Evolution of thinking models in automatic incident processing systems

Alexander Toschev, Max Talanov, and Salvatore Distefano

Kazan Federal University, Russia
atoschev@kpfu.ru,dr.max@machine-cognition.org,salvatdi@gmail.com
http://www.kpfu.ru

Abstract. In this paper we describe the evolution of the application of thinking models in automatically processing a user's incidents in natural language, starting with the model based on decision trees and ends up finishing with the human thinking model. Every model has been developed, prototyped and tested. The aArticle contains experiments results and conclusions for every model. After evolving several theories, we found the most suitable for solving the problem of automatically processing a users incidents.

Keywords: Artificial intelligence, Machine Understanding, Remote Infrastructure Management, NLP, Reasoning, Automation, Knowledge Base, Intelligent Agents

1 Introduction

Today IT infrastructure outsourcing is a very popular domain for thousands of companies all over the world. This domain connect companies that provide remote support for customer's IT infrastructure. It's very expensive to have personal IT department if your firm doesn't operate in the field of IT, which is why IT infrastructure outsourcing is a large domain with a lot of companies on the market [**?**]. So, efficiency is very important for this business due to high competitiveness [**?**]. During the development of outsourcing domain a lot of information systems have been created. For example, HPOpenView [**?**], ServiceNow which is much more popular, it also used in CERN [**?**]. But all of them required human specialist to work. For example, specialist registers incidents in the system, analyses them and tries to solve the issue. The system only holds workflow and sometime automatically assigns specialist.
This paper describes a system for detecting and solving incidents produced by user in natural language. The possibility of automation is based on a large amount of trivial incidents in such systems as "I do not have access to...", "Please install..." [**?**]. There are set of mandatory requirements:

- Understanding a request produced by a user in natural language;
- Capability to learn;
- Capability to reason (using the analogy, deduction, e.t.c.);

– Decision making.

We demonstrate the progress and evolution of models and architectures taking in account these requirements in the following.

2 Thinking models

To solve the problem we evaluate several techniques and models. In this section we present results of this evolution.

2.1 Intellectual Document Processing (IDP) model

One of the early tasks was to parse and understand documents. This is why the first model was named Intellectual Document Processing. The system makes meta-information from an uploaded document and is able to quickly find the required information. The main idea behind this system is decision tree algorithms [**?**]. General workflow of the system described below:

1. System should be trained, which is why user must upload initial information;
2. System proceeds these documents and tries to extract information;
3. System annotates information with tags like: “birth date”, “job title”, e.t.c.;
4. User uploads new documents and the system automatically extracts all valuable information, so you can simply navigate to this information using the tag tree.

On the figure 1 the architecture of the proposed solution is shown. To start use of the system training is required, so several annotated documents are provided to the Trainer component, after which the Evaluator processes them. After that Applier can work to extract information from the raw documents provided by the user.

Fig. 1. IDP architecture

The system has been designed with aim of supporting the HR division of an organization. So, they can quickly parse a lot of candidates proposals. Technically this system required a lot of initial data to start work. Moreover, if the system tries to processes an unsupported document format it will crash.
As the sample for testing input information, candidates application forms in the form of Word documents are used.

2.2 Menta 0.1 model

The second model was developed as an intelligent request processing system for automatic software generation. For example, if you need to simply add a new field or change it, you can do so automatically, so do not need spend a lot of time on simple requests. The system can process simple requests such as "Add name to the customer". The main workflow for this system is described below:

1. Receive request and formalize it;
2. Generate action (special object, which contains information for the next step);
3. Change the target application which is represented in OWL (Web Ontology Language) [**?**] model using the action from previous step;
4. Generate application by OWL model;

Decision tree technology is used for the solution search (find suitable solution option for the incoming request). This model supports requests in natural language by using Stanford Parser [**?**] as a NLP processing tool. This approach is much more intelligent than the previous one but it still has problems like:

- If the input request contains several errors in grammar, spelling, etc. system failed to proceed such requests;
- The solution search system works only for one application architecture and can't use information for other systems. For example, system knows how to add field to the class in system A, but it can't proceed the same operation for the system B.

For example, the model works with the following requests:

- Please add Name field to the Customer;
- Please remove Lastname field from the Customer;

Several forms of these requests and others are used for tests as experimental data. Demo contained several examples and runs as the Java console application. However, this model has the next limitations:

- Limited set of actions: only create/delete;
- No learning mechanism;
- No feedback to the user is provided;

That's why the next approach has been introduced.

2.3 Menta 0.3 model

This approach, which improves the solution search mechanism, has been discussed in general terms above. To this purpose the genetic algorithm is adopted also including:

- Acceptance criteria — set of rules to the solution;

– Data model in semantic network OWL representation;
– Use of logic in rules (such as induction, abduction, etc.).

