

Knowledge Request-Broker Architecture: A Platform for Social Computational Intelligence

Hamed Khandan and Takao Terano

Department of Computational Intelligence and System Sciences

Tokyo Institute of Technology, Japan

Email: hamed@trn.dis.titech.ac.jp, terano@dis.titech.ac.jp

Abstract

Computational systems and intelligent machines are facing with both compositional and environmental growing forms of complexity. We contribute a way to better deal with these forms of complexity by proposing a novel agent-based software platform: Knowledge request-Broker Architecture (KnoRBA). This solution aims at easily implementing self-organizing distributed software systems, mainly composed of socially interactive software agents. KnoRBA features a better way for heterogeneous agents to achieve mutual understanding: instead of specifying a protocol, it equips agents with programmatic means to provide knowledge to, and query it from the society. Utilization of KnoRBA is made possible by provision of appropriate language extensions and code libraries. This article, having reviewed the efficiencies and deficiencies of the existing literature, describes the basic principles, system architecture, and Java-based implementation of KnoRBA, as well as example case of a computational system made possible by it, called Social Regression.

Keywords: *Artificial Social Intelligence; Knowledge Request-Broker; Social Computational Intelligence; Agent Development Platform*

1: Introduction

“Why are there so many robots in fiction, but none in real life? I would pay a lot for a robot that could put away the dishes or run simple errands.”, says Steve Pinker in his book, “How the Mind Works” [16]. The reason why robots do not accompany human in real life, is that the real world environment is too complex for them to precept, understand, and deal with. This issue of *environment complexity* drives a vast number of research works done on artificial intelligence; yet, even major achievements are insignificant compared to the goal ahead. To name, there are about ten movements in artificial intelligence since 1980s, including neural, evolutionary, Bayesian, rule-based, fuzzy, and so on; but no single one of them could bring intelligence to the machines as it was desired.

In fact, any solution, no matter how good it is, reaches its performance limits at some point (according to No-Free-Lunch theorem [19]). It is discussed by scientists like Minsky [7] and Page [14], that not a single solution but a combination of vast number of them can be the key to achieve a better level of intelligence. Although Minsky and Page have different perspectives on the matter, they both agree on the approach of establishing an agent-based artificial society. The idea is, just as a team of specialists can achieve a better result than any single of them, and an organization works better than any of its specialist teams, and so on, by putting numerous and various solutions

inside specialist software agents and letting them to cooperate, better intelligence can be achieved and limits can be pushed farther.

Compositional complexity is another good reason, beside environment complexity, to peruse social approach to computational intelligence. The machines supposed to possess intelligence are becoming more decentralized, that is to say, being composed of numerous inter-networked processing units, rather than one. For example, in the field of ambient intelligence, the system consists of a network of numerous sensing and actuating, mobile and workstation devices that are supposed to act as an integrated intelligent entity. In such a system, parts and components may be changed, upgraded, or fail, and in all cases, the system must be flexible enough and smart enough to adapt itself to the new situation without losing its track. A social solution can prove efficient in this respect, because of two properties: (1) Structural adaptability, that is, the software can change its structure to adapt to the changes in the system; and (2) Redundancy: multiple solutions to the same problem can exist in the system at the same time, and if one could not function properly, for example, in an unexpected situation, the others can take over.

Despite numerous efforts, there are a number of problems yet to be properly addressed to make the development of such social systems possible, for which, this article intends to bring some contributions to the solution. These problems are (1) mutual understanding between heterogeneous agents, (2) architecture specification for general-purposed agents and their social system, and (3) API for development of social agent-based systems. These items shall be discussed in further details.

DARPA Knowledge Sharing Effort (KSE) [15] is one of the earliest movements to address the issue of mutual understanding between heterogeneous agents who utilize different approaches to precept and represent knowledge. KSE resulted in proposition of KQML (Knowledge Query and Manipulation Language) [2], and then FIPA-ACL (Agent Communication Language) [12], which by 2005, when FIPA merged into IEEE, and after, despite having a few followers like [17], was not well-received.

Generally, in order to make knowledge understandable to every one, it must be represented in a language that all understand, which may work for factual knowledge, but is simply impossible in the case of procedural knowledge. Procedural knowledge can be in many forms, like a program in any arbitrary language, in a neural network with its weights and connections, and so on. An effort to create a protocol that specifies representation of procedural knowledge either restricts it to an undesirable level, or does not serve the goal of mutual understanding. The former is the case with most cognitive architectures based on first-order-logic, and the latter is the case with KSE.

There is an alternative method to standardization of knowledge representation, that is, to only standardize the interface, and let the implementation to be in any arbitrary form. Given the fact that the same method have been widely received when applied to object-oriented paradigm as in Common Object Request-Broker Architecture (CORBA) [11], it is reasonable consider it as a better alternative. In the scheme proposed in this article, instead of providing a protocol, a standardized knowledge provision and query interface is provided to the developers through provision of an extension to programming languages and additional code libraries. Using these facilities, the user can invoke knowledge provided by all other software agents in the system, while protocol and network issues are taken care of by a module called the Knowledge Request-Broker (KRB) embedded inside each agent. KnORBA specifies detailed aspects of KRB.

