Extrospection: Agents Reasoning About the Environment

Daghan L. Acay, Philippe Pasquier, Liz Sonenberg

The University of Melbourne, Department of Information Systems

Abstract

Current practices for agent-infrastructure interaction enforce agent designers to hard code the name and the use of the infrastructure components in the agent. Therefore, agents can only function within an environment which is a priori known to the agent designer. Even in these environments, agents are not robust against the modification or failure of the infrastructures that may occur at run time. This makes agents fragile in the context of distributed, large scale, and complex environments. We argue that semantic technologies can be employed to overcome these limitations. This paper introduces an ontology called OWL-T that supports the specifications of infrastructure components that allow agents to reason about the functionality and the use at run time. We use the term extrospection for the reasoning done by the agents for the discovery and the use of the infrastructure components based on the tasks and the goals of the agent. A proof of concept implementation illustrates the use of OWL-T in a multi-agent foraging scenario.

1 Introduction

Agents in a multi-agent system (MAS) not only interact with one another but also interact with the infrastructure in their environment. Neither the set of agents nor the infrastructure in the environment should be assumed static. That is, various agents can join or leave the environment and various infrastructure components can be uploaded or modified at run time. Although, research on agent communication languages (ACL) [1] studied agent-agent interaction at the semantic level to cater for dynamism and heterogeneity, agent-infrastructure interaction still does not enjoy semantic interoperability. Thus, agents that are designed to execute in an environment with a fixed infrastructure can not adapt to the run time changes to the infrastructure components and simply fail to accomplish their task.

The aforementioned difficulties arise from the fact that the interaction between agents and the infrastructure components is usually defined through interfaces with no associated semantics [17, 13]. Such interfaces do not allow agents to reason about the purpose or the use procedure of the infrastructure components. Weyns et. al. [19] identify semantic interoperability as a future challenge at the protocol level and continue:

> "The issue of protocol governs the degree to which an environment is open to heterogeneous agents, or to agents that are designed without advance knowledge of the environment" [19, p. 39]

Infrastructure is defined as the set of components and services that help agents to transparently access the resources [19]. Further more, we use the term *tool* for the services that are relevant to agent's task. For example, a printer driver is a service and it becomes tool for an agent if the agent somehow needs to print a document to reach a goal. In this paper we will use the term infrastructure component and the service interchangeably.

Omicini et. al. [11] postulated that agents should be able to discover and use services that are appealing for their task at run time. They elaborate on five levels with increasing environment awareness on behalf of the agent. The work demonstrated in this paper corresponds to level four of their taxonomy which is defined as: agents capable of autonomously selecting and using the services in the environment. We employ semantic technologies for building an information model that can be interpreted by the agents for this purpose. We also define an unambiguous formal semantic (what the service can be used for), and pragmatic (how the service can be used) for such an information model. We propose OWL-T (OWL Tool), an ontology for describing infrastructure components in the environment.

The name is the result of combining OWL (Web Ontology Language) as the ontology language used and the concept of "tool" that is elaborated in the Activity Theory (AT) [2]. The aliCE research group at Università di Bologna [6] has already sketched the relation between AT [10] and MAS in various papers [15, 11]. As an extension, we use the term *extrospection* "observation or examination of things external to the self; examination and study of externals"[1] which is complementary to the term *introspection* "an examination of one's own thoughts an feelings."[2]

OWL-T advances the state of the art by introducing a semantic framework for agent-infrastructure interaction. OWL-T is the initial step towards a formal background for studying extrospection and to identify the effect of the environment over deliberation. OWL-T is also aimed to resolve the practical difficulties that arise in the context of large scale open multi-agent systems.

The paper is organized as follows. Section 2 describes a MAS architecture that will be built on, in Sec. 3 where the details of OWL-T are defined. Later, Sec. 5 describes a scenario where OWL-T is used. Finally, Sec. 6 summarizes the paper and elaborates on future directions.

2 An Architecture for MAS

According to our view, a MAS comprises agents and the environment as shown in Fig. 1. The environment further consists a set of infrastructure components and the information about them. In principle, agents and services in the environment are assumed to be developed and maintained by different parties. This view is also consistent with the

[1] Currently, the term extrospection is not in the general English dictionary. We have adopted the definition given at http://dictionary.reference.com/browse/extrospection

[2] On-line Merriam-Webster dictionary http://www.m-w.com/dictionary/introspection

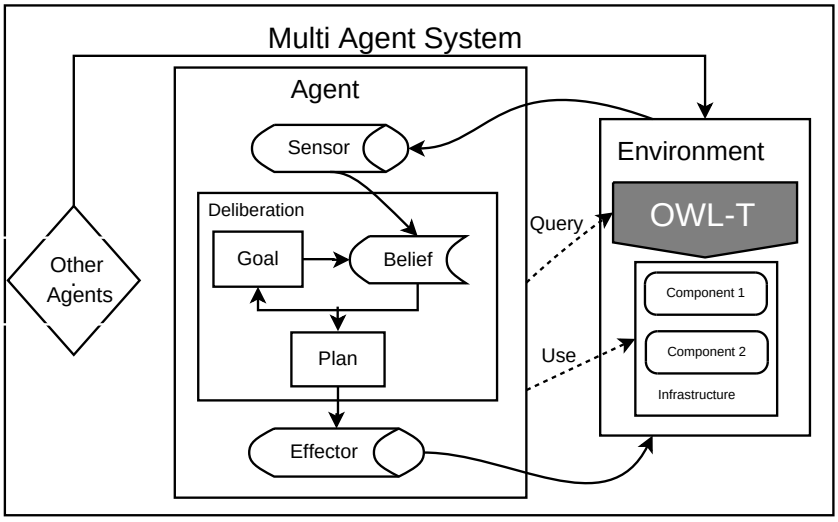

Figure 1: MAS architecture

service oriented computing [7] where the services and the service users are developed independently. The *query* link in Fig. 1 enables agent to discover the services through OWL-T. The *use* link orchestrate the exploitation of the selected service. Formally, a MAS $\mathcal{S}$ is defined as a tuple:

$$\mathcal{S} = \langle \mathcal{A}^+, \mathcal{E} \rangle \tag{1}$$

where: $\mathcal{A}^+$ is the non empty set of agents and $\mathcal{E}$ is the environment defined for $\mathcal{S}$. Models for $\mathcal{E}$ and $\mathcal{A}$ are given next.

2.1 Environment Model

The environment is a container for the infrastructure components and the information about those. Infrastructure components are concrete non-agent entities that can be invoked by the agents. The environment also actively maintains its own dynamics through its transition function [18]. The model for $\mathcal{E}$ is given as follows:

$$\mathcal{E} = \langle \mathcal{I}^*, \mathcal{O}, \mathfrak{T} \rangle \tag{2}$$

where $\mathcal{I}^*$ is the set of infrastructure components that reside inside the environment. $\mathcal{O}$ is the information model that formally defines the functionality and the use procedure of the services in the environment. The details of our information model, OWL-T, will be given in Sec. 3. The next two sections will present the infrastructure component model $\mathcal{I}$ and the transition function $\mathfrak{T}$.

2.1.1 Infrastructure Component Model

The initial assumption for the services in the environment is that they are not intentional, goal oriented, or autonomous as the agents are; in short, they are *function oriented* [18, 14]. The relation between the agent and the service is unidirectional and based on the *use* of the service by the agent. At the implementation level, the infrastructure component is actually assumed to be a computational object with its own properties and methods. A model that satisfies these assumptions can be given as follows:

Infrastructure component has a name: Each infrastructure component is uni-quely identified by an *id* drawn from the set inf. The alphabet for the infrastructure uses k, l.

Infrastructure component implements methods: The set of all the methods that are implemented by an infrastructure component is defined as the power set of constant symbols.

$$MS \stackrel{def}{=} \mathscr{P}(const)$$

Methods have input and output arguments: We define the input and the output arguments as:

$$IS \stackrel{def}{=} \mathscr{P}(const)$$
$$OS \stackrel{def}{=} \mathscr{P}(const)$$

So, a infrastructure component is a tuple:

$$\mathcal{I} = \langle \mathcal{M}, \mathcal{INP}, \mathcal{OUT} \rangle \tag{3}$$

where $\mathcal{M} \in MS$ is the set of methods that are implemented by the infrastructure component. $\mathcal{INP} \in IS$ (input set) and $\mathcal{OUT} \in OS$ (output set) are the sets of input and output arguments that the infrastructure component possesses due to the methods it implements.

2.1.2 Transition Function

We assume that there is an order for executing the methods over the infrastructure component. That is, methods can not be randomly invoked but should follow a sequence in order to guarantee the correct execution. We call Σ the set of all those sequences and σ a member of this set. We can visualize σ as follows:

$$\sigma : s_0 \stackrel{exec}{\longrightarrow} s_1 \stackrel{exec}{\longrightarrow} s_2 \stackrel{exec}{\longrightarrow} s_3 \ldots$$

where $\{s_0 \ldots s_n\} \in \mathcal{M}$ and $exec$ defines the transition that occurs when an agent executes a method. That is, every method invocation results in one step advance in σ. In our terminology, σ actually combines individual methods to accomplish an action, therefore we will refer σ as *the process*

model that realizes an action. The transition function $\mathfrak{T}$ is defined as follows:

$$\mathfrak{T} : \sigma \times \mathcal{M} \rightarrow \sigma \tag{4}$$

We will return to $\mathfrak{T}$ in Sec. 4 after we finish describing OWL-T. On the other hand, the overall environment model can be given by substituting (3) and (4) for (2) as follows:

$$\mathcal{E} = \{\langle \mathcal{M}_k, \mathcal{INP}_k, \mathcal{OUT}_k \rangle; \mathcal{O}; \mathfrak{T} : k \in inf\} \tag{5}$$

As an example of a service, let's consider a printer driver. We will call this driver the ***PrinterDriver1*** with two methods $\mathcal{M}_{PrinterDriver1}$ which are namely ***PrintDoc*** and ***Configure***. These methods have input arguments $\mathcal{INP}_k$ which are *document* and *configFile* respectively. No output arguments $\mathcal{OUT}_k$ are defined for the methods. Finally, transition function is defined as $\sigma_{PrinterDriver1} : Configure \xrightarrow{exec} PrintDoc$. The transition function identifies the order of the method call for correctly printing a document.

2.2 Agent Model

A minimal cognitive agent model that can interpret OWL-T is as follows:

Agent has a name: Each agent is uniquely identified by an agent *id* drawn from the set Ag.

