DOMINO – An Efficient Algorithm for Policy Definition and Processing in Distributed Environments Based on Atomic Data Structures

Joachim Zeiß, Peter Reichl, Jean-Marie Bonnin, Jürgen Dorn

DOMINO – An Efficient Algorithm for Policy Definition and Processing in Distributed Environments Based on Atomic Data Structures

Joachim Zeiß[1,2], Peter Reichl[2,3], Jean-Marie Bonnin[2], Jürgen Dorn[4]

[1] FTW Forschungszentrum Wien, Donaucitystr. 1, 1220 Vienna, Austria
[2] Télécom Bretagne, 2 Rue de la Châtaigneraie, 35510 Cesson-Sévigné, France
[3] University of Vienna, Währingerstr. 29, 1090 Vienna, Austria
[4] Vienna University of Technology, Favoritenstraße 9-11, 1040 Vienna, Austria
zeiss@ftw.at; {peter.reichl | jm.bonnin}@telecom-bretagne.eu;
juergen.dorn@ec.tuwien.ac.at

Abstract. While two decades of semantic web research so far have failed to fulfill the high initial promises and expectations, and the underlying quest for categorizing the world has led to increasing complexity and new levels of bureaucracy instead, in this paper we introduce a new concept for distributed reasoning aiming at simplicity, consistency and computational efficiency. To this end, we propose the DOMINO algorithm, which is based on term logic for realizing an efficient syllogism reasoner. This novel concept is illustrated and evaluated for a context experience sharing system and can be applied in distributed reasoning use cases integrating mobile devices.

Keywords: semantic policies, semantic data abstraction, rule definition, distributed reasoning, mobile reasoning, artificial intelligence

1 Introduction and Motivation

Following Tim Berners-Lee's vision that "a new form of Web content that is meaningful to computers will unleash a revolution of new possibilities" [5], significant research efforts have been undertaken to realize his idea of a "Semantic Web", focusing on converting raw data into information and turning search queries into question-answer use cases between human and machine. At the same time, Web service architects have integrated semantic web technologies into their API definitions, e.g. WSDL files for finding and collaborating automatically with Web services inside the cloud.
However, after two decades of semantic web research, we have to observe that the initial dreams have not come true so far, and have rather turned into increasing complexity and creating yet another level of bureaucracy instead, eventually causing more problems than they intended to solve. Indeed, the somewhat exaggerated desire for categorizing the world has led to a dictatorship of ontologies versus the democracy of folksonomies (currently blooming in Web 2.0), leaving key concepts of artificial intelligence, formal semantics and logic programming unexploited.

In this paper, we argue that, instead of e.g. introducing yet another (graphical) tool or a novel method or procedure for complexity reduction, we better use the powerful rich set of features in a simple way by introducing machine-readable rules which allow devices to act autonomously on behalf of their users. This enables a new way of distributed reasoning with software executing decisions based on user defined policies. More precisely, we aim at simplicity and intuitiveness by allowing users to verbally define their intents with (almost) no predefined vocabulary, only minimal semantic and syntactic requirements, and without the need to explicitly distinguish between conditions and conclusions. At the same time, detecting contradictions and contraries becomes part of the reasoning process and is performed directly while defining policies. Finally, for a given set of policy definitions, the reasoner should be able to run in a browser with reasonable response time, thus guaranteeing sufficient performance.
In order to achieve these goals, we have designed and implemented a reasoning system and language based on (Aristotelian) term logic [6] which has experienced a revival since the 1970s, mainly due to the constant criticism that predicate logic is unnatural as its syntax does not follow the one used in people's everyday reasoning. Also in the field of artificial intelligence it has gained momentum, e.g. as probabilistic term logic, "Non-Axiomatic Reasoning System" (NARS) or "OpenCog" [7][8].
The remainder of this paper is structured as follows: in Section 2, we provide essential background on term logic and discuss related work. Section 3 presents the DOMINO algorithm for semantic reasoning. Section 4 discusses the application of this approach for a specific use case, i.e. the context experience sharing system KRAMER [1]. Section 5 concludes the paper with a brief summary and outlook on future work.

