Fusing individual choices into a group decision with help of software agents and semantic technologies

Konrad Ponichtera, Mikołaj Małkiński, Jan Sawicki
Faculty of Mathematics and Information Sciences
Warsaw University of Technology
Warsaw, Poland
ponichterak@student.mini.pw.edu.pl

Maria Ganzha
Department of Intelligent Systems
SRI PAS
Warsaw, Poland
maria.ganzha@ibspan.waw.pl

Marcin Paprzycki
Faculty of Management and Technical Sciences
Warsaw Management Academy
Warsaw, Poland
marcin.paprzycki@mac.edu.pl

***Abstract*—What happens when hunger, individual eating constraints and student economy are combined with some computer skills? This contribution introduces a agent-semantic application, which expedites process of choosing where and what to eat, when a group of hungry students would like to meet and have lunch together. Furthermore, individual dietary restrictions are taken into account. Specifically, software agents are used to facilitate negotiation mechanisms used to find (i) place to meet, and (ii) common food, while semantic technologies are used to represent food and user profiles (i.e. food allergies). The initial system has been implemented and validated on the basis of selected use case scenarios.**

***Index Terms*—Software agents, semantic technologies, group negotiations, individual restrictions.**

I. Introduction

Nasi lemak? Kebab? Paneer tikka? Pizza? Everyday billions of people face questions, answering which sometimes seems to require inhuman effort. It is seemingly easy to choose what one wants to eat, but what if a group is to make a decision? Moreover, what if there are multiple special deals, e.g. half-price-off for meals prepared for large groups? Furthermore, what if the menu has large selection of dishes (even a pizza place can offer 30+ different pizzas)? Finally, what if some of future co-feasters have strict dietary restrictions (e.g. no durian, no pineapple, no crustaceans, etc.)? Obviously, there must be a way to deal with such situations, as even computer science students handle it on a day-to-day basis (growing hunger, and some peer pressure, usually, do the trick). Here, let us also note that, with growing number of food allergies, one of important issues is to be able to correctly recognize what ingredients are involved in creation of a given dish. Obviously, some people may laugh at warnings of the type: "this pack of hazelnuts may contain traces of nuts as they were processed where nuts are processed", which we can find on many products in a grocery store. Nevertheless, for someone allergic to pineapple, being served Hawaiian Pizza, is not going to be fun. In this context, with modern technology, there must be a better way than prolonged discussions, or manually removing dish after dish (case of linear search), when at least one person in the group states that (s)he will not eat it.

Therefore, we would like to suggest that a solution based software agents and semantic technologies can be the way of approaching such problem. Here, software agents are going to provide the negotiation infrastructure (which is what software agents are known to be very good at, see [15]), while semantic technologies allow representation of individual dietary restrictions (see, for instance [14]).

The core use case involves negotiations among a group of computer science students, who plan to eat together, while some of them have (known to them) food allergies. Here, let us note that nowadays, for economic reasons, restaurants [1] may use various marketing strategies to increase their income, and boost efficiency of their "backend", i.e. the kitchen. One of such options may be offering discounts on meals (or beverages) bought in large quantities. A consequence of such actions is eagerness of clients (students, in particular) to buy multiple plates of the same dish (being part of the promotion). However, this approach requires collaborative thinking. Reaching the (typical student) goal of eating as much as possible, while paying as little as possible, involves dealing with communication abilities, relationships within the group, self awareness of what one actually wants (and can) eat. However, it also depends on ability to compromise, persuade, encourage, or even manipulate each other. For the comfort of further analysis, and fixing less attention on the sociological and psychological aspects of the process of finding the common meal, it is assumed that all "personal preferences" are put aside (only dietary restrictions are taken into account).

Summarizing, we are interested in the following scenario. A group of students wants to go together to restaurant and order large number of portions of the same dish (to reduce cost), while maximizing satisfaction of constraints imposed by dietary limitations. To achieve this goal they will use an agent-semantic application that will perform negotiations on their behalf to determine the dish they should order (to maximize price-performance).

To present the developed solution, we proceed as follows. In Section II we introduce core results concerning: (i) application of software agents in negotiations, and (ii) use of semantic technologies to represent food and dietary restrictions. We fol-

[1] We will use term *restaurant* to denote any food serving establishment that can be found "around university campus", including bars, pubs, etc.