Agent has beliefs: The belief set of an agent is the power set of propositions with no constructors and no quantifiers:

$$BS \stackrel{def}{=} \mathscr{P}(const)$$

Agent have goals: Goals of an agent are defined in the same lines with beliefs:

$$G \stackrel{def}{=} \mathscr{P}(const)$$

Agent has a role and involved in activities: Again, the activity and the role sets are defined in the same lines as beliefs:

$$\begin{aligned} RoleSet &\stackrel{def}{=} \mathscr{P}(const) \\ ActivitySet &\stackrel{def}{=} \mathscr{P}(const) \end{aligned}$$

Agent can perform actions: The set of actions Ac that the agent can perform is the union of the set of internal actions Aa that operate on the agent's mental structure and the actions supported by the infrastructure component $Ai \subset \Sigma$ (see Sec. 2.1.2):

$$Ac \stackrel{def}{=} Aa \cup Ai$$

Then, an agent is a tuple:

$$\langle \mathcal{B}, \mathcal{G}, \mathfrak{a}, \mathfrak{Q}, \mathfrak{u}, \mathfrak{N}^B, \mathfrak{N}^G \rangle \tag{6}$$

where:
$\mathcal{B} \in BS$ is the agent's initial belief set.
$\mathcal{G} \in G$ is the agent's initial goal set.
$\mathfrak{a} : G \rightarrow Aa$ is the agent's private action generation function.
$\mathfrak{Q} : G \times Ag^{role} \times Ag^{activity} \rightarrow inf$ is the infrastructure query function that returns the possible components. The returned components support the goal, role and the activity of the agent. In that, $Ag^{role} \in RoleSet$ and $Ag^{activity} \in ActivitySet$
$\mathfrak{u} : inf \rightarrow Mi$ is the agent's service use function. Mi is the subset of all the methods that is supported by the service and formally defined as $Mi \subset \bigcup_{i=0\ldots k} \mathcal{M}_k$.
$\mathfrak{N}^B : BS \times Mi \rightarrow BS$ is the agent's belief next state function.
$\mathfrak{N}^G : G \times Ac \rightarrow G$ is the agent's goal next state function.
A multi agent system is given by substituting (6) and (5) for (1):

$$\begin{aligned} \mathcal{S} = \{&\langle \mathfrak{B}_i, \mathfrak{G}_i, \mathfrak{a}_i, \mathfrak{Q}_i, \mathfrak{u}_i, \mathfrak{N}_i^B, \mathfrak{N}_i^G \rangle; \\ &\langle \langle \mathcal{M}_k, \mathcal{INP}_k, \mathcal{OUT}_k \rangle, \mathcal{O}, \mathfrak{T} \rangle : i \in Ag, k \in inf\} \end{aligned} \tag{7}$$

Assume a hypothetical agent with the initial belief $\mathcal{B}$ of *hasDocument(doc1)* and the initial goal $\mathcal{G}$ of *Print(doc1)*. Additionally, assume that our agent is involved in the activity $Ag^{activity}$ called *WordProcessing*. The role Ag^{role} assigned to our agent is *PrintAssistant* (along with other possible roles). One expects that, with the above information, our agent should select the useful infrastructure component and invoke the necessary methods. Recall the printer server we have introduced previously, what if in the next version of the *PrintServer1* we change the method name from *PrintDoc* to *PrintDocument*. Moreover, what if a new server *PrintServer2* is introduced after the agent is deployed. Next section will elaborate on how OWL-T can be used by the agent to handle these kind of situations.

3 OWL-T: A Tool Ontology

The information that OWL-T captures is described by Gibson as follows:

> "... If you know what can be done with a graspable object, what it can be used for, you can call it whatever you please. ... The theory of affordances rescues us from the philosophical muddle of assuming fixed classes of objects, each defined by its common features and then given a name. ... But this does not mean you cannot learn how to use things and perceive their uses. You do not have to classify and label things in order to perceive what they afford." [4, p. 134]

Two approaches for describing the information in the environment can be extracted from the definition above.

Epistemic: This approach aims to classify and label things and relations between them. For example, a *table* has four *legs*, with some *height* and some *color*. The main reasoning task is to determine the semantically similar objects in the agent's knowledge base or in the external knowledge bases generally through some logic inferences.

Pragmatic: This approach is similar to the *affordance* that defines the functional nature of an object independent from its physical properties. For example, a *table* affords putting objects on it, so does a *computer*, although they are by no means similar in their physical

properties and may not be related if the epistemic approach to the information modeling is employed.

The epistemic and the pragmatic approaches to the information modeling are complementary [14] and OWL-T is built with the pragmatic perspective in mind.

Web Ontology Language (OWL) is the selected ontology language for constructing OWL-T. The three species of OWL in the order of decreasing expressiveness are: OWL-Full, OWL-DL and OWL-Lite [8]. OWL-T is built using OWL-DL which in turn is based on Description Logic (DL) [3]. In this section, we will briefly introduce DL and its relation to OWL-T.

OWL-DL is chosen for various reasons. First of all, the formal semantic of DL and well researched inference algorithms [3, Chapter 2] enable agents to execute logic query over the information model. Secondly, efficient "complete" and "sound" algorithms for DL have practical outcomes such as the decidability of the inference. The decidability makes DL suitable for the computer programs without sacrificing much from the expressiveness [3].

Any ontology that is written using DL is composed of two types of axioms. The first type of axioms is the combination of two sets of axioms namely the concept axioms (denoted with C) and the role axioms (denoted with R). The collection of C and R axioms are labeled as the *terminology* and stored in the "TBox (Terminology Box)."