Several architectures are proposed for intelligent agents. Cognitive architectures like Soar [6] and ACT-R [5] are made for specific type of intelligence, and therefore they are almost useless when other types of intelligence needed, and more useless when combination of different types of intelligence is needed. Newer efforts like JACK multi-agent system development platform [18] make

it possible to implement agents using an extension of a general-purposed programming language such as Java. This enables more freedom in implementing different solutions. However, JACK is merely based on BDI (Belief-Intention-Desire) model which may not be necessary for many types of intelligent agents. JACK does not provide strong means for knowledge provision and query. Plus that, even knowing Java, and agent-based paradigm, JACK extension is so extensive that makes it difficult for existing Java users to adopt its technology.

JACK is one of many agent-based system development platforms, most others like Repast, Repast Symphony [10], and SOARS [1], are mainly for agent-based simulation, rather than development of software agents. Those platform concerning with software agents, like Jini (Apache Rio Project), are mostly to provide networking and mobility facilities for agents rather than mutual understanding and intelligence. Hence, a suitable platform like KnoRBA seems to be required.

The idea of social software initiated long ago since the Society of Mind [7] was proposed. However, that theory, and the other ones in-line with it, like World-Wide-Mind project [13], and the Emotion Machine [8], do not very well explain how such a social system and its agents can be engineered. Proposition of KnoRBA is an attempt to provide methodology and tools for creation of social software.

The rest of this paper is organized as follows. In section 2, the concept of artificial society as used in this article is described. Section 3 depicts different aspects KnoRBA, including knowledge representation using relations, and agent architecture. Section 4 explains how to create agents and work with them using the current experimental implementation of KnoRBA for Java. In section 5, the Social Regression solution which is made possible by KnoRBA is given as a practical example. Finally, section 6 concludes the paper by reviewing the contributions in this work, pointing out strengths as well as deficiencies, and proposing some ideas for future works.

2: Society Concept

KnoRBA is created having a business-driven model for artificial society in mind, as shown in figure 1. A social software consists of a collection of interacting “specialist” agents. Each agent is a unit of cognition, meaning that, it represents cognition only as much as it is concerned with its specialty. For example, one agent uses CART algorithm for data mining, the other uses SVM or naïve Bayesian for the same reason, while another may have the algorithm to solve Hanoi Towers. Agents are like apps, i.e. small applications, they can be device drivers, controllers, GUIs, and so on. Thinking of agents as components of a program, unlike conventional objects, the way these agents form an integrated solution is not hard-coded, but is done by having them engaged in knowledge exchange with the others, through knowledge request or notification.

Since the social system is rather a soup of components than a set of rigid computer applications, an alternative concept to “application” should be defined. Instead of applications, there can be agents interfacing users with specific usage intension, with the rest of other agents. These agents are called “end-customers”, for the reason explained shortly. End-customers interact with other agents on behalf of the user, while those other agents may engage the others in turn, and so on. There can be multiple end-customers present in the system, and some agents may serve multiple customers at the same time. To model the architecture of this social system, analogy with “value chain” model in business is applicable (figure 1). Having this argument, three roles for agents can be imagined in such a business-driven architecture, namely, specialist, agency, and end-customer, as explained below. An agent can assume more than one of these roles.

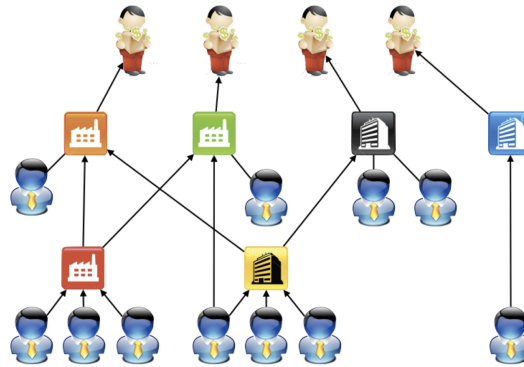


Figure 1. Three roles of agents in the business-driven model for KnoRBA-based artificial societies

Specialist Agents. In a sense, all agents in KnoRBA are specialist, meaning that, they can only carry out limited specific tasks, and can focus only on specific portion a problem. Some specialists are not concerned with the main problem at all and perform service tasks. For example GUI-Agent provides an interactive interface for the artificial world.

Agencies. Increase in population calls for organization and order. That is why individuals usually form and join organizations. Special type of agents called agencies is how these organizations appear in our model for artificial society. Agencies are like factories and corporations in the modern world. They register, manage and guide their member agents. To defined, an agency is a kind of specialist, with its product being the product of its members combined together in some way. Usually, an agency, has broader view on the problem-space than its member specialists, but its view is less detailed.