978-1-5386-7167-2/18/$31.00

low with discussion of how semantic technologies (Section III) and software agents (Section IV), are used in our application. Next, in Section V, we present technical aspects of the developed application. Experiments, illustrating how the application works, in selected cases of "hungry students problem", are presented in Section VI. Lastly, Section VII, summarizes our contributions and suggests further developments.

II. Related work

Let us start from observing that use of agents in negotiations, and semantic technologies related to food, has been described in multiple works. Let us start with how agents were used in negotiations. First, let us look at the paper "Semi-Global Leader-Following Consensus of Linear Multi-Agent Systems With Input Saturation via Low Gain Feedback" [16]. Here, agents are used not only to simulate a discussion (the titular consensus), but also to prove that (under some assumptions) such consensus is achievable. Asymptotic achievability of reaching a common decision is, obviously, the main assumption of this project. However, none of the agents (representing hungry students) is considered to be a "leader" – big difference. This difference, however, is not present in the second paper "Consensus of Linear Multi-Agent Systems by Distributed Event-Triggered Strategy" [12] which, apart from consensus reaching process, uses asynchronous responsive behaviors. What these two papers have in common, though, is a formal statement that "consensus is reachable".

On the contrary to previous choices, let us consider use of agents in smart cities, as reported in "Towards smart city: M2M communications with software agent intelligence" [6]. Here, the common goal is improving human life with an agent-oriented solution. To be precise, it highlights the comfort of transferring H2H (human to human) to M2M (machine to machine) conversations. Imagine, what if every "smart restaurant" had a hub, to which all agents of human clients could connect, communicate a choose a meal to reduce the overall cost? Just like in normal life, where there exist many ways of discussing so that some decision is obtainable, the very same methods can be, almost directly, migrated to the agent paradigm. Although, in our solution agents try to come up with a result by consensus, one could go with an argumentation-based mechanism, just like the one described in "Strategic Argumentation in Multi-Agent Systems" [17]. Both, in consensus and in argumentation approaches, it is crucial to build-up some knowledge about the subject of a dialog. While, in the latter option, agents use obtained knowledge to exploit arguments of others, effectively treating them as opponents, in the former case agents work together by sharing what they know so that one of them can announce that consensus has been achieved, which can be easily checked by querying every conversation participant if they are fine with that.

Coming back to the concept of "smart restaurant", one of fundamentals steps would be to describe the concept of food and all possible relations in a standardized fashion. Having the asserted taxonomy, described in a machine-readable format, one would be able to use computers for performing inference, potentially extracting non-trivial information even from a simple knowledge data set. One of such attempts is described in "User's Profile Ontology-based Semantic Framework for Personalized Food and Nutrition Recommendation" [3]. Here, the ontology engineering has been used in order to provide mechanisms of obtaining food recommendation based on many factors – from health conditions to cultural influence. For instance, for some people controlling blood sugar levels it may be difficult and often requires help of the experts. Fortunately, as demonstrated in [4], an automated system designed specifically for diabetic patients can be created. Similarly as in our work, by using OWL and SWRL, authors conduct a research and discuss satisfying outcomes. One of key features of such recommendation system should be to make them easily accessible to their users. Authors of [7] present a whole system for mobile platform, which provides suggestions about consumed food.

Note that consumers may have not only specific dietary restrictions, but also somewhat "fuzzy" preferences. Therefore, food recommending system should have "deep understanding" about possible offers. Here, a group of researchers, from a food delivery company, addressed this problem [10]. They used a graph, representing food, to better understand user queries. They showed that client may not necessarily be interested only in the "queried product", but also in similar ones. For instance, user might initially search for a Chinese meal, but decide to order Japanese food. Therefore, a graph-like data structure can perfectly represent such relations, as is demonstrated in [9]. This paper describes how various information about the Turkish cuisine was described using a hierarchical ontology model. Additionally, semantically described data is linked to appropriate images of food.

Lastly, note that a well-designed ontology can, and should, be used for multiple purposes. One of use cases is demonstrated in [5], where authors consider requirements to model food supply chain, in order to support safety regulations. This paper demonstrates how multiple food ontologies can be integrated.