2 Background and Related Work

2.1 Syllogisms and Term-based Logic

The DOMINO reasoning system as described later is based on Aristotelian logic which is composed of the three doctrines of categorical terms, propositions and syllogisms. Here, (categorical) terms are universal concepts like "man", "mammal", "human", "animal", etc. In the following description we will denote these terms with small letters x, y, z etc. (Categorical) propositions are formal sentences combining two different terms, e.g. x and y. There exist four different types of propositions:

- $A(x,y)$: All x are y; e.g.: All man are mammal.
- $E(x,y)$: No x are y; e.g.: No mammal are birds.
- $I(x,y)$: Some x are y; e.g.: Some Austrians are musicians.
- $O(x,y)$: Some x are not y; e.g.: Some mammal are not vegetarians

According to the above propositions, we call the x term "subject" and the y term "predicate". A proposition is called "premise" when it is applied to a syllogism for concluding a new proposition. Finally, "(categorical) syllogisms" are rules which allow inferring from a set of propositions (premises) additional propositions (conclusions). For instance, from the premises "All swan are white: $A(s,w)$" and "No white are raven: $E(w,r)$", the Aristotelian system will conclude "No swan are raven: $E(s,r)$". The corresponding rule is denoted as "$A(x,y)$, $E(y,z) \rightarrow E(x,z)$". Aristotle has defined 14 different such rules (the "συλλογισμοι"), see [10] for comments on the proofs.

2.2 Computational Syllogism

For implementing these rules, when comparing any two premises, first their terms *x*, *y*, *z* are compared. If any of the terms in one premise matches with a term in the other premise, concluding a new proposition is possible. To this end, it is checked whether the types of the premises match one of the mentioned rules; if yes, the rule is executed to produce a new proposition, which itself can serve as an additional premise, and so on. If a new proposition is already included in the set of existing premises, it is not added as a new premise; the iteration process ends if no new propositions are found.
While [6] describes a computational syllogism web page implemented on a php server, we have implemented our own DOMINO reasoner in pure Javascript. The implementation has a small footprint, does not depend on server side computing and can be run in any browser (even as local html file) or used as a library for W3C or other HTML widgets or apps for the WAC [9] runtime, the Firefox OS or Google Chrome. Our Javascript code allows detecting contradictions, contraries and sub-contraries to identify misleading ambiguities in premise definitions or conclusions. For the usage of our reasoner, triggers and their namespaces are defined as Javascript functions which are executed if namespace and trigger name match the terms in a concluded proposition.

3 DOMINO

3.1 Basic Idea of the DOMINO Algorithm

We now introduce a novel method for reasoning over data for automated decision making. We call it DOMINO as its basic functionality reminds of the well-known social game where bricks with two numbers on each side are connected to a chain where bricks with equal numbers lay side by side (see Fig. 1). In our DOMINO system, the bricks correspond to term logic premises, where each premise is composed of two terms (subject and predicate). Like with the domino game, bricks are linked according to syllogistic rules [10], which can be computed by software [6].
In principle, DOMINO may use the full set of syllogism rules. However, just like with Horn clauses [12], for decision-making based on positive affirmation using a single rule is sufficient, and we therefore restrict ourselves to a well-known rule which is traditionally called “Barbara” and chains propositions which are of type ”all” and have the term x in common: $A(p,x),A(x,q)\rightarrow A(p,q)$.
For computation purposes, the terms are represented as strings, and DOMINO checks whether they are matching. Note that matching of two expressions can be defined in different ways, e.g numbers checked for equality or matching strings (as used in [6]). In DOMINO, we opt for flexibility and allow numbers to be compared, or being matched with intervals or “<” and “>” relations, while strings could be matched using regular expressions. This also allows pattern matching, a feature no other reasoner of this kind provides as of today. Other matching strategies are conceivable, e.g. for matching date/time expressions and periods. DOMINO also allows executions on conclusion results validating the subject and predicate of a concluded proposition for an existing function to be invoked. The subject of a proposition would hold for the namespace in which a function with the name of the predicate would be executed. Such a function may even inject arbitrarily created new propositions for further com-

