Curious Cat – Mobile, Context-Aware Conversational Crowdsourcing Knowledge Acquisition

LUKA BRADEŠKO, Jožef Stefan Institute and Jožef Stefan Postgraduate School, Ljubljana Slovenia
MICHAEL WITBROCK, IBM Thomas J Watson Research Center, NY, USA
JANEZ STARC, Jožef Stefan Institute and Jožef Stefan Postgraduate School, Ljubljana Slovenia
ZALA HERGA, Jožef Stefan Institute and Jožef Stefan Postgraduate School, Ljubljana Slovenia
MARKO GROBELNIK, Jožef Stefan Institute, Ljubljana Slovenia
DUNJA MLADENIČ, Jožef Stefan Institute and Jožef Stefan Postgraduate School, Ljubljana Slovenia

Scaled acquisition of high quality structured knowledge has been a longstanding goal of Artificial Intelligence research. Recent advances in crowdsourcing, the sheer number of internet and mobile users and the commercial availability of supporting platforms offer new tools for knowledge acquisition. This paper applies context aware knowledge acquisition that simultaneously satisfies users’ immediate information needs while extending its own knowledge using crowdsourcing. The focus is on knowledge acquisition on a mobile device, which makes the approach practical and scalable; in this context, we propose and implement a new KA approach that exploits an existing knowledge base to drive the KA process, communicate with the right people, and check for consistency of the user-provided answers. We tested the viability of the approach in experiments using our platform with real users around the world, and an existing large source of common sense background knowledge. These experiments show that the approach is promising: the knowledge is estimated to be true and useful for users 95% of the time. Using context to proactively drive knowledge acquisition increased engagement and effectiveness (the number of new assertions/day/user increased for 175%). Using pre-existing and newly acquired knowledge also proved beneficial.

- Information systems → Information systems applications → Spatial-temporal systems → Location based services → Information systems → World Wide Web → Web searching and information discovery → Personalization → World Wide Web → Web applications → Crowdsourcing → Answer ranking → Information retrieval → Document representation → Ontologies → Theory of computation → Logic → Verification by model checking, Automated reasoning → Information systems → Data management systems → Graph-based database models → Hierarchical data models → Human-centered computing → HCI → Interaction paradigms → Natural language interfaces → Human-centered computing → Interaction design → Interaction design process and methods → Activity centered design, Contextual design → Human-centered → Collaborative and social computing → Collaborative and social computing theory, concepts and paradigms → Social content sharing, Collaborative content creating → Human-centered → Ubiquitous and mobile computing → Ubiquitous and mobile devices → Smartphones → Computing methodologies → Artificial intelligence → Natural language processing → Natural language generation → Computing methodologies → Artificial intelligence → Knowledge representation and reasoning → Ontology engineering.

Author’s addresses: Luka Bradeško; Janez Starc; Dunja Mladenič; Marko Grobelnik, Artificial Intelligence Laboratory, Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia; Michael Witbrock, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA.

Permission to make digital or hardcopies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credits permitted.

To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2010 ACM 1539-9087/2010/03-ART39 $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000

ACM Transactions on xxxxxxxx, Vol. xx, No. x, Article xx, Publication date: Month YYYY
INTRODUCTION

Mobile devices can be seen as tools for satisfying their users’ information and communication needs. With the growing number of mobile users and technical developments of the offered mobile services, context aware services are becoming very common. Focusing on mobile context aware conversation, the context of the user can be obtained from sensors on the user’s mobile device as well as directly from conversation with the user and from the past user behavior. We address the problem of responding to the user’s information needs from the perspective of a knowledge-driven process developed around crowdsourcing Knowledge Acquisition (KA). In this setting obtaining and formalizing the right knowledge is crucial. This knowledge is, of course, acquired through some kind of a KA process, which can be manual, automatic or semi-automatic.