The second type of axioms are employed to relate the concrete objects in the domain to the terminology. Take for example the *Tool* concept C defined in OWL-T(Table 1). Then, *Tool(PrinterDriver1)* is an axiom for the concrete individual called *PrinterDriver1*. *PrinterDriver1* in turn is called the "instance" of the *Tool* concept. The convention for DL inference engines is to call "ABox (Assertion Box)" to the collection of axioms about the individuals.

In DL, the semantic of T-Box and A-Box is given in the model-theoretic form. Model theory[3] is an interpretation of Tarski[4] semantic where the set theory is chosen as the meta language $\mathfrak{M}$ to define the truth values of the sentences in the language $\mathscr{L}$. An interpretation $\mathfrak{I} = \langle \triangle^I, I \rangle$ for the language $\mathscr{L}$ is defined as the combination of:

1. A non-empty set $\triangle^I$, a.k.a. domain of interpretation
2. An interpretation function I which assigns
   - every atomic concept C to a set $C^I \subseteq \triangle^I$
   - every atomic role R to a set of binary relations $R^I \subseteq \triangle^I \times \triangle^I$

For an extensive analysis of DL and its semantic, we refer the reader to [3].

3.1 OWL-T Terminology

The corresponding axioms for C and R are given in Table 1 and 2 respectively. Quantifiers in the Table 1 should be interpreted as follows: $\exists_{=1} R.C$ means that there should be one and only one role R that exists between the concepts of C. Combination of $\exists R.C \sqcup \forall R.C$ means that there exists at least one or more relations between the concepts of C. Other quantifiers such as $\exists, \forall,$ and $\neg$ have usual meanings. Concepts that are derived from the axioms of Table 1 are interpreted by mapping the constructors of DL on to the set theory axioms. For example, the semantic for the complex concept $AbstractConcept$ (line 14 in Table 1) can be defined as $\mathfrak{I} \models AbstractConcept(x)$ iff $x \in AbstractConcept^I \subseteq \triangle^I$ where $AbstractConcept^I$ is the set defined by the intersection of the sets $IdealProperty^I \cup Interface^I \cup PhysicalProperty^I$ and the complement of $ActionModel^I \cup AgentModel^I \cup ObjectModel^I \cup OrganizationModel^I$.

OWL-T is organized under five concept categories which are mutually exclusive (lines $1, 5, 8, 11,$ and 14).

$ActionModel$**:** This category (lines 1–5) is used to delineate the activity hierarchy. The activity hierarchy represents three different types of the agent behavior. The reactive behavior of the agent is given by the $Process$ concept (line 4) which is used to define a single step of the $ProcessModel$. $Process$ is connected to the $Belief$ concept through the roles $assumption$ and $effect$. The $Action$ concept represents the goal oriented agent behavior. The connection between $Action$ and $Goal$ is established through $useCondition$ and $useResult$ roles. The $Action$ concept (line 3) is also linked to the $ProcessModel$ through the role $hasProcessModel$. That is, $Action$ is not atomic and may need a sequence of reactive behavior defined by the $ProcessModel$ before completion. The $Action$ concept also defines the $Artifact$ that is the input to the action and transformed after the $Action$ completion. Finally, the concept of $Activity$ (line 2) relates the goals of distinct $Action$s to a group activity. That is, $Activity$ is a wrapper which encapsulates $Action$s of the otherwise independent agents and defines the system level goals. The $Artifact$ which is shared and transformed by different $Action$s allow us to relate the actions of different agents. $Activity$ also describes the $Role$s via the role $actorsInvolved$ that may take part in the $Activity$.

$ObjectModel$**:** This category (lines 5–7) construes the objects in the environment. $Artifact$ (line 6) is an $Object$ which is transformed through the $Activity$ and the relation is established via roles $objectInvolved$ and $output$ (line 2). The second kind of object in the environment is $Tool$ (line 7) with two properties: $PhysicalProperty$ inherited from the $ObjectModel$ and $IdealProperty$. Any $Tool$ relates its functionality and use procedure through its $IdealProperty$. $Tool$ and $IdealProperty$ are combined using the role $affords$.

$OrganizationModel$**:** The category (lines 8–10) is aimed to put in place the access policies for a particular $Tool$ through $Role$ and $Group$ definitions. $Group$ is $composedOf$ $Role$s (line 9). On the other hand, $Role$ concept (line 10) has a role called $hasGoal$ to further enhance the relation between $Role$ and the allowed $Action$s by the $Tool$ for an agent.

$AgentModel$**:** The $AgentModel$ (lines 11–13) is the union of $Belief$ (line 12), and $Goal$ (line 13).

$AbstractConcept$**:** Concepts (lines 14–18) are supplemental to the categories above. $PhysicalProperty$ (line 17) is included in the terminology to support the spatial and the shape characteristics of an $Object$. $PhysicalProperty$ describes the location of