End-Customers. As described above, end-customers stand at the end of value-chain and represent the interest of the user in the system, making it possible for him/her to utilize the capabilities of the system.

2.1: Identity versus Functionality

In most computing platforms, entities are identified by their unique addresses or identifiers. Pointers, references, and identifiers in programming languages, and IP address in the Internet are some examples. However, some believe that the components should be identified by their functionality, rather than their identity. So-called “ADS” [9] and CORBA work based on this concept. In the real world, both methods are used: while people have identifiers, like names, or national IDs for addressing them, in business interactions their function is more concerned, e.g. barber, programmer, manager, and so on. In KnoRBA, although agents have globally unique addresses, most of transactions begin by addressing them by their functionality through knowledge inquiry. After knowing agents with desired functionality, an agent may optionally memorize the identity of some other agents for further interactions.

2.2: Potential Solvers

It is not the case that all intelligent agents know how to solve a problem upon their creation. In fact most implementations require a certain phase of learning, before the agent can actually answer any question. These agents who may be able to solve a problem, but only after training, are called *potential solvers*. Being a feature introduced in KnoRBA, it is possible to identify potential solvers, select proper ones and “award contract” to them to have them to participate in a specific part of a problem by training themselves accordingly.

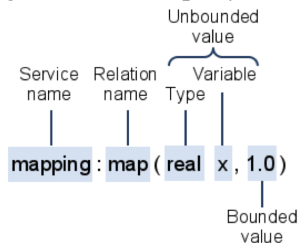
3: Architecture

3.1: Relations

Unlike objects which are collections of fields and methods (or functions), in KnoRBA relations are the basic elements. Relations are more general than fields and methods. A function (method) is a specific type of relation and variables (fields) can be represented as unary relations. Relations are considered because, in one hand they look very similar to functions (methods) which are common in almost all programming languages and can be defined in a very similar way, and in the other hand they can be used to encode and inquiry knowledge very efficiently. The group theory gives this confidence that almost everything can be expressed in terms of relations. KnoRBA makes it possible to use preexisting facilities in a host language to implement and manipulate relations. KRB includes a built-in inference engine that calls relations in a way that the agent can answer to the given queries.

Just like functions and methods in many programming languages, KnoRBA relations are defined with a signature assigned to each. For example, a relation that maps a real number to another one, can be defined as `map(real, real)`. Relations does not have return values, but they can be either true or false. The truth value should be determined by the implementation of a relation based on the arguments given to it. Like CORBA, KnoRBA has its own independent typing system which consists of seven primary types (integer, longint, real, string, boolean, agentid, and raw) and two complex types (record and recordset). The implementation of the “any” type is under investigation.

Relations are nested in packages called “services”. For example, `mapping:map(real, real)` is the well-qualified signature of `map()` relation, indicating that this relation is contained in “mapping” service. To query for a relation, the following notation can be used.



Unbounded values are those expected to be found by the implementation of the relation. Because KnoRBA is strongly typed, the type of the value should be specified.

Complex queries can also be made out of multiple relation invocations. For example, the following is supposed to make the receiver agents to find all x and y value pairs that can be mapped into each other, while x is between 0 and 1. In a KnoRBA query, comma means logical “and”, semicolon means logical “or” (not used here), and the query ends with a question mark. For the grammar of KnoRBA query language, see appendix A.

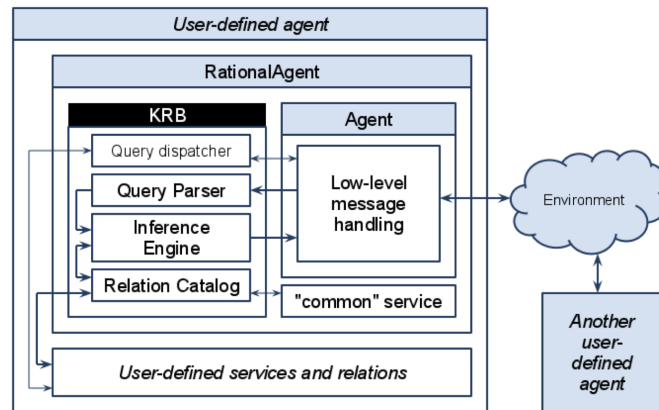


Figure 2. Inside a KnoRBA agent

```
mapping:map(real x, real y),
mapping:map(real y, real x),
0<x, x<1 ?
```

3.2: Query versus Notification

There are two way of calling relations implemented by user, namely, query, and notification. In the case of query, some variables in relation call can be unbounded, to make a fill-in-the-blank types of question; or if all variables are bounded, the truth value of the whole statement is expected to be found. That is the purpose of a query: fill in the blank, or confirm tautology. However, that is not the purpose of every transactions. In many cases, an agent may just want to make the other ones to know something, or want to give some assumption to some target agents before asking them a question, only a uni-directional transaction is required, which is referred to as “notification” in this text. A notification message looks like query messages with the difference that they end with period ‘.’ instead of question mark, and they cannot have unbounded variables. Here is an example that checks if the name of the receiver agent is ‘MyAgent’, and if so, awards it a contract to solve a problem, and runs its internal clock.

```
common:name('MyAgent')?
mapping:contractAward("something", parameters).
clock:run().
```

3.3: Agents

Any agent in KnoRBA has a globally unique address in form xxxxxxxx:yyyyyyyy in which the left hand part is the hardware address of the generating machine and the right hand part, is a unique counter on that machine. The in KnoRBA address-based low-level inter-agent messaging mechanism is provided, but is mostly used internally by KRB.