The overall solution architecture presented on figure 2. It contains modules: MentaClient — web-service client of the system, could be the GUI application in .NET; MentaController — the main component of the system that invokes all the other components in the workflow; Communicator — component that processes the responses of the MentaClient and creates requests to MentaController; KBServer(Knowledge Base server) — the storage of ontological information used in the system; SolutionGenerator — the generator of the Solution for the MentaClient request. System works with specialist according to the following workflow:

Fig. 2. Menta 0.3 architecture

1. User sets acceptance criteria by using special tool;
2. The system combines different components of the application to build by genetic algorithms application suitable to acceptance criteria;
3. The system checks final generated solution by asking user.

Acceptance criteria set of logic rules: the system checks it by using the logic engine NARS [**?**]. However, after testing we found that the creation of acceptance criteria is not trivial and takes a lot of time. Moreover, on large models genetic algorithms take long time to find suitable solutions. Detailed description of this approach is available in the article [**?**].
For the model validation we used predefined sets of acceptance criteria, a target software application description and a set of possible modules. After setting up acceptance criteria the system generates the solution as a changeset for the existing application.

2.4 TU model

After evaluating all models the final model had been build based on Marvin Minsky theory of six levels [**?**]. This model contains all general parts of the previous models. The general modules of this system are:

– Incident;
– Learning;
– Solution search;
– Solution application.

The model is much more general than the previous ones and can be used in several domains. The main concepts of Minsky's theory are: Critic, Selector, WayToThink. Critic is a probabilistic predicate. In other words it is a trigger, which activates after several events. After activation critic checks if it reacts to

this event by using a set of logic rules, which are evaluated by the reasoning server PLN in [**?**]. After checking, it returns the Selector with the set of resource for processing incoming request. List with definitions of implemented critics follows:

- Incoming request — activates when incoming request comes from user;
- Natural language processing — activates for the incoming request and run build (Way-to-think) of the semantic model of the incoming request;
- Preliminary splitter — activates for the incoming request and run the correction appliers: grammar, spelling, annotating for the incoming request;
- Time control — activates from time to time to check request processing time;
- Emotional state control — activates when system state change and run resource allocation.

The next component is the Selector. Its job is to obtain resources from the internal knowledge base. The Selector returns another Critic or another Selector or Way-to-think. Below is the list with definitions of implemented Selectors:

- Direct instruction problem Selector;
- Problem with desired state Selector.

For example, "incoming request critic" returns "direct instruction problem" selector which returns Way-to-think "solve direct instruction problem". The third component of the model is Way-to-think. It's processes data modification in general. In brief, it solves incoming request. In the IT outsourcing domain waytothink can be:

- Learned knowledge — system already knows how to solve this issue;
- Adaptation — system applies existing knowledge by using the analogy such as "please install Firefox" and "please install IE";
- Reformulation — system puts newly obtaining knowledge to database by cleaning up it to more general way such as "please install office" to "please install msi";
- Cry4Help — system raises question to the user;
- Solution search — different ways to find the solution. The solution search algorithms use tree distance algorithm to found suitable set of how-to to solve the problem. How-to's incapsulated in the Solution concept, which is linked to the problem. The search works between incoming problem and existing problem;
- How-to — special Way-to-think that represent solution for problem. It has been built as a general algorithm for applying solutions. Such as a set of script commands;

The second important part of the model is thinking levels: higher level incapsulates more complex behavior than including them. These levels are:

1. Instinctive
2. Learned
3. Reflective

4. Self-reflective
5. Self-conscious

In the theory it was abstract concept. In the TU model we have made domain specific and adopted to software system.
The first level — instinctive — includes instincts such is the egg-retrieval behavior of the graylag goose [**?**]. The level — self-conscious — includes higher goals and ideals. Regarding incident processing the first level controls auto-generated incidents such us "LOT2345 required." If the system can't find result on the first level it goes down to the next level. On the second level system proceed incident by natural language processing software and creates semantic networks, after that several critics become active and try to classify problem.
The 3-rd level controls request state and sets goals for the system. For example, the base goal for the system is to help user. Goals have tree structure. Subgoals of the help user goal are: understand request, classify problem, find solution.
The Fourth level controls the time of the request, and if it runs out of time raises the problem on the next level.
The Fifth level initializes communication context and performs user iteration.
The 6-th level controls the general system state — emotion state. It also controls system's resources. So, if the 4-th level raises the problem of request's time it adjusts resources to solve this request as soon as possible. On this level system also performs monitoring which can indicate that replacement of hardware components or resource upgrading is required.
Another problem was how to pass data between thinking levels, current operation context and knowledge base of the system. Special concepts were introduced:

- Short term memory — exist during the current request and located in the memory;
- Long term memory — after finishing request processing system merges Short term memory to the knowledge base.