[3] http://en.wikipedia.org/wiki/Model-theory

[4] http://plato.stanford.edu/entries/tarski-truth/

(1)	*ActionModel*	$\equiv$	*(Activity⊔Process⊔Action)⊓¬(AbstractConcept⊔AgentModel⊔ObjectModel⊔ OrganizationModel)*
(2)	*Activity*	$\sqsubseteq$	*ActionModel⊓∀output.Artifact⊓∃output.Artifact⊓∀objectInvolved.Artifact⊓ ∃objectInvolved.Artifact⊓∀actorsInvolved.(Role⊔Group)⊓ ∃actorsInvolved.(Role⊔Group)⊓¬(Process⊔Action)*
(3)	*Action*	$\sqsubseteq$	*ActionModel⊓∀actionParameters.Artifact⊓∃actionParameters.Artifact⊓ ∀useCondition.Goal⊓∀useResult.Goal⊓∀hasProcessModel.ProcessModel⊓ $\exists_{=1}$hasProcessModel.ProcessModel⊓¬(Activity⊔Process)*
(4)	*Process*	$\sqsubseteq$	*ActionModel⊓∀assumption.Belief⊓∀effect.Belief⊓¬(Activity⊔Action)*
(5)	*ObjectModel*	$\equiv$	*∃appears.PhysicalProperty⊓∀appears.PhysicalProperty⊓¬(AbstractConcept⊔ ActionModel⊔AgentModel⊔OrganizationModel)*
(6)	*Artifact*	$\sqsubseteq$	*Object⊓∀satisfies.Goal⊓∃satisfies.Goal⊓¬Tool*
(7)	*Tool*	$\sqsubseteq$	*Object⊓∀affords.IdealProperty⊓∃affords.IdealProperty⊓¬Artifact*
(8)	*OrganizationModel*	$\equiv$	*Group⊔Role⊓¬(AbstractConcept⊔ActionModel⊔AgentModel⊔ObjectModel)*
(9)	*Group*	$\sqsubseteq$	*OrganizationModel⊓∀composedOf.Role⊓∃composedOf.Role⊓¬Role*
(10)	*Role*	$\sqsubseteq$	*OrganizationModel⊓∀hasGoal.Goal⊓∃hasGoal.Goal⊓¬Group*
(11)	*AgentModel*	$\equiv$	*(Belief⊔Goal)⊓¬(AbstractConcept⊔ActionModel⊔ObjectModel⊔ OrganizationModel)*
(12)	*Belief*	$\sqsubseteq$	*AgentModel⊓¬(Goal)*
(13)	*Goal*	$\sqsubseteq$	*AgentModel⊓¬(Belief)*
(14)	*AbstractConcept*	$\equiv$	*(IdealProperty⊔Interface⊔PhysicalProperty)⊓¬(ActionModel⊔AgentModel⊔ ObjectModel⊔OrganizationModel)*
(15)	*IdealProperty*	$\sqsubseteq$	*AbstractConcept⊓∀hasInterface.Interface⊓∃hasInterface.Interface⊓ ∀usedInActivity.Activity⊓∃usedInActivity.Activity⊓¬(Interface⊔PhysicalProperty⊔ ProcessModel)*
(16)	*Interface*	$\sqsubseteq$	*AbstractConcept⊓∀forRole.Role⊓$\exists_{=1}$forRole.Role⊓∀supportedActions.Action⊓ ¬(IdealProperty⊔PhysicalProperty⊔ProcessModel)*
(17)	*PhysicalProperty*	$\sqsubseteq$	*AbstractConcept⊓$\exists_{=1}$hasShape.⊤⊓$\exists_{=1}$hasLocation.⊤⊓¬(IdealProperty⊔ Interface⊔ProcessModel)*
(18)	*ProcessModel*	$\sqsubseteq$	*AbstractConcept⊓∃composedOf.Process⊓∀composedOf.Process⊓¬(IdealProperty⊔ Interface⊔PhysicalProperty)*

Table 1: OWL-T concept terminology

an $Object$ with the role $hasLocation$ and the shape of it with $hasShape$. $IdealProperty$ (line 15) relates a $Tool$ to an $Activity$. That is, it defines under which $Activity$ the $Tool$ is useful. It also describes an $Interface$ through role $hasInterface$. $Interface$ (line 16) formalizes supported $Action$s for a given $Role$ via the role $supportedActions$. Supported actions cater for the different uses of the $Tool$. For example, a printer can be used to print a document by an agent. Yet, another agent may use the printer to send a failure note to a user. A $Tool$ does not allow a random $Process$s to be invoked by the agents in order to ensure the correct operation of the infrastructure component. The concept of $ProcessModel$ (line 18) articulates the order of the $Process$ and is $composedOf$ a set of $Process$es with an ordering relation.

usedInActivity	*hasInterface*	*affords*
satisfies	*actionParameters*	*composedOf*
appears	*objectInvolved*	*actorsInvolved*
output	*hasGoal*	*processModel*
useCondition	*useProcedure*	*forRole*
assumption	*effect*	

Table 2: OWL-T role terminology

3.2 Semantic of OWL-T

The semantic of the concepts in Table 1 is based on specifying the domain of interpretation $\triangle^I$. The interpretation $\mathfrak{I}$ of the $Tool$ concept is the set inf given in Sec. 2.1.1. Formally given as follows:

$$\mathfrak{I} \models Tool(x) \text{ iff } x \in inf$$

The relation between OWL-T and the agent model given in Sec. 2.2 is as follows:

$$\mathfrak{I} \models Belief(x_i) \text{ iff } i \in Ag \wedge x \in \mathfrak{B}_i \Rightarrow x \in Belief^I$$
$$\mathfrak{I} \models Goal(x_i) \text{ iff } i \in Ag \wedge x \in \mathfrak{G}_i \Rightarrow x \in Goal^I$$