putation of conclusions. As no duplication of premises is possible, each trigger function routine would be called exactly once. By linking DOMINO bricks all of which include execution triggers, a sequence of function executions can be defined.
Alternatively, one does not need to trigger functionality during the computation: instead, once as all possible conclusions have been obtained, one could search for particular expressions and take decisions based on the filtering results. The reasoning process would be performed only once and its results are kept in a cache. The conclusion process is finished if no new (unknown) proposition can be deduced, and the list of all given and conducted proposition is the result set of the reasoning.
Summarizing, DOMINO is a computational syllogism system using a subset of term logic figures and conditions to reason over data represented as term logic premises. The basic exercise of concluding new information out of existing premises is performed by linking the term of one premise with a term of another premise. Two premises may be linked if they have a common term, which computationally is verified by checking if the terms in string format are equal. DOMINO extends this notion of identity also to checking number range and regular expressions, as basically any kind of rule or procedure could be used to declare two terms as being equal. Also multistep comparison is possible, where initially high accuracy strict checking is applied, and if this does not bring the desired result, in a second run comparison checks can be relaxed, to produce a match according to the "Barbara" rule. Another novelty added to DOMINO is the possibility to trigger execution of software routines if new premises are concluded. Note that, while Aristotelian syllogisms are concluding on categories, not on individuals, for pragmatic reasons we do not distinguish between categories and individuals, classes or instances in DOMINO. Thus, the syllogistic terms may lose their philosophical meaning, the validity of conclusions, however, remains.

3.2 Formal Description

As we only use premises of type $A(p,q)$ in the sense of "all p are q", we abbreviate this statement to be a simple data tuple t made of the two values p and q as follows:

$$A(p,q) \quad \Leftrightarrow \quad t = (p,q) \tag{1}$$

Next, we define the conjunction operator " $\oplus$ " for the "Barbara" syllogism:

$$A(p,x), A(x,q) \rightarrow A(p,q) \quad \Leftrightarrow \quad (p,x) \oplus (x,q) = (p,q) \tag{2}$$

Then, let K_0 define the initial set of all predefined tuples where none of the tuples in the set has identical p and q values (to avoid indicating mere tautologies which are useless for producing new tuples). This initial set of tuples represents the entire data (facts and rules) of our knowledge base, i.e. all data or information which is already known before starting the reasoning process:

$$K_0 = \left\{ (p_x, q_x) \middle| p_x \neq q_x; x \in X \right\} \tag{3}$$

where X is the set of indices related to the tuples.
Finally, we define the cardinality of a tuple as being 1 for elements of K_0:

$$|t| = |(p,q)| = 1 \quad \text{if} \quad (p,q) \in K_0 \tag{4}$$

$$|t_j \oplus t_k| = |t_j| + |t_k| \quad \text{for all } t_j, t_k \in K_{i-1}; \quad i = 1, 2, \ldots \tag{5}$$

The cardinality of a derived tuple equals the sum of the cardinality of the two tuples on which the derivation is based. As no tuple can be represented twice, always the shortest way to produce tuple t_k applies, and the cardinality is thus uniquely defined.

We now define formally the DOMINO algorithm. Starting with K_0 as described in (3), step i of the algorithm uses the elements of K_{i-1} to produce a new set of tuples:

$$B_i = \left\{ (p_y, q_y) \middle| \exists q_x : (p_y, q_x) \in K_{i-1} \wedge (q_x, q_y) \in K_{i-1} \wedge (p_y, q_y) \notin K_{i-1}; x, y \in X \right\} \tag{6}$$

Hence, B_i represents all the new information inferred based on set K_{i-1}, therefore

$$K_i = K_{i-1} \cup B_i \tag{7}$$

and so on for $i = 1, 2, 3, \ldots$, until $B_i = \varnothing$. The algorithm converges as K_0 is a finite set and no tuple can be represented twice. Note that the cardinality of a tuple represents its importance in terms of the new information, i.e. newly inferred tuples are considered more important as already existing information. In general, the cardinality n of a tuple is rooted in chaining n-1 other tuples, hence the larger n the more previous information has been condensed to this new piece of information, indicating as well a higher computational effort.