Knowledge acquisition using an appropriate representation and subsequent knowledge maintenance, are two of the fundamental and as-yet not-completely-solved challenges of Artificial Intelligence (AI). Knowledge in suitable for machine reasoning and, credible knowledge is still very expensive to obtain and to maintain. This is becoming increasingly obvious, with the rise of chat-bots and other conversational agents and AI assistants. The most developed of these (e.g., Siri, Cortana, Google Now, Alexa), are backed by huge financial support from the companies that produced them; the lesser-known ones still required several person-years of effort by individuals (e.g., Wilcox’s bots [1], etc.) or smaller companies (e.g., Josh AI - www.josh.ai). Although these systems may use machine learning and statistical approaches, a substantial portion of their production effort lies in knowledge acquisition, which is sometimes hidden in (hand-) coded rules for request/response patterns and corresponding actions. The proposed novel approach to context aware crowdsourcing knowledge acquisition implemented in our system Curious Cat, moves knowledge acquisition towards the point where it will no longer be a bottleneck for building knowledge-dependent systems. This is in particular achieved by:

• Looking at KA as a part of the final goal, rather than as a means to an end. Thus, we propose to incorporate KA into the end system as a part of its natural interaction.
• Building on the top of existing and newly acquired knowledge simultaneously, and using it to understand the user’s context and automatically drive the KA process deeper and deeper through this usage. This is done by automatically identifying missing pieces of knowledge and then deliberately trying to get these missing elements through directed crowdsourcing and reasoning. Our contributions in this paper are summarized as follows:

1. A novel knowledge acquisition approach that uses logical inference over context and prior knowledge to automatically construct precisely targeted natural language crowdsourcing tasks (“questions”) for the right audience at the right moment. The process that drives knowledge acquisition is, at the same time, more directly addressing the user’s information needs based on the user context and the responses of other users who have been in similar contexts. The newly acquired knowledge is immediately utilized to construct better or more detailed questions and thus drive the KA process further.

2. A shift from NL-pattern-driven conversational agents with a knowledge base to remember facts, to knowledge driven agents where NL patterns only
support the conversion from language into logical form, but do not directly drive the conversation.

3. A technical (system) implementation of the above listed two contributions as a **working real-world prototype** which shows the feasibility of the approach and a way to connect many independent and complex sub-systems. Sensor data, natural language, inference engine, huge pre-existing knowledge base, textual patterns and crowdsourcing mechanisms are connected and interlinked into a coherent interactive application.

These contributions make our approach a self-maintained and ever growing natural language conversational agent that can be used for non-KA-related tasks as well as KA, or in a kind of mixed scenario. The latter, mixed, scenario is our preferred target setting.

The key motivation behind using existing knowledge is to ensure that the system at least partially “understands” a user’s requests and its own responses, and to use that understanding, inter alia, to determine what it doesn’t know, and to support asking for the missing pieces of knowledge. The user provided answers are checked for consistency, validity and soundness against the existing knowledge by applying inference. Moreover, in the case of insufficient evidence for rejecting or supporting the user’s answer, the system can ask other users in a form of on-demand crowdsourcing. In this way, our system Curious Cat is a nearly self-sustained and self-maintained system, which performs its own KA in addition to serving the user-focused task/search it was designed for. Because its acquisition is guided and controlled by the combination of knowledge, inference and crowd responses, the knowledge that can be acquired in this way is expected to be of high quality and can therefore be automatically incorporated into the system’s knowledge base without a need for additional manual review, resulting in cost savings.
To test the proposed novel approach and assess its usefulness, we built a proof-of-concept conversational assistant named Curious Cat (Fig. 1) whose task is to provide its users with some interesting or useful information about the places they visit, while also being able to support incidental conversation ranging over common sense topics. This is achieved by extending Cyc (Lenat, 1995), a very large inferentially-productive Common Sense Knowledge Base (KB), resulting from more than 900 (as of 2006 (Matuszek, Cabral, Witbrock, & DeOliveira)) non-crowdsourced human years of effort.

Besides the knowledge driven KA approach, an equally important part of the system is a novel crowdsourcing approach preserving user privacy and the ability to store the user's beliefs about the world. This means that the knowledge the users provide through their answers can be local to the user and affect the other users only if promoted to general knowledge. If a user deliberately or accidentally misleads the KA process, this generally affects only the way the system interacts with that user, while having a minimal impact on the other users.