That is, for every constant in the belief or the goal sets of the agent, there is a constant in the domain of interpretation $\triangle^I$ that is a member of the sets $Belief^I$ or $Goal^I$ respectively. Furthermore, $Goal^I \in G$ formulates the semantic relation between the agent goal set and an interpretation of the concept $Goal$. Moreover, $Belief^I \in IS \cup OS$ defines that the agent beliefs are used as the input to the method invocation and the method outputs are used as the belief updates for the agent.
The interpretation of the $Process$ concept is the union of all the methods implemented by the services in the environment. Formally, the semantic of $Process$ is given by:

$$\mathfrak{I} \models Process(x) \text{ iff } x \in \bigcup_{k=0\ldots n} \mathcal{M}_k \wedge k \in inf$$

The interpretation of $Action$ is a set of constants. Finally, concepts $Activity$ and $Role$ are given as follows:

$$\begin{aligned} \mathfrak{I} &\models Activity(x) \text{ iff } x \in ActivitySet \\ \mathfrak{I} &\models Role(x) \text{ iff } x \in RoleSet \end{aligned}$$

4 Pragmatic of OWL-T: Extrospection

In the context of agent communication, Werner [16, p.64] defined the pragmatic as the transformation of the agent mental state after receiving a message. In the context of the agent-infrastructure interaction, the transformation takes place after two previous phases. In the first phase, the agent queries OWL-T for the available services that are usable to reach its goal. In the second phase, the agent will query the current state of the $ProcessModel$. That is, the agent retrieves the method that is allowed by the service. Finally, the agent's mental state is updated after the $Action$ and $Process$ are successfully executed. The extrospection covers all three steps and enable agents to discover and use the service to reach their goals. Next three sections will detail the three steps.

4.1 Infrastructure Query

The infrastructure query $\mathfrak{Q}$ in (7) returns the infrastructure components that can be used by the agent. There are three preconditions to find service that the agent can use. First, the agent should have access permission to use the selected service with respect to its $Role$ definition. Second, the agent should be involved in an $Activity$ for which this service can provide support. Third, the agent's goal should overlap with the $Goal$s defined for the $Action$s that are supported by the service. We do not consider the case where multiple candidate services can be returned returned by the query. Agent's service selection function is one of the future directions. Query function can be captured formally as follows:

$$\mathfrak{Q}_i(\delta, \rho, \lambda) = E_k \tag{8}$$

where: $\delta, \rho,$ and λ satisfy $\mathfrak{I} \models$ *(E_k.affords ∘ hasInterface ∘ forRole).Role(ρ) ∧ (E_k.affords ∘ usedInActivity).Activity(λ) ∧ (E_k.affords ∘ hasInterface ∘ supportedActions ∘ useCondition).Goal(δ).* The operator ∘ enables us to drop the unnecessary concept names while traversing the ontology. Above formula can be expressed with the omitted concept names added in parenthesis. E_k is the ($Tool$) which $affords$ an ($IdealProperty$) which $hasInterface$ ($Interface$) $forRole$ $Role$ ρ and the same ($Tool$) E_k $affords$ an ($IdealProperty$) which defines the $usedInActivity$ for the $Activity$ δ and again the same ($Tool$) E_k $affords$ an ($IdealProperty$) which $hasInterface$ ($Interface$) that defines $supportedActions$ of the form ($Action$) whose $useCondition$ is $Goal$ δ.

4.2 Infrastructure Use

As soon as the agent discovers the infrastructure component that it can use via the query given by (8), it can invoke a particular $Process$ over the infrastructure component as long as it has the necessary $Belief$s defined by the $assumption$ of the $Process$. The function $\mathfrak{u}$ in (6) is defined as:

$$\mathfrak{u}_i(E_k) = \alpha \tag{9}$$

where: α satisfies $\forall$ *i,k.* $i \in Ag \wedge k \in \mathfrak{Q}_i$ and $\mathfrak{I} \models$ *(E_k.affords ∘ hasInterface ∘ supportedActions ∘ hasProcessModel ∘ composedOf).Process(α)* and *(α.assumption).Belief* $\subseteq \mathfrak{B}_i$. The semantic of the operator ∘ is the same as above.
Using (8) and (9), agents can enhance their capabilities through extrospection. $Process$ invocation over the infrastructure component has two effects: first on the component side and second on the agent side. The effect on the infrastructure component is to advance the process model one step according to the transition function $\mathfrak{T}$ as given in Sec. 2.1.2. What we define here is the relation between the concept $ProcessModel$ and $\mathfrak{T}$.

$$\mathfrak{I} \models ProcessModel(x) \text{ iff } x \in \Sigma$$

The effect of the $Process$ invocation over the agent mental state is described next.

4.3 Mental effect

Using the infrastructure component has two effects. The agent goal set is updated after the $Action$ is accomplished, whereas the agent belief set is updated after every $Process$ invocation. For example, assume an $Action$ which is composed of a sequence of five $Process$es. In this case, beliefs $\mathfrak{B}_i$ of the agent i will be updated five times and the goals $\mathfrak{G}_i$ the agent i will be updated only once at the end of the execution of $ProcessModel$. Thus, the belief and goal transition functions $\mathfrak{N}^B$ and $\mathfrak{N}^G$ of the agent are given by:

$$\begin{aligned} \mathfrak{N}_i^G(\mathfrak{G}_i, \gamma) &= \delta \\ \mathfrak{N}_i^B(\mathfrak{B}_i, \rho) &= \psi \end{aligned} \tag{10}$$