III. Use of semantic processing

Let us now describe how semantics provides information available to negotiating agents. As seen, ontologies can be used to represent food ingredients and dietary restrictions. An ontology is an explicit specification of a conceptualization [11]. Using an ontology to describe primitive relation of food ingredients causing an allergy may seem like an overkill. However, this approach was chosen to facilitate extensibility and configurability. Generally, there were two possibilities of obtaining an ontology that could be embedded in our application. (1) To develop an ontology, precisely matching food of interest. (2) To use an existing one, developed by somebody and shared on a free license. Following good practices of ontology engineering, and taking into account that ontology development is not our goal, we have selected a well-known pizza ontology [2]. Note that after a minor extension (depicted in Figure 1), it can contain information required for our application. Obviously, change of the underlying domain

representation, to a more mature one, would be necessary for a real-world system. However, let us stress that this would not require major changes to the core of the developed application, thanks to the loose coupling between business logic and data included in the ontology. For instance, to introduce new allergies it would be enough to add their classes to the ontology and define associated SWRL rules.

A. Extension of ontology with allergies, consumer and custom properties

As stated, in addition to food, we need to represent dietary restrictions. Let us note that there exists a more mature, and considerably more detailed ontology, namely the *Food Ontology* [8]. It contains a huge amount of ingredients, from which meals can be composed, and includes more than 200 SWRL (*Semantic Web Rule Language*) rules, which allow to express relations between entities using concepts used in Web Ontology Language: classes, individuals and properties. It is an appropriate way of sharing the same set of rules between applications. Furthermore, it allows utilization of reasoning, based on defined rules. The mentioned ontology utilizes SWRL rules to describe, which ingredients should be avoided by a person with one of the four allergies: egg, fish, gluten and lactose. However, for sake of simplicity, it was not selected as the knowledge base for our application. Instead, the pizza ontology has been extended with concepts related to allergies, which exist in *Food Ontology*. After defining custom SWRL rules, describing interaction between pizza toppings and allergies, the ontology was rich enough to provide non-trivial reasoning capabilities. Specifically, pizza ontology was extended with following classes:

- *Allergy* – a base class representing allergy,
- *NutAllergy*, *LactoseAllergy*, etc. – specific allergies,
- *Consumer* – person who can have allergies and eats food.

The following object properties were also introduced:

- *eatsPizza* – domain: *Consumer*, range: *Pizza*,
- *hasAllergy* – domain: *Consumer*, range: *Allergy*,
- *hasNutRisk*, *hasLactoseRisk*, etc. – domain: *Consumer*, range: *PizzaTopping*.

In case, when a more sophisticated ontology was used, range of *hasNutRisk* or *hasLactoseRisk* should be limited to a particular ingredient, which a person having that allergy does not tolerate. Since pizza ontology does not describe ingredients of pizzas, only toppings have been taken into account in SWRL rules. Therefore, for the time being, the range of mentioned properties is *PizzaTopping*.

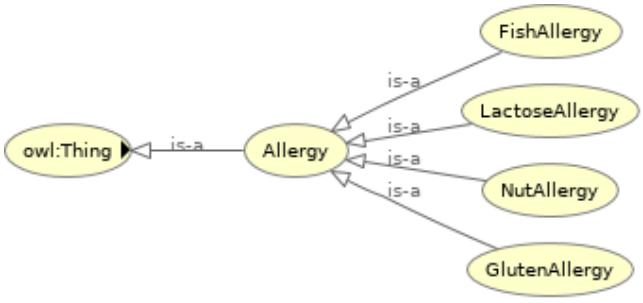

Fig. 1. Extension of pizza ontology – allergy and its subclasses

B. Using SWRL rules for inferences

One of purposes of using an ontology is to create a reusable domain representation. SWRL rules make excellent choice for further describing of the model data, thanks to the reasoning potential they provide. There are several reasoners capable of processing such rules (e.g. HermiT, FaCT++ and Pellet). In our solution, a fork of Pellet, Openllet [1], was chosen. A sample rule follows:

```
pizza:Consumer(?c)
^ pizza:hasAllergy(?c, ?a)
^ pizza:NutAllergy(?a)
^ pizza:eatsPizza(?c, ?p)
^ pizza:toppingPresent(?p, ?t)
^ pizza:NutTopping(?t)
-> pizza:hasNutRisk(?c, ?t)
```

Here, let *Consumer c* have an allergy *a* and suppose that *a* is a *NutAllergy*. In order to determine whether *Consumer c* can eat a pizza *p*, it's topping *t* has to be examined. Thus, if *t* is a *NutTopping*, it carries a risk (*hasNutRisk*) of triggering an allergy attack in *Consumer c*.