A user defined agent should be a sub-type of RationalAgent (which in turn is a sub-type of Agent), and implement desired relations according to the desired specialty of that agent. KnoRBA specification defines certain machinery in KRB to register user-defined relations, handle requests, resolve complicated queries against internally defined relations and provide response to them. The architecture of RationalAgent is depicted in figure 2.

As depicted in figure 3.c The transaction begins with an agent sending a query to others through Query Dispatcher module. The query message is usually broad-casted (or multi-casted, if a list of

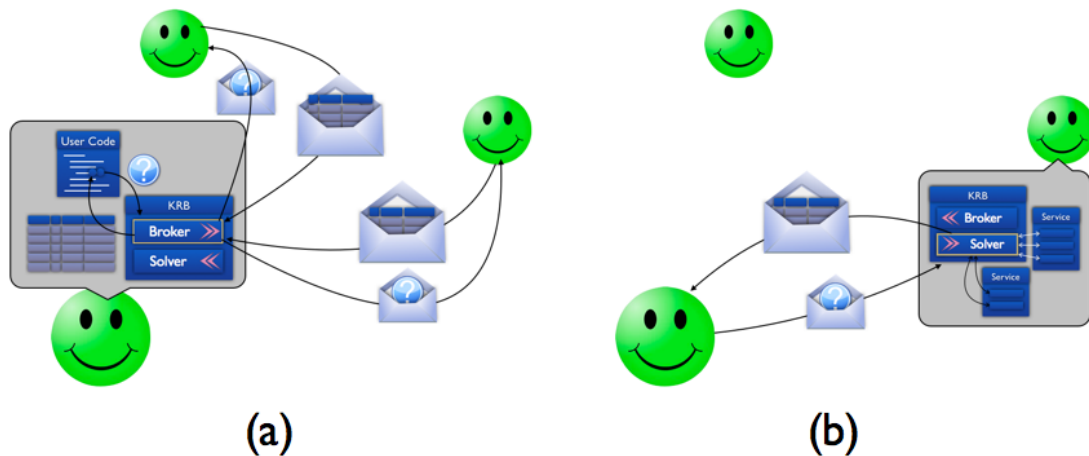


Figure 3. Interaction between KnoRBA agents: (a) An agent dispatches a query and collects the responses in a resultset, (b) an agent receives a query, finds the answer, and sends the result back.

pears is specified). All other agents who receive the message, parse it into logical sentences, and resolve the included relations against those defined in the user code. If at least one of the relations failed to be resolved, the solver agent discards the query. Then, the inference engine invokes the relations according to the query to provide an answer for the inquiring agent. The agent who have initiated the query waits for replies until the specified deadline, or until it receives replies from all specified peers. Until then, it may collect a number of replies. KRB in the inquiring agent combines all replies in an object called `ResultSet` and makes it available to the agent's program code. `ResultSet` provides a number of operations on data like sorting, grouping, averaging, etc. which is usually needed for processing the gathered information.

When inquired, an agent usually receives a query consisting of a list of logical sentences. Then, the job of that agent is to confirm it as a tautology, or produce all the determinable combination of unbounded variables which make it a tautology, and send the result back. The inference engine for doing this task is built within the KRB itself. The inference engine works based on bounding mechanism which is similar to the Prolog language. It uses the Relation Catalog to call relations implemented by the agent several times (as much as needed) in the inference loop, and produce the results. The final result will be a set of records, each of which consisting of a unique combination of values for the unbounded variables for which the given query is a tautology. If there is no unbounded variables in the query, the result will be a single empty record. If the query proved not to be a tautology, the agent simply does not reply.

3.4: Sharing Ontology

Agents and objects are both components, one in KnoRBA and the other in CORBA. In order to make components to work together they must understand each others interface. In CORBA, Interface Definition Language (IDL) is used for that purpose. Any CORBA object should implement a single IDL completely.

In the world of cognitive systems, this is done through specification of ontology which explains what symbolic means an agent has to understand and describe its world. OWL/RDF is an example language to define ontology shared in a cognitive system. For KnoRBA, a new approach called

Knowledge Interface Schema (KIS) is introduced. KIS can be compared to IDL with two differences. First, it is stored on a server so that everybody can access the very same schema, and second, agents are not restricted to implement only one schema, and also are not required to implement any of them completely. The format of KIS is defined in appendix B. Some examples can be found in section 5.

