owl.man.ac.uk/factplusplus/*
[8] *http://jena.sourceforge.net/*
[9] *http://owl.man.ac.uk/hoolet/*
[10] *http://www.w3.org/Submission/SWRL/*

event models.

3. Our inference engine may compare lots of data types not only binary or numerical like date in temporal logic, possibilities and necessity modal logic, deontic, and so on. We also added scales to attached values to word.

4. Because our ontologies are stored in a database (Figure 13).

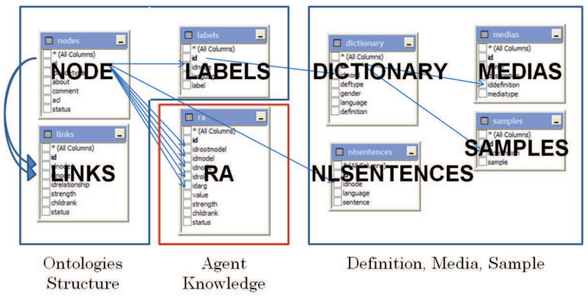


Figure 13. Database Storage

Ontologies nodes are models or concepts stored in the *nodes* table joined to textual labels for any natural languages in the *labels* table. Ontologies relationships between nodes are stored in *links* table. EKRL matching simply wraps and executes SQL queries. Special links composing event models and event instances between concepts and models in a role are stored in the *ra* role-arguments table. *ra* table contains only a list of indexes of nodes.

They are automatically created at the insertion of new facts (i.e. at the reception of a new event) so all semantic and meaning are presents in this table to provide a powerful and fast querying system, matching is just a direct comparison of nodes indexes and not a tree browsing reducing the algorithmic complexity of the searching function.

5.4 Retrieving or Querying Behaviors in EKRL

These operations consists to query knowledge base using models ready to quickly match simple or composed events (past facts). Filing roles like *Date* or *Duration* permits to find a temporal set of events. Filing role like *Location* permits to find a spatial set of events. We can do the same for objects, activities and other concepts. Each of roles can be found using different granularities in space and time. This multi dimensional aspect can also be managed for all roles and combination of roles (i.e. multiple contexts).

Different levels of abstraction of behavioral aspects of the events are simply found by choosing the appropriate root predicate (see section 3.3). For the different levels of scenarios, we have to match a list of models with composed events in space and time. This list of queries follows possible scenarios (Table 7).

Table 7. Example of Queries

“Who and when someone did gym at home ?” Behave:Gym Subject: human_beings (*limited to Human*) Location: home (*limited to Home*) Date1: date time start Date2: now (*limited to before now*)
“Who and where someone did or will do gym in this time period?” Behave:Gym Subject: human_beings (*limited to Human*) Location: locations Date1: date time start Date2: date time end
“When and where John did gym in july 2010?” Behave:Gym Subject: John Location: locations Date1: 01/07/2010 0:00 Date2: 31/07/2010 23:59

6 Use Case of Multimodal Interaction

6.1 Implementation

We modeled a multi layers agents’ architecture for robotics multimodal interaction where agents have a semantic memory and an inference engine previously presented (Section 4.1). Thus, each agent has the ability to listen and perceive the environment (Figure 14).

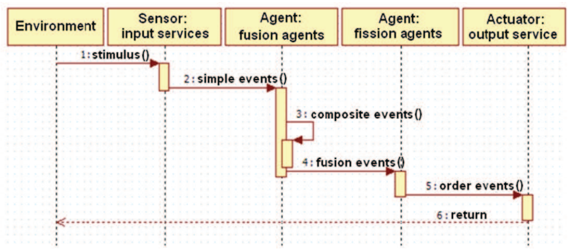


Figure 14. MAS Architecture for Interaction

Input Services drive hardware sensors, or any webservices in the network (pervasive and ambient aspects); they convert and send information to next layer agents. Output Services drive actuators connected to the environment. Fusion agents realize fusion process by composing higher level events (i.e. meaning of the situation) and send it to other agents of next abstraction layers. Fission agents are building lower level events (i.e. decision events or orders to execute) to send to output services using query models too.

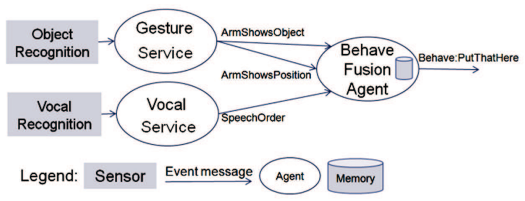


Figure 15. Fusion Agents

Table 8. "Behave:PutThatHere" Event model

Behave:PutThatHere
Subject: coord(ArmShowsObject, ArmShowsPosition, SpeechOrder)
Sender: coord(GestureServices, VocalServices)
Location: locations
Date1: date time start

Table 9. "Behave:PutThatHere"

Behave: PutThatHere
Subject: coord(ArmShowsObject, ArmShowsPosition, SpeechOrder)
Sender: coord(G1, G3, V1, V2)
Location: lab1
Date1: 09/10/2010 10:04