IV. Use of software agents

Recall that the purpose of proposed application is to allow groups of people to negotiate what food (here, which pizza) they can eat so that the smallest number of different pizzas is bought (to save money), while considering individual allergies. Let us now discuss how software agents can facilitate negotiations. The negotiation process consists of three stages: (i) broadcasting data set (list of available dishes), (ii) consulting with knowledge bases, and (iii) communicating, to exchange local knowledge between agents.

In the proposed approach, there are two types of agents – *Waiter* and *Consumer*. A single *Waiter* (with user interface as in Figure 2) sends list of available pizzas and waits for the final order from *Consumer* agents. *Consumer* agents are created at the beginning of application lifetime, and represent "hungry students". On creation they are provided with the information about user allergies, through interface, as in Figure 3.

Determining if a meal can be eaten is achieved through semantic reasoning. Basically, if one of pizza ingredients triggers an allergy then the whole pizza has potential of triggering an allergy. Reasoning is described in Section III-B.

A. Negotiation process

Process of establishing what to eat starts with the waiter (or employee of the restaurant) coming to the table where the clients ("hungry students") are waiting. Then the application is started, e.g. on a tablet (or a laptop). Next, the waiter specifies pizzas that are available within the promotion (moves them to the "available" list). Obviously, such list could be

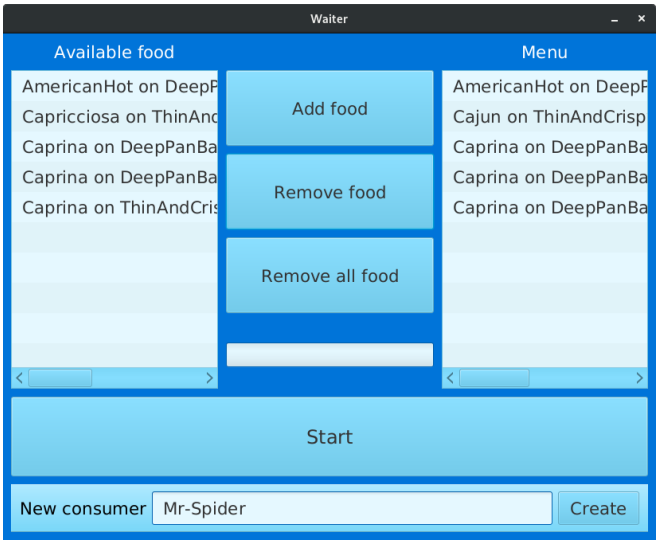

Fig. 2. *Waiter* agent window, allowing to compose menu, create new *Consumers* and initiate reasoning.

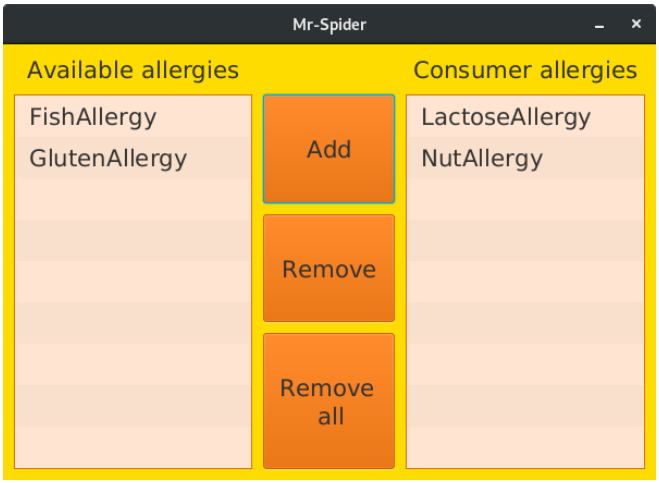

Fig. 3. *Consumer* agent window, allowing to select user allergies.

also uploaded from the restaurant server. Next, *Client* agents (representing users) are initialized and users add their allergies to each of them (see, Section IV). After these prerequisites are completed, autonomous negotiations, to determine the meal, begin.