3.5: Environment

On each KnoRBA enabled computer platform, there is one instance of KnoRBA runtime environment. The main task of the environment is to register all the agents exist on its platform and to rely message internally between those agents. The environment can instantiate agents, if the code is available, or accept migrating agents. However, because of extensibility issue, the environment itself do not engage in networking; this task, as well as any other task should be done by specialist agents. The functionality of the environment is represented to the agents by a representative agent that implements `environment` service, hence can be accessed through KnoRBA query and notification mechanism.

4: KnoRBA for Java

KnoRBA for Java consists of a soft language extension that enables implementation of services and relations in Java; plus the definition of primary classes including `Agent`, `RationalAgent`, `ResultSet` and container classes for KnoRBA variable types.

4.1: Creating Agents

The general skeleton of the implementation of an agent is as follows. The agent is made as a single class extending the `RationalAgent` class. A service is defined as an inner class (i.e. non-static class) with `@Service` annotation. `@Service` accepts an optional argument that specifies interface schema implemented by that service. Relations inside a service are defined as methods with `@Relation` annotation. The argument of `@Relation` specifies the signature of that relation, consisting of its name, and list of its arguments. Here is an example:

```
class ExampleAgent extends RationalAgent {
    @Service("http://www.knorba.org/schema/mapping.kis")
    public class mapping {
        @Relation("map(in out real a, in out real b)")
        public boolean map(...) {...}

        @Relation("accuracy(in real x, out real a)")
        public boolean accuracy(...) {...}
    }
}
```

The name of the method representing a relation is not important for KRB, but it should have specific arguments. The first argument, being a `VRecord`, contains the parameters passed to the relation through a call by the inference engine, which is done with accordance of the query or notification being processed. The second argument is provided for fill-in-the-blank queries. For each answer, if any, a record should be added to it, and the value combination for that answer should be set in it. In the case that the call is a notification, the second argument is set by inference engine to `null` when calling the relation. The third argument provides access to information about

the active transition session, including the UID of the asking agent, and a list of user-defined session variables. The return value should be the truth value of the relation given the provided values for its arguments. Here is an example:

```
@Relation("mapping:map(in out real a, in out real b)")
public boolean map(VRecord q, VRecordSet a, Session s) {
    double arg1 = q.getReal("a");
    if(q.isUnbound("b")) {
        a.putReal("b", someFunctionOf(arg1));
        return true;
    } else
        if(someFunctionOf(arg1) == q.getReal("b"))
            return true;
        return false;
}
```

4.2: Interaction

The following shows an example of an agent performing query transaction with the others. RationalAgent provides two methods, ask() and askAll() for that purpose. The ask() method is used for multi-cast and single-cast queries, while askAll() is used for broad-cast ones. These methods accept an argument that specifies the maximum amount of time that the agent is going to wait for the answers. This parameter is optional for ask(): if not specified, the agent will wait until all enlisted peers respond. In the following example, a query is given to all available agents to sample their knowledge (if any) about stock prices in a specific period. After collecting the replies, the obtained knowledge is sorted based on accuracy a and made accessible to the rest of the program code by putting them into some arrays. Then, having retrieved the list of the agents replied to this query, their names are asked.

```
String query = "mapping:mapName('stock-prices'),"
              + "mapping:range(real min, real max),"
              + "common:sampler(real min, real max, 0.1, real x),"
              + "mapping:map(real x, real y),"
              + "mapping:accuracy(real x, real a) ?";

ResultSet rs = askAll(query, a_deadline);

ResultSet sorted = rs.sort("a", SortOrder.Descending);
double[] x = sorted.getRealColumn("x");
double[] y = sorted.getRealColumn("y");

AgentGroup repliers = rs.getReplyingPeers();
ResultSet namesRS = ask("common:name(string n)?", repliers, deadline);
...
```

ResultSet object provides variety of handy operations on data, including sorting, grouping, selection of distinct values, grouped statistics (count, min, max, sum, and average), and row and column filtering. The operations are implemented in indexed manner, hence being fast and memory efficient.

Potential solvers. The method getPotentialSolvers() is provided in ResultSet object which returns an AgentGroup containing the identities of potential solvers (as in section 2.2), if any.

5: A Practical Example: Social Regression

In this section, the development of artificial intelligence through creation of artificial social systems is shown. The designated task of this society is to solve a regression problem. The regression problem can simply be stated as finding a mapping $m : \mathbf{x} \mapsto \mathbf{y}$ between a pattern-space ($\mathbf{x} \in \mathbb{R}^n$) and a target-space ($\mathbf{y} \in \mathbb{R}^m$) based on the given sample data. The solution is implemented by creating a *fortune landscape* parallel to pattern-space and letting the agents to be guided by it through their instinctive self-interest, and allocate the pattern-space and establish connections in the way that their society advances towards an optimal solution to the given problem. This solution is referred to as “social regression” throughout this article. Detailed explanation of social regression can be found in [4].