Way-to-thinks are working with short term memory and after successful incident processing the system copies newly obtained knowledge to long term memory. This way bad solution are not copied to knowledge base.
System knowledge base is build over the non-sql database engine Neo4j. Solutions, problems, Way-to-think, Critics, Selectors. That's why all system components are dynamic.
One of the main option of the system is to learn new knowledge by interaction with the user. At the beginning system has a few concepts: object and action, which increase by learning. Learning process is quite similar to request processing, but instead of searching for the connections between incoming concepts and knowledge base's concepts it creates them. For example, learning sequence: "Browser is an object. Firefox is a browser". Now the system knows two new concepts: Firefox and browser. Also it builds links between browser and Firefox.
This paper extends the Minsky's model into a new architecture shown in figure 3. It contains five components: MessageBus — connector between different system's modules; TUWebService — main communication component; CoreService —

general processing module; ClientTarget — client component for solution applying; DataService — data provider component, includes knowledge base. The model has been extended by specific Critics, Selectors, Way-to-thinks. Thinking levels have been adapted to system tasks. Since the architecture is general, the application of this model is very flexible from helpdesk to virtual assistant. More detailed description of architecture is available in [**?**].

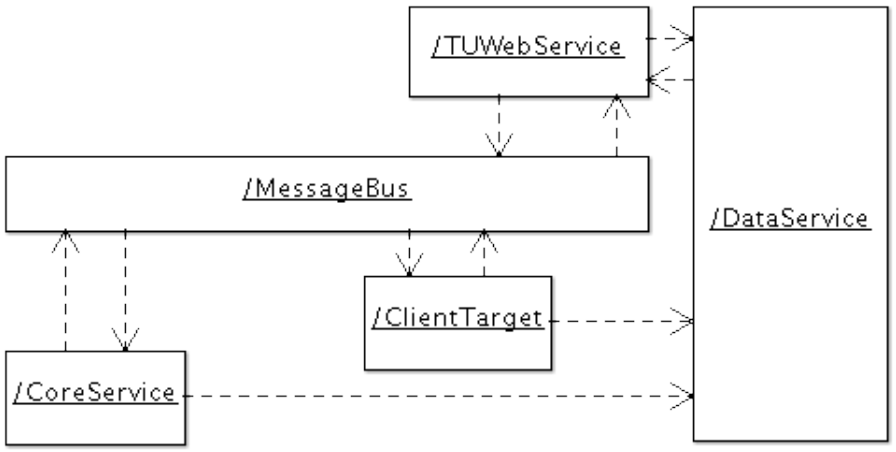

Fig. 3. TU architecture

3 TU. How it works

Let's take as a example the incident "Please install Firefox". The critic "Inbound Incident Received" reacts and sets up the global goal "help user". This activates critics that are associated (a criteria of a critic matches) with this goal starts processing incident. This triggers the critic "get incident type", that activates "natural language processing" way-to-think, which in its turn extracts a semantic network from the inbound incident description. Several critics connected to the sub goal of "get incident type" are activated. All of them analyze the inbound context (short term memory) with semantic network of incident. Usually a critic has several rules, that are populated using the inbound context and evaluated via probabilistic reasoner. As the result a critic returns a probability of an action. Let's system has three incident type classification critics: direct instruction (when request is a direct instruction, like "please install something"); problem with desired state (i want office instead of Wordpad); problem without

desired state (i can't open pdf file). In this case the *direct instruction* critic returns maximum probability, because request contains all required concepts for this particular critic. The approximate dependencies structure: action — please; object — Firefox. In the case of *problem with desired state* there are missing concepts: subject (which describes problem), previous state (current situation). After activation of the critic *direct instruction* returns the way-to-think which tries to search for a solution. The critic to *problem with desired state* triggers the *simulating and modeling* way-to-think that searches for a suitable action to put the system from current state to the desired state. After the solution was found (several solutions could be selected) TU tries to apply it to the target system and collects a feedback from a user. If the first solution doesn't help system tries second solution, third and so on using the probability measures.
Let's look at another example: "I install Internet Explorer previously, but i need Chrome" (original syntax errors saved). Critics : *direct instruction* and *problem with desired state* are selected, but penalty decrease probability of the *direct instruction critic* and proper *problem with desired state* will be activated. It's important to mention that all of these concepts are the part of knowledge base and can be dynamically extended, this way system could be build as rally flexible.

4 Experimental data

To test the TU model and software a special set of requests have been built. They was extracted from real time systems of ICL-Services [**?**]. Initial dump contained over 1000 incidents. After processing, 43 unique incidents have been preselected. Why are they unique? Because other incidents are just variants of these 43. For example, "Please install Office" and "Please install browser" are similar — simply run install program.
TU model can be tested Among these 43 incidents 85% have been successfully understood by the system. And the if solution is available in database — it is selected.

5 Conclusions

We described 3 models which were created to automatically process a user's request. After investigation we found that the TU model is the most suitable solution. We consider it to be a very good combination of an application and the flexibility of human's thinking. This model provides the ability to build the system in a restricted-resource environment (because it does not use neural networks), which has been proven in the research by creating software using this model.

Acknowledgments

Thanks to Marvin Minsky for his incredible theory.