3.3 Evaluation Against Triple-Based Reasoning

Defining Horn clauses with DOMINO

With traditional inferring systems, Horn clauses are used to define rules or formulae for computing conclusions. A definite clause has exactly one positive literal, while a clause with no negative literal is called a fact and in propositional logic is expressed as $\neg p \vee \neg q \vee \ldots \vee \neg t \vee u$, which can also be written in the form of an implication, i.e. $(p \wedge q \wedge \ldots \wedge t) \rightarrow u$ (meaning: if *p*, *q*, ..., *t* are all true then *u* must be true as well).
Declaration of those rules, e.g. for policy definitions, is based on implications, i.e. "IF-THEN" declarations, quantification and variables. Decision-making is declared by means of logic programming in conjunction with semantic web ontologies. With DOMINO, it is not necessary to define quantifications, and IF-THEN statements become obsolete, as declaring a decision as well as executing a decision process by the reasoner is entirely based on linking DOMINO bricks (like with the social game).
While DOMINO is not capable of quantifying variables, it is possible to define rules the same way it is done for Horn clauses in propositional logic, and any Horn clause construct can be expressed as a DOMINO sequence, see Fig. 1. Basically, DOMINO expresses implications by splitting conditions when they are defined, and recombining them when they are evaluated with actual data.

For example, a single condition for buying a product when the price matches 99.90 ("*price = 99.9 ⇒ buy*") reads with DOMINO as "{(*trigger*, *price*), (*99.9*,*buy*)}". When only these tuples are present, no conclusions can be drawn, but if another tuple is added (e.g. by browsing through a database or checking the annotated web page of an online shop) which says "(*price*, *99.9*)", applying the conjunction results in (*price*, *99.9*) ⊕ (*99.9*, *buy*) = (*price*, *buy*), and we can conclude (*trigger*, *price*) ⊕ (*price*, *buy*) = (*trigger*, *buy*). Hence, upon conclusion of a new tuple, DOMINO checks the namespace *"trigger"* for a function called *"buy"* and, if present, will execute it.

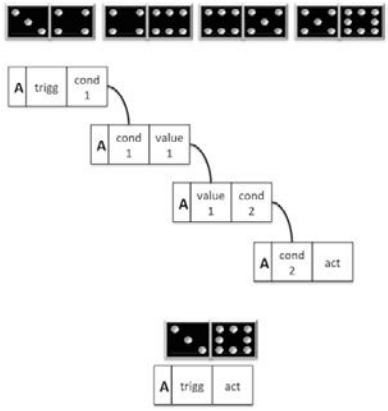

Figure 1: Domino decision sequence

The same principle works for an arbitrary number of conditions. Consider for instance the Horn clause " $p = x \wedge q = y \wedge r = z \rightarrow \text{action1}, \text{action2}$ ". This rule will be represented as the following set of DOMINO tuples:

$$R = \{(\text{trigger}, p), (x, q), (y, r), (z, \text{action1}), (\text{action1}, \text{action2})\} \tag{8}$$

Upon retrieval of the data set $D = \{(p, x), (q, y), (r, z)\}$, we will apply $R \cup D$ in order to produce the result set

$$\{(\text{trigger}, \text{action1}), (\text{trigger}, \text{action2})\}, \tag{9}$$

which will execute the functions "action1" and "action2" in namespace "trigger".
Finally, we would also like to demonstrate how to model Horn clauses for a rule like *"if p and q have the same value then do action"*. The Horn clause reads

$$p = ?\,val \wedge q = ?\,val \Rightarrow \text{action} \tag{10}$$

The corresponding DOMINO tuples look like

$$\{(\text{trigger}, p), (p, p\_val), (q\_val, q), (q, \text{action})\} \tag{11}$$

Eventually, the tuples specified in (11) would conclude to (trigger, action) for the data tuples {(*x*, *q_val*), (*p_val*, *x*)}.

Declaring decision trees with DOMINO

Although separated in the examples so far, in DOMINO there is no difference between data declarations and defining conditions or implications – it is all about declaring pairs of terms. This generalization or abstraction of facts and rules into a common format allows declaring also complex decision patterns, for example decision trees.