Figure 15 shows services and fusion agent in the famous interaction example of "Put That Here" [19]. Table 8 is a model event that represents "*a human giving an order and pointing an object and a location*". To build instance of this model, Behave fusion agent queries this model, and inference engine realize the composition of previously stored events matching it. Table 9 is one instance of happened event in a same time period and sent by 4 services embedded into the robot or belonging to the house. G1 and G3 are two services connected to two different sensors in charge to detect human gesture. V1 and V2 are also two services in charge of vocal recognition with two mikes. Behave fusion agent (BFA1 instance) can check if several recognition events match to help following decision system. The interpretation of events happening in environment is very simple and fast. The matching operation following a query will give a true description of the event. Fission agents take the incoming event(s) and, doing the same processes of matching and unification will produce order event(s) sent to drive actuator(s) depending on the models in their memory.

6.2 Experimentation

We need events in order to check multimodal fusion and fission processes. For the purpose of our experiment, we have implemented our semantic agents in JADE platform on a workstation. As today no hardware sensors and actuators communicate in EKRL, we embedded an EKRL concentrator on a Roboard[11] 101 card (Figure 16) connected in TCP-IP network and able to:

– simulate virtual sensors to generate perfectly controlled EKRL messages flow,

– convert input signals coming from actual sensors (connected to IP ports of the Roboard or via the networking interface) into EKRL messages,

– drive actual actuators (also connected to IP ports of the Roboard or via the networking interface) using EKRL output messages.

[11] http://www.roboard.com

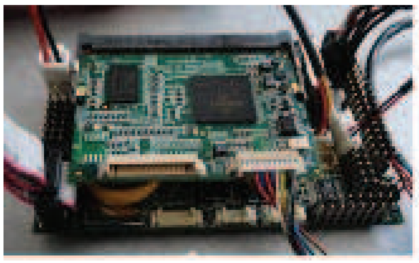
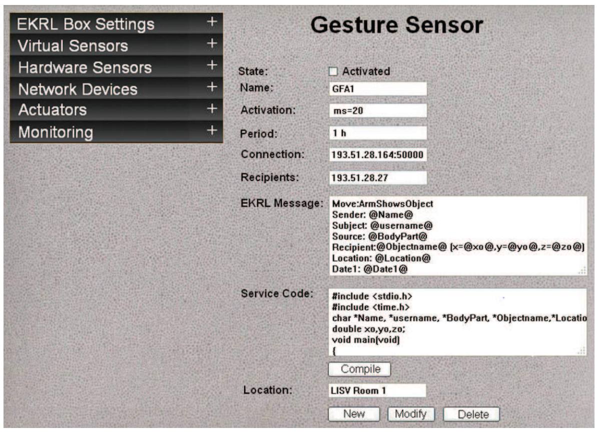


Figure 16. EKRL Concentrator Box



Gesture Sensor

State: ☐ Activated
Name: GFA1
Activation: ms=20
Period: 1 h
Connection: 193.51.28.164:50000
Recipients: 193.51.28.27

EKRL Message:
```
Move:ArmShowsObject
Sender: @Name@
Subject: @username@
Source: @BodyPart@
Recipient:@Objectname@ (x=@xo@,y=@yo@,z=@zo@)
Location: @Location@
Date1: @Date1@
```

Service Code:
```
#include <stdio.h>
#include <time.h>
char *Name, *username, *BodyPart, *Objectname,*Locatio
double xo,yo,zo;
void main(void)
{
```

Location: LISV Room 1

Figure 17. EKRL Concentrator Website

Our box has been built to generate or manage multimodal inputs/outputs in simulation, in virtual reality room and in real environments. It has a secured website (Figure 17) to set sensors and actuators drivers, to set the tasks scheduler, to write EKRL messages with free variables to send, and C programs to replace free variable of the EKRL message by the input data given by the sensor service

Gesture recognition sensors (G1 & G2 MS Kinect) and vocal recognition sensor's mikes (V1 & V2) are connected to EKRL concentrator. Gestures and vocal messages are read by the driver. Services converted data into EKRL messages and send them to fusion agents.

Test procedure (during 25 seconds) consists to:

– control the number of events sent to input queue of the agent,

– monitor agent's inference engine during test by looking at the target service queue containing composite events sent by the agent,

– check correct events found in the agent's memory after the test.

6.3 Results

We focused our analysis on ignored events compared to well-composed events in order to check generated meaning (i.e. consistency). After validation of event models, we check the good correlation between expected output and inputs. We also check the robustness of the inference engine with noisy events and by increasing the number of events in time (maximum load). Results are presented in Figure 18 and Table 10.