For the first entry in the equation, we define $\gamma \in \mathfrak{Q}_i$ and $\mathfrak{I} \models$ *(γ.affords ∘ hasInterface ∘ supportedActions ∘ useResult).Goal(δ).* For the second entry, we define $\mathfrak{I} \models$ *(γ.affords ∘ hasInterface ∘ supportedActions ∘ hasProcessModel).Process(ρ) ∧ (ρ.effect).Belief(ψ).* Overall, (8), (9), and (10) define the pragmatic of infrastructure component use.
Recall our hypothetical agent with the initial belief $\mathcal{B}$ of *hasDocument(doc1)* and the initial goal $\mathcal{G}$ of *Print(doc1).* If we assert the facts *Tool(PrintServer1)*, *Action(Printing)*, *Process(Configure)*, *Process(PrintDoc)*, *Goal(Print(doc1))* in OWL-T, agents can discover and use the necessary infrastructure component only by querying OWL-T. If one changes the name of the service or add a new service, all he needs to do is to assert new facts in OWL-T. There is no need to change the agent code since the next query will return the updated information. Next section will give a more detailed example.

5 Example: Foraging Agents

The implementation presented here should be seen as a proof of concept and it does not employ all the aspects introduced by OWL-T. However, we see it as a good starting point to explain the use of OWL-T. Our aim is to show that the dynamic composition of the infrastructure can be handled without any modification to the agent code.
We use the Java programming language for this implementation. Ontology is built using Protégé software [9]. Repast[5] is chosen as the simulation environment. Runtime query is implemented using Jena [5] reasoning engine.

[5] http://repast.sourceforge.net/

$\mathfrak{B}_i^0$	=	{XLocation(x),YLocation(y)} The agent's initial location on the grid environment
$\mathfrak{G}_i^0$	=	{Food} The agent initially search for food
$\mathfrak{a}_i$	=	Uniform random distribution to use internal action or infrastructure component action
$\mathfrak{Q}_i$	=	Jena implementation for querying OWL-T
$\mathfrak{u}_i$	=	Java RMI implementation
$\mathfrak{N}_i^B$	=	{XLocation(x),YLocation(y)} that belong to resulting grid after the agent moves
$\mathfrak{N}_i^G$	=	{Food} when the agent is at the nest location $\vee$ {nest} when the agent is at the food location

Table 3: The agent implementation

Runtime invocation and the service use is handled by the RMI (Remote Method Invocation) packed with Java VM (Virtual Machine).

5.1 The Scenario

A set of agents need to find the shortest path between their nest and the food source. We have implemented a grid world environment of the size 25×25 using Repast. Two infrastructure components, namely $Pheromone$ and $Signpost$ were developed. Agents were implemented as Java threads and distributed randomly in the grid world at the initialization. The location of the food and the nest was fixed in every simulation.

5.2 Agents

The agent name set is $Ag = \{ag_1, ag_2, ...ag_n\}$. The belief set is defined as $BS = \{XLocation(x), YLocation(y)\}$ that refers to the location of the agent in the grid world. The goal set of every agent is defined as $G = \{Food, Nest\}$ which specifies the goal for reaching the food location and the nest location respectively. Agents are associated with the $Role = \{foragingAgent\}$ and the $Activity = \{foraging\}$. Agents themselves are capable of moving randomly and have an internally defined action $Aa = \{MoveRandom\}$.
No direct reference to the infrastructure components and their use is given in the agent code. The agents are only employed with the query function to query the environment for the infrastructure components that can be used during $foraging$. The information about both the functionality and the use procedure of the infrastructure components are embedded in OWL-T so agents can retrieve which $Actions$ are supported and which $Process$es can be executed over the infrastructure component. Therefore, agents can interact with the infrastructure without a priori knowledge about the environment.
Table 3 defines the initial state of the agent when the simulation starts and the belief and goal transforming functions that manipulate the initial states as the agent continues its $foraging\ Activity$.

5.3 Infrastructure Components

The first infrastructure component that can be exploited by the agents to reach their goals is called $Pheromone$. Dynamic of $Pheromone$ is given by Parunak in [12]. At every simulation step, agents can use $Pheromone$ or make a random move to determine their position in the next step of the simulation. The environment, on the other hand, is responsible for the pheromone diffusion and evaporation which are independent from the actions of the agents.
The second infrastructure component is called $SignPost$ that points to the neighbouring grid closer to the nest location if the agent has the $Goal\ Nest$. Otherwise, the sign shows the neighbouring grid that is closer to the food if the agent has the $Goal\ Food$.
Concrete infrastructure is implemented in Java. The $Pheromone$ corresponds to the component "Pheromone.java" and the $SignPost$ has the corresponding class file "SignPost.java." They both have $usedInActivity$ being $foraging$. Both components are intially defined for $Role = \{dummy\}$.
Methods, $\mathcal{M}_{pheromone} = \{$*FollownestPheromone, FollowFoodPheromone*$\}$, are implemented by Pheromone.java. Similarly, SignPost.java implements methods $\mathcal{M}_{signpost} = \{$*FollownestSign, FollowFoodSign*$\}$. $Actions\ A_i =$ *Goingnest, GoingFood* are defined in OWL-T to relate the methods above to the goals $nest$ and $Food$ respectively.