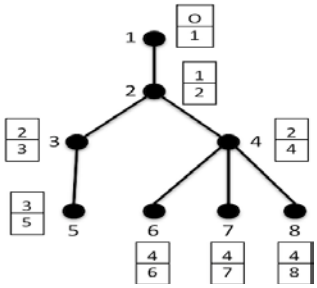

Figure 2: Domino decision tree

The example given in Figure 2 maps to the following DOMINO sequences and can be proved by recursively applying the conjunction operator:

(0,1) ⊕ (1,2) = (0,2), (0,2) ⊕ (2,3) = (0,3), (0,3) ⊕ (3,5) = (0,5).
(0,1) ⊕ (1,2) = (0,2), (0,2) ⊕ (2,4) = (0,4), (0,4) ⊕ (4,6) = (0,6).
(0,1) ⊕ (1,2) = (0,2), (0,2) ⊕ (2,4) = (0,4), (0,4) ⊕ (4,7) = (0,7).
(0,1) ⊕ (1,2) = (0,2), (0,2) ⊕ (2,4) = (0,4), (0,4) ⊕ (4,8) = (0,8).

In fact, not only trees but any kind of graph topology can be described. If, e.g. by accident, a graph is created which contains loops, the reasoning process will still come to an end, as each derived tuple is added to the set of existing tuples for reiteration if and only if it does not already exist.

3.4 Advantages over Conventional Semantic Reasoning

The advantages of DOMINO over existing reasoning systems based on propositional or first order logic are manifold. As it does rely on term based calculation for its reasoning instead of logical operations, it allows for

- human readable data and rule definitions;
- easy integration with tag based systems and hence relating messages and media of different topics down the conclusion chain and defining their informational distance using cardinalities (for example, given T1 containing tags #x and #y, T2 containing tags #y and #z and T3 containing #z and #w, T1 on topic #x would be associated to T3 about topic #w because of T2);
- easy integration of pattern matching or any other equality concept (e.g. range checks, probability or similarity statements) into the reasoning algorithm.

Furthermore, DOMINO allows for closing knowledge gaps by asking the user simple yes/no questions: tuples that might be worth being combined, as they have a certain cardinality value, can be presented to the user asking if the second value of one tuple

can be related to the first value of another tuple. In contrast to reasoners based on predicate logic, DOMINO avoids running into loops, which is especially important when it comes to allowing user-defined rules and reasoning on mobile devices. Finally, DOMINO does not require any special set up or topology descriptions for distributed evaluation of rules. Instead, distributed reasoning becomes as simple as putting tuples together from any data sources. As DOMINO's knowledge base solely consists of tuples and does not distinct between rules and facts, rules may change dynamically based on new data arriving or additional data sources tied in the reasoning process.

3.5 Readability of DOMINO Tuples

The human readability of DOMINO tuples relies on the usage of terms and the linking of pairs of terms through the "Barbara" syllogism which is closely related to natural thinking (cf. G. Evan's homophonic theories for semantic investigations [17]).
Consider for instance the information "My availability status *is* busy" which translates to (my_availability_status, busy). This transition is typical for patterns à la "all a are b", "this is that" or "c has d", e.g. "My house has a door.", "The door is green.", etc.
Verbally expressed rules in DOMINO are defined as follows: "Check the eyes. Are they blue? Check hair. Is it blonde? Ask for a date. Check humor. Is it good? Marry this person!" will become (check, eyes), (blue, hair), (blonde, ask_for_date), (ask_for_date, humor) (good, marry!). With data tuples (hair, blonde) and (eyes, blue) the check would result in asking for a date, and with an additional tuple (humor, good), the happy end becomes almost inevitable.
Hence, a tuple (1) can be a name-value pair, (2) when inferred/produced it can be a question and answer, and (3) for a rule the left side entry is the expected value of a previous parameter and the right side entry the name of the next parameter to evaluate.
Alternatively, hashing the terms of a sentence can produce tuples, e.g.: "Today #Peter was very #happy and the #weather was #nice" leads to (peter, happy), (weather, nice). When interested in special properties, this can be filtered and combined to new tuples.
Finally, in formal logic, the word "not" is an operator, while DOMINO does not imply the concept of negation. The semantic relevance of "not" is assigned to the person using it: "Should I buy the house? The wall is green, the roof is pink, do not buy it" is modeled as (should_I_buy_the_house, wall), (green, roof), (pink, do_not_buy_it). With (wall, green) and (roof, pink), DOMINO suggests (should_I_buy_the_house, do_not_buy_the_house), where the way the word "not" is interpreted is up to the user.