The *Waiter* (an agent, not the actual person) broadcasts the "menu" to all of *Consumer* agents and waits for their response. Each *Consumer* uses knowledge about its user allergies to prune the menu (using semantic reasoning). If the menu becomes empty (i.e. there was nothing that could be eaten by the user, without triggering an allergy risk), *Consumer* informs the *Waiter* and other *Consumers* that it quits.

Otherwise, *Consumer* registers itself in the agent platform's *Yellow Pages Service*. The *YPS* allows agents to determine where they should send the messages – each *Consumer* queries *YPS* for the list of other *Consumers* that are to participate in negotiations of a given group.

Negotiations consist of *Consumer* selecting an acceptable pizza, and asking all other *Customers* if they agree. Other agents check if this pizza is acceptable and respond with this information. The initial *Consumer* collects all responses and if at least one of them was negative, removes the pizza from the list of potential dishes. In the meantime, it responds to the questions asked by other *Consumers*. Note that it is not necessary to reason on the ontology again – it is enough for the *Consumer* to check if given pizza is on the "safe food list". Negotiations stop when a common pizza is found or, when individual dietary requirements cannot be satisfied (see, Section VI).

V. Technical aspects

The core functionality of the proposed approach has been implemented. We have focused on agent negotiations and use of semantic reasoning. Obviously, the proposed approach has been developed to be implemented as a mobile application. However, technical issues introduced by placing agents on mobile devices and running application over the Internet were out of scope of current contribution. Keeping this in mind, let us now describe the key technical aspects of the implemented approach. Here, note that these choices would remain the same if the application was to run on mobile devices.

Waiter and *Customer* agents (and their interactions) have been implemented using Java Agent Development Framework (JADE). JADE provides programmer with an agent abstraction model, task execution mechanisms, peer to peer agent communication and a Yellow Pages Service.

To extract information from the ontology, Apache Jena was chosen. Jena provides support for ontologies built using RDF Schema and OWL. The underlying language, used for representing given ontology, is transparent when using Jena framework. Its interface is simple and intuitive. For instance, to get a reference to given class which exists in ontology, one has to use the following code:

```
OntClass consumerOntClass =
    ontModel.getOntClass(ONTOLOGY_NS +
    "Consumer");
```

where *ONTOLOGY_NS* is the namespace of given ontology and the method itself is invoked on the loaded ontology model. To create an individual of this class it is enough to call:

```
Individual consumerIndividual =
    ontModel.createIndividual(ONTOLOGY_NS +
    consumerName, consumerOntClass);
```

where the method is called on the loaded ontology and the *consumerOntClass* is a class from the model, extracted earlier.

Ontologies allow using a reasoner to derive non-trivial facts about modeled concepts. For this purpose, Jena exposes an inference API. For our application, Openllet [1] was selected. However, any other reasoner could have been used. Reasons behind this choice were: compatibility with Jena, full support for OWL 2, support for SWRL, and the fact that the original

Pellet is no longer an open-source project. To configure it, an ontology has to be loaded:

```
Model model =
    FileManager.get().loadModel(ONTOLOGY_PATH,
    "RDF/XML");
```

Here, the asserted model is loaded from the file with path *ONTOLOGY_PATH*, using Jena's *FileManager* static helper class instance. It was also instructed that the given file complies with the RDF format. Then, Openllet's specification has to be used when creating actual inferred ontology model:

```
import static PelletReasonerFactory.THE_SPEC;
// ...
OntModel ontModel =
    ModelFactory.createOntologyModel(THE_SPEC,
    model);
```

Again, we use static helper method of Jena's *ModelFactory*, providing it with Pellet's *OntModelSpec* (*PelletReasonerFactory.THE_SPEC*), which encapsulates description of components of the ontology model, including the reasoner. As a second argument we provide the *Model* that was loaded from the file. Although Jena's *OntModel* is a subclass of *Model* and they can be used in the similar way, it is the former that contains knowledge, inferred by the reasoner. This allows to easily distinguish between asserted an inferred knowledge.