5.4 Simulation Results

Simulation was done through three consequent phases. First, neither $SignPost$ nor $Pheromone$ were accessible to the agent since we set the $forRole$ entry of both infrastructure components to $dummy$. As a result, when the agents with $foragingAgent\ Role$ queried OWL-T none of the infrastructure components were returned to the agents. Under this circumstance, the only action that agents could execute was the $RandomMove$. Later in the second phase, we set the $forRole$ entry of the $Pheromone$ to $foragingAgent$. Agents started to exploit the $Pheromone$ and their behavior was changed from the random move to the pheromone following. In the third phase, $SignPost$ was made available instead of $Pheromone$ by following the same procedure.
Changes were made at the run time and the agent code was not changed. Since agents were querying the available infrastructure components and their use procedure through OWL-T, we only updated OWL-T as described above. An agent that retrieve the name and the use procedure of the infrastructure component invoked the methods via RMI. Thus, a direct reference to the infrastructure component was not necessary and the agents could discover and use the infrastructure component at the run time.

6 Conclusion and Future Directions

Semantic interoperability lets agents to reason about the functionality and the use of the infrastructure, hence agents can better adapt to the changing components. Since OWL-T is function oriented (pragmatic), it enables agent to discover and use infrastructure components without any a priori knowledge of them. Thus, OWL-T supports extrospection which is defined as "observation or examination of

things external to the self; examination and study of externals."
We believe that the implications of extrospection are far reaching. One such implication is to control the agent behavior and the composition of the environment without modifying the agent code. For example, the environment designer may define the access policies for an infrastructure component, based on the $Role$ of an agent. The allowed agents may reach their goals in a way that has not been defined explicitly in its code. The environment engineer may also define how the infrastructure component can be used by modifying the $ProcessModel$. This results in an enhanced control over the behavior of the agents and their coordination.
As a future work, we plan to enhance the $ProcessModel$. Currently, we assume a sequential $ProcessModel$ with no branching, which limits the set of processes that can be expressed. The remedy to this drawback is readily available in the extensive process definition of OWL-S [7] and it may be the next step for OWL-T to adopt the process model of OWL-S. A second improvement would be to identify the means for choosing an infrastructure component in situations where there is more than one candidate. In that case, there should be an ordering relation over the infrastructure components so that agents can make a choice. An evaluation function of some sort will strengthen the link between the service use and the agent deliberation. We are also considering the work done by Ricci et. al. [13]. They brought fore the *artifact* abstraction, which corresponds to the entities in the environment other than agents. We believe that merging the artifact framework and the semantic interoperability will benefit us both terminologically and conceptually for developing the future MAS.

References

[1] Fipa communication act library specifications. Technical report, FIPA: Foundation for Intelligent Physical Agentsho, 2002.

[2] Erik Axel. One developmental line in european activity theory. In Michael Cole, Yrjö Engeström, and Olga A. Vasquez, editors, *Mind, culture, and activity: Seminal papers from the laboratory of comparative human cognition*, chapter 11, pages 128–146. Cambridge University Press, 1997.

[3] Franz Baader, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *Description Logic Handbook*. Cambridge University Press, 2002.

[4] James Jerome Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, 1979.

[5] http://jena.sourceforge.net/documentation.html. Jena OWL/RDFS API.

[6] http://www.alice.unibo.it:8080/aliCE/. aliCE research group.

[7] http://www.daml.org/services/owls/. OWL-S. WEB.

[8] http://www.w3.org/TR/owlref/. Web ontology language (OWL).

[9] Holger Knublauch, Matthew Horridge, Mark Musen, Alan Rector, Robert Stevens, Nick Drummond, Phil Lord, Natalya F. Noy, Julian Seidenberg, and Hai Wang. The Protege-OWL experience. In *Fourth International Semantic Web Conference (ISWC2005)*, Galway, Ireland, 2005.

[10] Alexei Nikolaevich Leontyev. *Activity, consciousness and personality*. Prentice Hall, Englewood Cliffs, NJ, 1978.

[11] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Agens Faber: Toward a theory of artefacts for MAS. *Electronic Notes in Theoretical Computer Sciences*, 150(3):21–36, 29 May 2006.

[12] H. Van Dyke Parunak. 'Go to the ant' : Engineering principles from natural agent systems. *Annals of Operations Research*, 75:69–101, 1997.

[13] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. CArtAgO: An infrastructure for engineering computational environments in MAS. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *3rd International Workshop "Environments for Multi-Agent Systems" (E4MAS 2006)*, pages 102–119, AAMAS 2006, Hakodate, Japan, 8 May 2006.

[14] Erich Rome, Patrick Doherty, Georg Dorffner, and Joachim Hertzberg, editors. *Towards Affordance-Based Robot Control*, Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

[15] Mirko Viroli, Andrea Omicini, and Alessandro Ricci. Engineering MAS environment with artifacts. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *2nd International Workshop Environments for Multi-Agent Systems , E4MAS 2005*, Utrecht, The Netherlands, 26 July 2005.

[16] Eric Werner. Logical foundations of distributed artificial intelligence. In Greg O'Hare and Nick Jennings, editors, *Foundations of distributed artificial intelligence*, pages 57–117. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[17] Danny Weyns and Tom Holvet. A reference architecture for situated multiagent systems. In *Environments for multi-agent systems*, 2006.

[18] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first-class abstraction in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, February 2007. Special Issue on Environments for Multi-agent Systems.

[19] Danny Weyns, H. Van Dyke Parunak, Fabien Michel, Tom Holvoet, and Jacques Ferber. Environments for multiagent systems: State-of-the-art and research challenges. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *First International Workshop on Environments for Multi-Agent Systems, E4MAS 2004*, pages 1–47, New York, NY, USA, 2004. Springer.