4 Application Example: KRAMER

In [11] a phonebook application has been introduced that provides users with context information about their contacts and notifies them in important situations. This application deploys the so-called KRAMER system, which uses an abstraction process to derive situations where to perform the afore-mentioned notifications by identifying semantic generalizations of user-defined situations that are similar to each another [1]. In this chapter we describe how DOMINO can be used to perfectly describe semantic information for decision making in such an environment.

Semantic similarity is defined by two conditions [1]: (1) the conceptual graphs of similar situations have to match, and (2) there needs to be a close semantic distance between each other. Both can be easily evaluated with DOMINO by first checking if tuples exist where start nodes and end nodes are comparable for all the graphs. Secondly, due to the substitution process during DOMINO reasoning, these (start-node, end-node) tuples will also be generated if sub-graphs are added or if nodes are linked to each other via several paths. The cardinality of those tuples is used to compare the similarity of these paths.

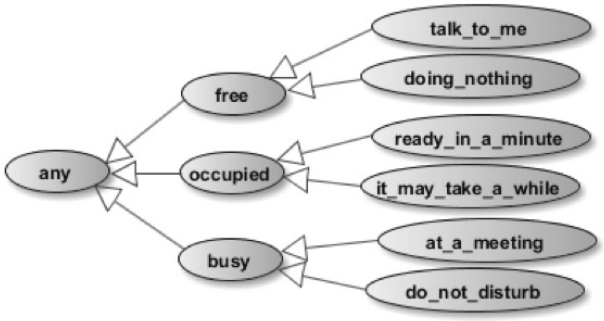

Figure 3: KRAMER availability taxonomy (source [1])

Fig. 3 depicts the KRAMER availability taxonomy. With DOMINO, the is-a relationship taxonomy is described in the form of A-type propositions, such that the relationship *"x is-a y"* is expressed by the proposition *"all y are x"*, giving the DOMINO tuple (*y*,*x*). Hence, the relationships denoted in Fig. 3 result in the DOMINO sets:

A_1 = {*(any_av, availability), (free, any_av), (occupied, any_av), (busy, any_av), (talk_to_me, free), (doing_nothing, free), (ready_in_a_minute, occupied), (it_may_take_a_while, occupied), (at_a_meeting, busy), (do_not_disturb, busy)*}

Applying the DOMINO algorithm produces the following tuples with cardinality 2:

A_2 = {*(at_a_meeting, any_av), (do_not_disturb, any_av),(ready_in_a_minute, any_av), (it_may_take_a_while, any_av), (talk_to_me, any_av), (doing_nothing, any_av), (free, availability), (occupied, availability), (busy, availability)*}

After one more iteration, the following tuples with cardinality of 3 are produced:

A_3 = {*(at_a_meeting, availability), (do_not_disturb, availability), (ready_in_a_minute, availability), (it_may_take_a_while, availability), (talk_to_me, availability), (doing_nothing, availability)*}

For the location taxonomy, the following tuples exist:

L_1 = {*(any_loc, location), (bistro, any_loc), (bar, any_loc), (office, any_loc), (downtown, any_loc)*}

We now apply the DOMINO algorithm and get

L_2 = {*(bistro, location),(bar, location),(office, location),(downtown, location)*}

The third taxonomy is about relations:

R_1 = {*(any_rel, relation), (mother, any_rel), (parent, any_rel),(boss, any_rel)*}

After applying DOMINO yet for another time, we end up with

R_2 = {*(mother, relation), (parent, relation),(boss, relation)*}

Eventually, we have derived the three taxonomy sets:

$A = A_1 \cup A_2 \cup A_3$; $L = L_1 \cup L_2$; $R = R_1 \cup R_2$

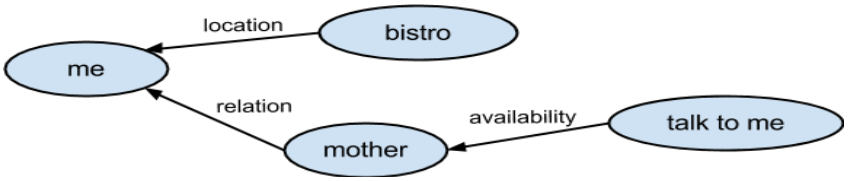

Figure 4: Kramer sample situation concept (source [1])

According to Fig. 4, the concept suggests that the location of the user and the availability of the related contact person should be of interest. In terms of DOMINO:

C = {(me, my_availability), (any_av, my_location), (location, my_relation), (relation, con_location), (any_loc, con_availability)}

Note that we introduced the identifiers *location* from the Location taxonomy set *L* and *relation* from the Relation taxonomy set *R*. This allows for any value under *location* and *relation* in their related taxonomy trees to be valid.
Expected tuples to be produced are: (me, *location*) and (me, *availability*). If these tuples are produced, the sample situations below can be considered to be similar to the concept. These two tuples also express that for the given situation: "I am at location: *location* and my contact is available for: *availability*".

	me		**contact**		
	location	**availability**	**relation**	**location**	**availability**
1	bistro	any	mother	any	talk to me
2	bistro	any	parent	any	free
3	bar	any	mother	any	doing nothing
4	bistro	any	mother	downtown	free
5	office	any	boss	any	busy
6	bar	any	mother	any	it may take a while

Table 1: Example contact situations (source [1])

Taking the exemplary contact situations listed in table 1 [1] gives the DOMINO sets:

S_1 = {(my_availability, *any_av*), (my_location, *bistro*) (my_relation, *mother*), (con_location, *any_loc*), (con_availability, *talk_to_me*)}

S_2 = {(my_availability, *any_av*), (my_location, *bistro*) (my_relation, *parent*), (con_location, *any_loc*),(con_availability, *free*)}

S_3 = {(my_availability, any_av), (my_location, bar) (my_relation, mother), (con_location, any_loc), (con_availability, doing_nth)}

S_4 = {(my_availability, *any_av*), (my_location, *bistro*) (my_relation, *mother*), (con_location, *downtown*), (con_availability, *free*)}

S_5 = {(my_availability, *any_av*), (my_location, *office*) (my_relation, *boss*), (con_location, *any_loc*), (con_availability, *busy*)}

S_6 = {(my_availability, *any_av*), (my_location, *bar*) (my_relation, *mother*), (con_location, *any_loc*),(con_availability, *it_may_take_a_while*)}

Now we apply: $S_i \cup C \cup L \cup R$ for all the samples, which adds following start-node, stop-node tuples mentioned above to the following situation sample sets:

S_1 : (me, bistro), (me, talk_to_me)
S_2 : (me, bistro), (me, free)
S_3 : (me, bar), (me, doing nothing)
S_4 : (me, bistro), (me, downtown)
S_5 : (me, office), (me, busy)
S_6 : (me, bar), (me, it_may_take_a_while)

Observe that S1, S2, S3, S5 and S6 match the concept, whereas S4 does not. The difference becomes more obvious when we also add the availability set A to all S:

S_1 : (me, location), (me, availability) // also (me, free)
S_2 : (me, location), (me, availability)
S_3 : (me, location), (me, availability) // also (me, free)
S_4 : (me, location)
S_5 : (me, location), (me, availability)
S_6 : (me, location), (me, availability) // also (me, occupied)

With this final evaluation we come to the same and potentially more granular grouping of situation as the KRAMER system shown in table 2.

Gr.	#	me	contact	availability	
I	1	bistro	mother	talk to me	
	2	bistro	parent	free	
	3	bar	mother	doing nothing	
	5	office	boss	busy	
	6	bar	mother	it may take a while	
II	4	bistro	mother	free	downtown

Table 2: Kramer semantic grouping (source [1])

5 Conclusions and Outlook

In this paper we have introduced a new concept for distributed reasoning in mobile networks. Based on term logic, we aim at simplicity, consistency, and computing efficiency. For easy adaptation by users in mobile networks we intend to achieve simplicity and intuitiveness by allowing them to verbally define their intents. We provided a formal description of the algorithm and demonstrated its capability by remodeling standard proposition logic and decision graph concepts. We finally illustrated and evaluated this concept for a context experience sharing system targeted for mobile applications.