As a part of the validation experiment, Curious Cat has been publicly available online since the end of 2012 and is still running after 3.5 years of continuous activity. In this time, over 700 users around the world have registered in the system and provided over 57,978 answers, which combined with inference, resulted in more than 394,000 units/pieces of knowledge.

Although the implementation used in our experiments uses a particular knowledge base, reasoning engine and a set of NLP methods, the proposed KA approach it implements is general and can be repeated using any sufficiently broad and inferentially productive existing KB and a reasoning engine with the appropriate meta-reasoning (reasoning about its own structures) capabilities. In this direction, we
have developed and demonstrated an alternative implementation using our own simple KB and inference engine – Umko (Bradeško, et al., 2015) instead of using a well-established AI knowledge base Cyc (Lenat, 1995).

2. PROBLEM DEFINITION
This paper addresses construction of a knowledge acquisition system based on the use of existing knowledge, machine inference and mobile context to trigger natural language knowledge acquisition at user-appropriate moments. It addresses a broad range of as-yet incompletely solved problems from the fields of artificial intelligence, machine learning, natural language processing and human computer interaction. We propose a novel approach that addresses several of these problems and whose implementation additionally addressed the technical problem of scalability and workflow construction. For ease of understanding and later explanation of the approach and the working system, the remainder of this section describes the addressed problems grouped into six main topic groups. Detailed descriptions and evaluations are provided for three out of the six problems (described in subsections 2.2, 2.3 and 2.5) which constitute the scope and main contributions of this paper.

The proposed approach is knowledge driven, with knowledge connecting all of the components. By this we mean that interactions are triggered by knowledge and the results are stored in knowledge base (KB). User context is obtained through a real-world application that monitors the user’s activity and location through mobile GPS and accelerometer sensors. This raw context data must be corrected, clustered, classified and enriched (as described in sections 2.2.1 and 4.4.1) to obtain activities and locations. The results of this analysis are then asserted into the KB as contextual knowledge. Thus the KB needs to have a rich enough knowledge representation and vocabulary (section 2.1.1, 2.1.3 and 4.3) to be able to store the data. The newly asserted context can trigger forward chaining operation of the inference engine (section 2.1.2) which can result in the generation of a logical formula representing a comment or a question (section 2.3.1) that the system intends to show to the user. This logical formula is converted to natural language (sections 2.4.1 and 4.5.1) and communicated to the user. The user’s answer that is obtained in natural language, is converted back to logic (sections 2.4.2 and 4.5.2), checked against the existing KB for consistency, and inserted as new knowledge into the KB (sections 2.3.2 and 4.6). After this initial interaction, the system can determine whether to continue the conversational path with the user or not (section 2.4.3). New knowledge inserted in the KB can be used to drive the production of new questions/comments/suggestions, or to check with other users whether an answer is valid (sections 2.5 and 4.7). Other users then unknowingly vote for the validity of the existing knowledge by answering questions directed at them.

2.1 Knowledge Engineering, Representation and Reasoning
The proposed Knowledge Acquisition (KA) approach described in Section 4 is completely knowledge driven. One of the main building blocks of our system is its knowledge base, which must be based on a representation language expressive enough to support the knowledge structures required to drive the behavior of the system, and to represent the wide variety of knowledge it may gather from the users. Additionally, the inference engine needs to be able to access knowledge from the KB and needs to be powerful enough to perform reasoning over it.

2.1.1 Knowledge Base. The Knowledge base is a crucial part of our KA approach (marked in purple in Fig. 2). It dictates the expressivity of the knowledge
representation that we must use, as well as (together with the inference engine) the overall speed of the KA system. Although our approach is generally KB independent, each KB has its own specific characteristics, which need to be taken into consideration when representing and storing the acquired knowledge. As a part of this research we have considered three knowledge bases and the appropriateness of their knowledge representations. (1) The main system reported here was built based on the Cyc KB (Lenat, 1995), in a form similar to Research Cyc. (2) Inspired by the Open Cyc representation we also developed our own knowledge base and inference engine, Umko, designed to be as simple as possible while still supporting several of the typical use-cases presented here. Because of its simplicity, the approach built around Umko can run in totality on a 2015-quality smart phone. (3) Additionally, to test the generality of the approach, we used similar idea with a standard RDF knowledge representation (Bradeško, Moraru, Fortuna, Fortuna, & Mladenic, 2012), which can be used with any RDF compliant triple store.