There are a number of schema used in the implementation of social regression, but the subject of regression is implemented around a schema called mapping and is defined as follows.

```
SERVICE mapping {
  RELATION mapName(string name);
  RELATION map(real x, real y);
  RELATION accuracy(real x, real a);
  RELATION range(real left, real right);
  RELATION assignment(string function, real center);
  RELATION fortuneVision(real a, real b, real left, real mid, real right);
  RELATION enumerable();
}
```

Agencies implement an additional schema as follows.

```
SERVICE agency {
  RELATION member(in out long address);
  RELATION join(in long address);
  RELATION leave(in long address);
}
```

Unlike conventional social simulation systems, `RationalAgents` run asynchronously. Therefore, a thread inside each of them that “ticks” once a while, runs simulation cycles for that individual agent separately. This thread can be controlled via clock service interface.

```
SERVICE clock {
  RELATION run();
  RELATION stop();
}
```

The implementation of social regression for this paper consists of the following agents.

- **LookupTable:** To make the implementation more realistic, the training data is not fed into the agents by invisible hands from outside, but rather put in this type of agents and let them be inside the artificial world, rather naturally. Then, any agents looking for training data, looks for some body who has it, eventually finds this agent and retrieve the data to train itself. In present experience, this is the only type of agent implementing mapping:enumerable() relation.
- **Basis agents:** Adaptive agents forming the leaves of social network of specialists. They directly interact with pattern-space and try to fit a curve to the area of their interest.
- **Mixer agents:** agencies forming the midlevel nodes in the social network of specialists. In this implementation, only `Mixer` agents implement the `agency` schema.
- **Trainer agent:** is there just to ask random question (bootstrapping) from all specialists to engage them in learning and working on the problem.

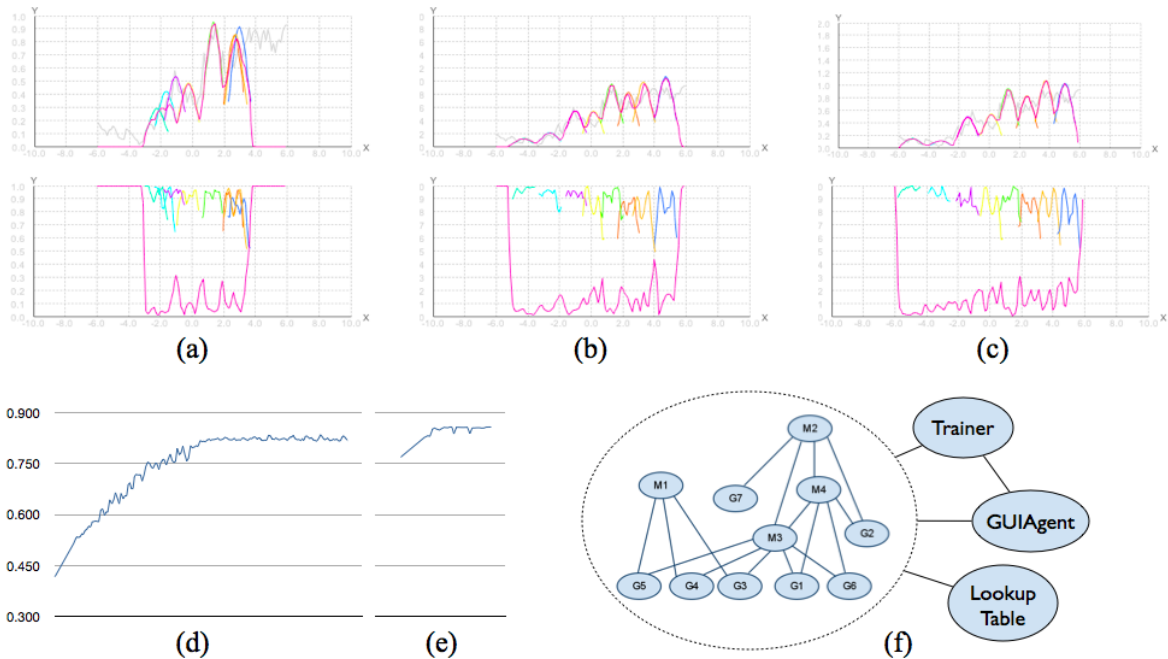


Figure 4. (a), (b), and (c) randomly placed specialists propagate on the pattern-space to gain control on high income areas. The upper charts show individuals and the the overall output of the ensemble, while the lower ones show accuracy of individuals and overall *error* of the ensemble. (d) The average ensemble error being reduced while the time passes. (e) reduction of average ensemble error when specialists are distributed evenly in the beginning. (f) the social system of the social regressor

- **GUIAgent** is assigned to the GUI which is developed to interact with, and keep track of specialists, visually. It also plays the role of the end-customer.