Summarizing, the DOMINO concept is a novel approach for automated decision making in a distributed mobile environment. The advantages over traditional semantic web or first order logic programming approaches are its simplicity, flexibility and adaptability for verbal human reasoning. As the system is designed for end users, as part of our future work we plan to perform a user study and compare the results with related work in [4, 14-16] to show the advantages over predicate logic based systems. We will investigate a simplified modeling of conditional variables and the interpretation of tuple cardinalities as semantic similarity indicators and quantification of information quality. Beyond that, current and future work uses the DOMINO algorithm for various other applications (like, e.g., Location-Aware Campaigning).

References

1. Szczcerbak, M., Bouabdallah, A., Toutain, F., Bonnin, J.-M.: Generalizing Contextual Situations. Proc. IEEE ICSC 2012, pp. 293-301, Palermo, Italy, Sept. 2012.
2. Bouabdallah, A.; Toutain, F.; Szczerbak, M.; Bonnin, J.: On the benefits of a network-centric implementation for context-aware telecom services. Proc. ICIN 2011
3. Zeiss, J., Zhdanova, A.V., Gabner, R., Bessler, S.: Semantic Policy Aware System Description. FTW Technical Report, FTW-TR-2007-010, September 2007.
4. Zhdanova, A.V., Zeiss, J., Dantcheva, A., Gabner, R., Bessler, S.: A Semantic Policy Management Environment for End-Users and its Empirical Study. In: Schaffert, S., et al. (eds.): Networked Knowledge - Networked Media: Integrating Knowledge Management, New Media Technologies and Semantic Systems, pp. 249–268, Springer 2009.
5. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Feature Article, Scientific American, May 2001.
6. Glashoff, K.: Computational Aristotelian Term Logic: Introduction. http://webapp5.rrz.uni-hamburg.de/syllogism/aristotelianlogic/ introduction.html
7. Wang, P.: Non-Axiomatic Reasoning System - Exploring the essence of intelligence. PhD thesis, Indiana University. http://www.cogsci.indiana.edu/farg/peiwang/papers.html, 1995.
8. Opencog: General Intelligence via Cognitive Synergy. http://opencog. org/theory/
9. Zeiss, J., Günther, P., Davies, M.: The Role of WAC in the Mobile Apps Ecosystem. Bookchapter, InTech, 2012.
10. Cohen, M.: Aristotle's Syllogistic. URL: http://faculty.washington.edu/smchen/433/ Syllogistic.pdf.
11. Szczerbak, M., Toutain, L., Bouabdallah, A., Bonnin, J.-M.: Collaborative context experience in a phonebook. Inter-Clouds and Collective Intelligence Workshop at IEEE AINA 2012, pp. 1275-1281, 2012.
12. Horn, A.: On sentences which are true of direct unions of algebras. Journal of Symbolic Logic, 16, 14–21, 1951.
13. Zeiss, J., Zhdanova, A.V., Gabner, R., Bessler, S.: Semantic Policy Aware System Description. FTW Technical Report, FTW-TR-2007-010 (2007), September 2007.
14. Zeiss, J., Gabner, R., Zhdanova, A.V., Bessler, S.: A Semantic Policy Management Environment for End-Users. Proc. I-Semantics 2008, Graz, Austria, 2008.
15. Bessler, S., Zeiss, J.: Semantic Modeling of Policies for Context-aware Services. In: World Wireless Research Forum (WWRF), 17th Meeting , Heidelberg, November 2006.
16. Zeiss, J., Sanchez, L., Bessler, S.: Policy-driven Formation of Federations between Personal Networks. 16th IST Mobile and Wireless Communications Summit, Budapest, July 2007.
17. Evans, G.: Pronouns, Quantifiers, and Relative Clauses (I), Canadian Journal of Philosophy vii, pp. 467–536, 1977.