User interface has been created in JavaFX, standardized alternative to Swing toolkit, equipped with many mechanisms used in modern programming like data binding or stylesheets.

In order to wrap all components, and provide dependency injection mechanisms, Spring Boot – a Spring's convention-over-configuration solution for creating easily runnable applications – has been chosen. It substantially decreases amount of configuration necessary to start Spring and facilitates creating easily-readable, modular and extensible code.

VI. Experimental verification

The proposed application has been thoroughly tested. Here, we report only two core scenarios.

A. When everything goes well

Let us start with an optimistic scenario. Suppose that the group of users does not have many dietary restrictions and may eat majority of available pizzas. The scenario proceeds as described in Section IV-A). The *Waiter* sends list of pizzas to all participating *Consumers*. They establish which pizzas are acceptable and start sending proposals to others (one to many communication), choosing pizzas from the list. Simultaneously, on the receiving end, they verify if received proposals are "OK".

Negotiations can become complicated, see Figure 4). Finally, if one of pizzas is accepted by everybody, agent that suggested it sends the decision to the *Waiter*. When the *Waiter* gets such information from any of the agents, it closes negotiations (as consensus has been reached) and displays results (see, Figure 5). Note that the *first* message that reaches the *Waiter* specifies pizza to be ordered. While it is possible that,

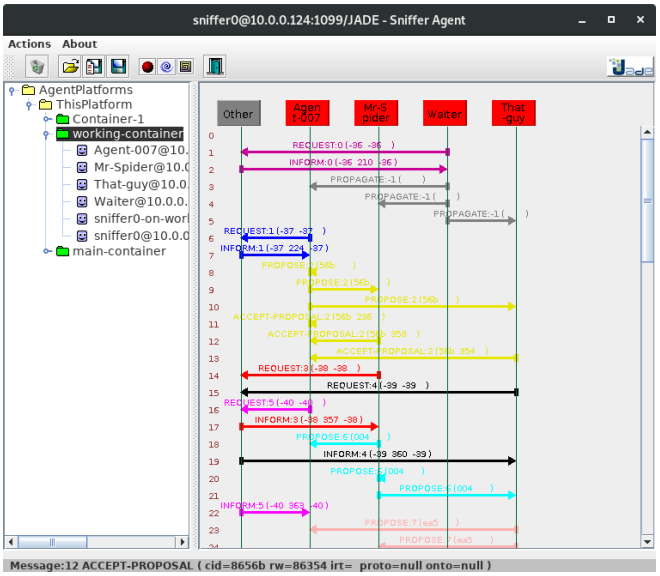

Fig. 4. JADE sniffer, showing communication between *Waiter* and *Consumer* agents. It can be easily seen how *Waiter* uses PROPAGATE message to distribute menu and *Consumers* use PROPOSE messages to suggest pizzas to other agents

Fig. 5. Result window, showing the order. We can see that it was chosen by consensus by all *Customer* agents.

while the first winning pizza is communicated to the *Waiter*, another pizza is found to be acceptable as well (negotiations have not been closed, yet), message informing that this pizza was selected is discarded, as the *Waiter* already has a winner for this group.

B. When NOT everything goes well

Now we will go through a pessimistic scenario, where the dietary restrictions prevent reaching consensus. The beginning of the negotiation process is the same. The *Waiter* sends list of available pizzas, *Customers* start proposing, accepting, rejecting. Let us now assume that there is (at least) one person

Fig. 6. Result window, showing the order consisting of two pizzas. A case when more than one pizza is chosen is possible only when the panic behavior occurs – in this case, one of the *Consumers* has triggered panic behavior and ordered extra pizza.

that has dietary requirements that introduce problems the case of the particular group. After negotiations including proposals, acceptances and rejections *Customer* agent, representing this person, is left with no pizza "OK" for the user (let us name this case: "panic behavior"). Here, *Customer* contacts the *Waiter* directly, and asks for a personalized meal. Moreover, it informs other *Customers* that it is out of negotiations. In this case, remaining *Customers* start negotiating anew. After *Waiter* receives information that remaining *Consumers* reached consensus, a result window appears, showing more than one pizza to be ordered (see Figure 6).