The purpose of this experiment is to show the implementation of the synthesized artificial society actually works and serves its purpose. The data used for training in this experiment is taken from actual historical records of exchange rate of Japanese Yen for US Dollar in the period between mid 2006 and mid 2009. In this experiment, *Basis* agents are kept extremely simple: each of them can fit a Gaussian bell-shaped function to the portion of the curve in its area of coverage.

As shown figure 4.a, specialists are randomly placed on the pattern-space. Then they start to propagate to allocate areas with higher expectation of income (figures 4.b and 4.c). While doing so, they help reduce the ensemble error as well, as shown in figure 4.d. To make a sense of comparison with HME (Hierarchical Mixture of Experts) [3], in figure 4.e, the specialist are initially distributed uniformly (like a hard wired HME), and the improvement in the result is observed. This confirms that, not only social regression with its complex concurrency, intelligence, and interaction model works seamlessly, but it also can outperform existing solutions.

6: Conclusion

Knowledge Request-Broker Architecture (KnoRBA) is made to provide a basis for development of an emerging generation of socially intelligent software systems. Although the idea of such systems is almost as old as AI, the implementation of such systems have been a major challenge ever since. Having its roots in both widely successful object request-broker systems in one hand, and cognitive architectures in the other hand, KnoRBA is proposed to become a practical solution for this issue.

To state the contribution of this article, having explained the potential role of social software paradigm in operating complex systems in a complex world, KnoRBA is proposed to be a step towards realization of this paradigm. For that purpose, KnoRBA addresses three major issues concerned with creation of a computationally intelligent artificial society.

First, to realize mutual understanding between heterogeneous agents, instead of persuading the failed efforts like KSE in proposition of a global language or protocol, it takes advantage of request-broker paradigm. By doing so, not only heterogeneous agents can interact, but also because issues like concurrency, networking, query resolution, and knowledge reception is internally taken care about, and made transparent to the developer, interacting with a society of heterogeneous agents is made even easier than interacting with a remote database system. In KnoRBA, agents can be resolved either by their identity or their functionality.

Second, KnoRBA provides an engineering method for implementing social systems by picturing an architecture for social agents and their society. Such an architecture was missing in abstract theories like the Society of Mind and The Emotion Machine, and other efforts inline with these theories.

Third, KnoRBA provides API tools for practically implementing a social software. Unlike the case of cognitive systems, KnoRBA agents can be implemented in a general-purposed language, enabling the developers to create wider variety of solutions more easily. Unlike JACK, not only KnoRBA architecture can cover wider variety of intelligent agents, but also the footprint of its extension on the host language is kept minimal, making it easy for existing developers to learn and adopt its technology.

To make a practical case, the experiment of social regression, given in this paper, besides demonstrating a seamlessly working prototype of KnoRBA, it shows that social computational intelligence is effective to create better systems and how much a platform like KnoRBA is important to facilitate creation of such intelligent systems. In this experiment, it is observed that multiple Basis agents, Mixer agents, LookupTable agents and a GUIAgent are autonomous heterogeneous agents, despite having radically different ways of understanding and representing knowledge, work together to carry out a shared social task.

Speaking about deficiencies, implementing KnoRBA agents are not as easy as implementing CORBA objects, because relations are not natively supported in object-oriented languages and a work-around is needed. High message cost and the time spent for communication should also be considered by the users of KnoRBA.

Proposition of KnoRBA opens plenty of new areas for future work. In short-term, networking and agent mobility is considered to be implemented. Networking will be implemented by introducing respective specialist agents. For mobility, hibernation of agents (serialization), streaming and reviving them should be implemented. For mid-, and long-term, a central repository for interface schema should be available to public so that they can contribute community-made agents to the network. Hopefully, it can become a big collection of interactive algorithms, and a realization of the Society of Mind. Exploring the applications of the concept explained in this article for Robotic

systems, and ambient intelligence solutions, would be the most exciting.

A: BNF Grammar of KnoRBA Queries

```

<query> ::= <sentences>
<sentences> ::= <sentences> <sentence> | <sentence>
<sentence> ::= <question> | <notification>
<question> ::= <complex> "?"
<notification> ::= <complex> "."
<complex> ::= <and-complex> | <or-complex> | "(" <complex> ")" | <call>
<and-complex> ::= <complex> "," <call>
<or-complex> ::= <complex> ";" <call>
<call> ::= <service-list> ":" <name> "(" <arg-list> ")"
<service-list> ::= <service-list> ":" <name> | <name>
<arg-list> ::= <arg-list> "," <arg> | <arg> | empty
<arg> ::= <unbounded> | <string> | <numeric>
<unbounded> ::= <type> <name>
<type> ::= <name>
<string> ::= "'" <chars> "'"
<chars> ::= [any character except "'"]*
<numeric> ::= [-]? [1-9][0-9]* [.[0-9]+]?
<name> ::= [a-zA-Z][a-zA-Z0-9]+
    
```