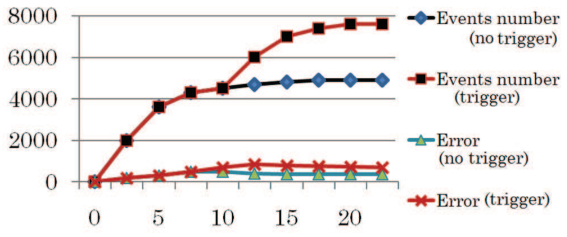


Figure 18. EKRL Concentrator Website

Table 10. Experimental Results

Events	Consistency	Robustness
Normal <1000 events/s	100%	100%
Overload >1000 events/s	71% 29% ignored	100%
Noisy 50% unknown events	50% 50% events ignored	100%
Noisy 50% false data in known events	42% 17% events not in time so ignored	96%

Consistency decreases when too much events occur. Performance of agents decreases due to their processing speed but robustness is always good because, in this human situation, events are very redundant so there is no impact on outputs. Agents are able to ignore events when they are not matching predicate name, time, location and other roles in event. Robustness remains good in case of noisy data without taking into account uncertainty measure coming from sensors and hence corrupted data.

Consistency is weak because false data in events impact correct events at 8%.

7 Conclusion & Future work

In this paper, we presented an EKRL able to model behaviors at different levels of abstraction in a semantic memory that can be integrated in multi agents systems. Modeling and understanding the environment for agents is easy to handle due to our ontologies structure. Our EKRL is suitable for multimodal interaction modeling, agent communication language, context aware application and intelligent Robotics applications. We will pursue this work in validating semantic architecture with behaviors coordination and temporal synchronization human daily activities monitoring, engineering and robotics applications.

References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach,* Prentice Hall, USA, 2003.

[2] D. Weyns, A. Omicini, and J. Odell, *Environment as a first class abstraction in multiagent systems,* In International Journal on Autonomous Agents and Multi-Agent Systems, Springer, 2007.

[3] B.F. Malle, *How the Mind Explains Behavior: Folk explanations, meaning, and social interaction,* MIT Press, Cambridge, 2004.

[4] Macal C.M. and North M.J, *Tutorial on agent-based modeling and simulation part 2: How to model with agents,* In Proceedings of the Winter Simulation Conference, L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds, 2006.

[5] Allan R., *Survey of Agent Based Modeling and Simulation Tools,* Technical Report: http://epubs.cclrc.ac.uk/work-details?w=50398, 2009.

[6] Gopal Sakarkar and Sachin Upadhye. *Article: A Survey of Software Agent and Ontology. International Journal of Computer Applications.* Foundation of Computer Science February 2010.

[7] A. Bessam and M.T. Kimour, *Multi-view Metamodeling of Software Architecture Behavior,* Journal of Software, 2009.

[8] M. P. Singh, *Agent communication languages: Rethinking the principles,* In Proceedings: Communications in Multiagent Systems, LNAI2650, M.-P.Huget (Ed), Springer, 2003.

[9] M. P. Singh, *A social semantics for agent communication languages,* In IJCAI Workshop on Agent Communication Languages, Springer-Verlag, Berlin, 2000.

[10] F. Guerin, *Specifying Agent Communication Languages*, PhD Thesis, 2002.

[11] G.P. Zarri. *Representation and Management of Narrative Information, Theoretical Principles and Implementation.* In: Series: Advanced Information and Knowledge Processing. Springer-Verlag, London, 2009.

[12] ODM/OMG: *Ontology Definition Metamodel,* Object Modeling Group, September 2008.

[13] OCL: *Object Constraint Language formal* v2.2, http://www.omg.org/spec/OCL/2.2/, February 2010. (Last release)

[14] D. Calvanese, J. Carroll, G.D. Giacomo, J. Hendler, I. Herman, B. Parsia, P.F. Patel-Schneider, A. Ruttenberg, U. Sattler and M. Schneider, *Owl2 web ontology language profiles,* 2009.

[15] F. Silva Parreiras, T. Walter, S. Staab, C. Saathoff, T. Franz, *APIs agogo: Automatic Generation of Ontology APIs*, In: Proceedings of the 3rd IEEE International conference on Semantic Computing, Santa Clara, 2009.

[16] Goumopoulos A. and Kameas A.,*A Service Oriented Architecture Combining Agents and Ontologies Towards Pervasive Adaptation*, 5th International Conference on Intelligent Environments (IE'09), Series: Ambient Intelligence and Smart Environments IOS Press, Spain, 2009.

[17] L. Obrst, *Ontologies for semantically Interoperable Systems, Conference on information and knowledge Management.* Proceedings of the twelfth international conference on Information and knowledge management, ACM Press, New York, NY, 2003.

[18] Blasch E., Kadar I., Salerno J., Kokar M.M., Das S., Powell G.M., Orkill D.D., Ruspini E.H, *Issues and Challenges in Situation Assessment* (Level 2 Fusion), In Journal Of Advances In Information Fusion, 2006.

[19] R.A. Bolt, *Put That Here: Voice and gesture at the graphics interface*, Proceeding ACM SIGGRAPH '80 Proceedings of the 7th annual conference on Computer graphics and interactive techniques, 1980.