VII. Concluding remarks

Let us note first that the problem of making a group decision has already found many solutions in the human history. Dictatorship, democracy, consensus and many more proved to be effective in various situations and circumstances. Nevertheless, while the purpose of the project was not to find the most efficient way to decide what to eat together, the results appeared to be quite satisfying and far more effective than we have expected.

Moreover, what has been implemented in the prototype includes only very preliminary functionalities, while the extensibility and modularity of the proposed approach allows comfortable future development of the software to even more complicated and universal solution. Let us list some of the considered directions of development.

First, as indicated above, there exists *Food Ontology* that covers much larger domain and includes substantially larger number of rules. It will be included in the application. Note that it can be easily extended with rules covering additional allergies.

Second, in addition to food restrictions, user preferences will be added. Hence, not only unacceptable meals will be eliminated, but expressions like "I like that one more than the other" will be captured. Here, experiences from [13] will be applied. Specifically, weight will be added to dishes (as an annotation within the ontology) and used in negotiations.

Third, a planned technical upgrade is to migrate the application to a mobile solution, adding support for multiple clients, and perhaps even connecting complete strangers, with the goal of eating cheaper (Groupon-type solution). Here, possibility of integrating the proposed approach with personal assistants like Alexa, Cortana or Google Assistant will be also considered.

Finally, taking into account that the developed application will have food knowledge-base, combined with representation of allergies, it will be possible to naturally combine it with information about exercises to develop a more complex application facilitating all around fitness support. Here, obviously, ontology of fitness will have to be developed (or found and applied).

References

[1] Openllet, OWL2 reasoner in Java, built on top of Pellet. https://github.com/Galigator/openllet.
[2] Pizza ontology. protege.stanford.edu/ontologies/pizza/pizza.owl.
[3] Ahmed Al-Nazer, Tarek Helmy, and Muhammed Al-Mulhem. User's profile ontology-based semantic framework for personalized food and nutrition recommendation. In *ANT/SEIT*, 2014.
[4] A. Arwan, M. Sidiq, B. Priyambadha, H. Kristianto, and R. Sarno. Ontology and semantic matching for diabetic food recommendations. In *2013 International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 170–175, Oct 2013.
[5] B. V. Batlajery, M. Weal, A. Chapman, and L. Moreau. prfood: Ontology principles for provenance and risk in the food domain. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, pages 17–24, Jan 2018.
[6] Min Chen. Towards smart city: M2m communications with software agent intelligence. *Multimedia Tools and Applications*, 67(1):167–178, Nov 2013.
[7] D. Çelik, A. Elçi, R. Akçiçek, B. Gökçe, and P. Hürcan. A safety food consumption mobile system through semantic web technology. In *2014 IEEE 38th International Computer Software and Applications Conference Workshops*, pages 348–353, July 2014.
[8] Duygu Çelik. Foodwiki: Ontology-driven mobile safe food consumption system. *TheScientificWorldJournal*, 2015:475410, 07 2015.
[9] Ö. Ö. Ergün and B. Öztürk. An ontology based semantic representation for turkish cuisine. In *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4, May 2018.
[10] Xian Xing Zhang Ferras Hamad, Isaac Liu. Food discovery with uber eats: Building a query understanding engine. Technical report, Uber Eats, June 2018.
[11] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.
[12] W. Hu, L. Liu, and G. Feng. Consensus of linear multi-agent systems by distributed event-triggered strategy. *IEEE Transactions on Cybernetics*, 46(1):148–157, Jan 2016.
[13] M. Kruszyk, M. Ganzha, M. Gawinecki, and M. Paprzycki. Introducing collaborative filtering into an agent-based travel support system. In *2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*, pages 439–443, Nov 2007.
[14] Mladenka Vukmirovic Marcin Paprzycki Michał Szymczak, Maciej Gawinecki. Ontological reusability in state-of-the-art semantic languages. *PTI Press*, pages 129–142, 2006.
[15] Sascha Ossowski. *Agreement Technologies*. Springer, 2013.
[16] H. Su, M. Z. Q. Chen, J. Lam, and Z. Lin. Semi-global leader-following consensus of linear multi-agent systems with input saturation via low gain feedback. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(7):1881–1889, July 2013.
[17] Matthias Thimm. Strategic argumentation in multi-agent systems. *KI - Künstliche Intelligenz*, 28:159–168, 2014.