B: BNF Grammar of Knowledge Interface Schema

```

<service> ::= "SERVICE" <name> "{" <members> "}"
<members> ::= <members> ";" <member> | <member>
<member> ::= <relation>
<relation> ::= "RELATION" <name> "(" <arglist> ")"
<arglist> ::= <arglist> "," <arg> | <arg> | empty
<arg> : ::= "in" <in-arg-rest> | "out" <out-arg-rest> | <arg-rest>
<in-arg-rest> ::= "out" <arg-rest> | <arg-rest>
<out-arg-rest> ::= "in" <arg-rest> | <arg-rest>
<arg-rest> ::= <type> <name>
<type> ::= <name>
<name> ::= [a-zA-Z][a-zA-Z0-9]+
    
```

References

- [1] Hiroshi Deguchi. Agent based simulation for complex social systems by soars – design, simulation and analysis of massively multi-agent social systems. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems*, May 2007.
- [2] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. Kqml as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management, CIKM '94*, pages 456–463, New York, NY, USA, 1994. ACM.
- [3] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6(2):181–214, 1994.
- [4] Hamed Khandan and Takao Terano. Synthetic design of a social regressor and its implementation using knowledge request-broker architecture. In *Proceedings of the Second World Congress on Nature and Biologically Inspired Computing (NaBIC2010)*, 2010.
- [5] Christian Lebiere, Mike D. Byrne, John R. Anderson, Yulin Qin, Dan Bothell, and Scott A. Douglass. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, 2004.

- [6] Richard L. Lewis. Cognitive theory: Soar. In Neil J. Smelser and Paul B. Baltes, editors, *International Encyclopedia of the Social & Behavioral Sciences*, pages 2178–2183. Elsevier Inc., Highway 50 East, Linn, MO 65051, USA, 2004.
- [7] Marvin Minsky. *The Society of Mind*. Simon & Schuster, 1988.
- [8] Marvin Minsky. *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. Simon & Schuster, 2006.
- [9] Kinji Mori. Expandable and fault tolerant computers and communications systems - autonomous decentralized systems. *Computers and Communications, IEEE Symposium on*, 0:228, 1999.
- [10] M.J. North, E. Tatara, N.T. Collier, and J. Ozik. Visual agent-based model development with repast symphony. In *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*, Argonne National Laboratory, Argonne, IL USA, November 2007.
- [11] Object Management Group. *Common Object Request Broker Architecture (CORBA/IIOP) V3.1*. Object Management Group, 2008.
- [12] Foundation of Intelligent Physical Agents. Agent communication language specifications. Online at <http://www.fipa.org/repository/aclspecs.html>, 2002.
- [13] Ciarán O’Leary and Mark Humphrys. Building a hybrid society of mind using components from ten different authors. *Advances in Artificial Life*, Volume 2801:839–846, 2003. ISBN 978-3-540-20057-4.
- [14] Scott E. Page. *The Difference: How the Power of Diversity Creates Better Groups, Firms, Schools, and Societies*. Princeton University Press, 2007.
- [15] Ramesh S. Patil, Richard E. Fikes, Peter F. Patel-Schneider, Don Mckay, Tim Finin, Thomas Gruber, and Robert Neches. The darpa knowledge sharing effort: Progress report. In Bernhard Nebel, editor, *Proceedings of the Third International Conference on Principles Of Knowledge Representation And Reasoning*. Morgan Kaufman, 1992.
- [16] Steven Pinker. *How the Mind Works*. W. W. Norton & Company, 1999.
- [17] Milind Tambe and David Pynadath. Towards heterogeneous agent teams. *Multi-Agent Systems and Applications*, pages 187–210, 2006.
- [18] Michael Winikoff. Jack intelligent agents: An industrial strength platform. In Gerhard Weiss, Kathleen M. Carley, Yves Demazeau, Ed Durfee, Les Gasser, Nigel Gilbert, Michael Huhns, Nick Jennings, Victor Lesser, Katia Sycara, Michael Wooldridge, Rafael Bordini, Mehdi Dastani, Jrgen Dix, and Amal Fallah Seghrouchni, editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 175–193. Springer US, 2005.
- [19] David Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, pages 1341–1390, 1996.

Biography



Hamed Khandan is a Ph.D. candidate at Tokyo Institute of Technology, Department of Computational Intelligence and System Sciences. He received his bachelor degree in computer software engineering in 2004, and holds a M.S. in mechatronics from the Science and Research branch of I. Azad University (2008). Research interests include: philosophy of science, emotional computing, social computational intelligence, and distributed machine control.



Takao Terano is a professor at Department of Computational Intelligence and System Science, Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology. He received BA degree in Mathematical Engineering in 1976 from University of Tokyo, M. A. degree in Information Engineering in 1978 from University of Tokyo, and Doctor of Engineering Degree in 1991 from Tokyo Institute of Technology. His research interests include Agent-based Modeling, Knowledge Systems, Evolutionary Computation, and Service Science. He is a member of the editorial board of major Artificial Intelligence- and System science- related academic societies in Japan and a member of IEEE, AAAI, and ACM. He is also the president of PAAA.