2.1.2 Inference Engine. In addition to the KB, a crucial part of the proposed system is an inference engine (marked in red in Fig. 2) that can reason over the knowledge stored in the KB. The inference engine is used to detect the missing knowledge and infer what to ask and when, based on the context that is asserted as a part of the knowledge. Similarly to the knowledge base (section 2.1.1), the inference engine influences the speed and complexity of the approach. Our approach was again twofold. The main experiments were based on the Cyc inference engine which is tightly linked to the Cyc KB and thus able to apply inference over the biggest existing common sense KB. Additionally, we developed the inference capabilities (forward and backward chaining) inside our custom developed inference engine – Umko, needed to test the generalizability of the proposed approach and to make it possible to run on embedded devices.

2.1.3 Knowledge Representation (KR). The problem of representing knowledge in a computer is as old as the field of the AI research and has been tackled from various perspectives, but still not completely solved. The proposed approach relies on the KR which is powerful enough to describe the real world we live in. Additionally, it needs to cover knowledge about the knowledge itself (meta-knowledge), enabling inference over internal knowledge structures, questions and answers. This inference is used to produce statements and questions and interactions between them as part of a dialogue used to communicate with the user in the process of knowledge acquisition. We have tested two approaches, where one was fully based on the Cyc KB, and the other based on Umko, where we only created the minimal upper ontology and vocabulary to support the KA task.

2.2 Context (Information) Extraction
To be able to ask relevant questions, connected to the user or something that the user is currently doing or knows, and to ask at the right time/place, maintaining and using user context is crucial. Using context and the newly acquired knowledge to drive the KA process, is one of the main contributions of this paper. We propose a novel approach which can lead to additional knowledge not available otherwise.

Nowadays, the obvious information source to get to know the user is his or her mobile phone, which, with its sensors can provide an extensive and valuable source of information. However, the users need to approve using their phone’s sensors, and additionally, the raw measurements of sensor data need to be processed and
understood to the extent that it is possible to use that information in some sort of knowledge representation. In our approach, we used two types of the context. One was provided by the users directly via answering questions about themselves (section 2.2.2). Another part of the context was mined from the phone sensors as described in sections 2.2.1 and 4.4.1.

2.2.1 Mobile Sensor Stream Analytics (Mined Context). Mining raw sensor measurements and extracting the meaning out of the data is a sub-problem on its own. The proposed KA approach relies on mining mobile phone GPS coordinates and accelerometer sensors to extract the places where the user stays for a while, time of the visit, duration of stay, and the path taken to come there. Additionally, we are enriching these stay points with the name and the type of location. This was done by applying a two pass stay point detect (SPD) algorithm (Quannan, et al., 2008) and the Foursquare API. This is described in more detail in section 4.4.1. Additionally, we use accelerometer sensors, to detect user activity (walking, running, cycling, driving, still), which is provided by Google as a part of the standard Android API. The resulting information extracted from the sensors is asserted into a protected portion of the knowledge base as part of its contextual knowledge. This is depicted as an orange arrow in Fig. 2.

2.2.2 Knowledge Acquired from the User In addition to the context that can be mined automatically from the mobile sensors, the KA process benefits substantially if it has additional information about the user. For instance, what language she/he speaks, what are his/her interests, profession, food he/she likes, etc. This knowledge can be used by the KA rules to fine tune the questions and suggestions in order to generate more questions that the user can answer. Using context knowledge can make the questions more interesting for the user (e.g., asking the user if an already stored wireless password is still valid for a particular location that the user is currently at). In addition to specific hand crafted questions targeted at users’ personal information, context acquisition can use gained knowledge to ask even more questions. Wherever there is some part of knowledge in the KB that can be connected to the person, the system constructs a personal question for each user.

2.3 Knowledge Acquisition

The main scientific contribution of this paper is a new approach to tackling the knowledge acquisition problem. The proposed approach combines natural language crowdsourcing, usage of prior knowledge, context and newly acquired knowledge. For this, we had to address many sub-problems of knowledge driven KA as described in this section. The KA logic is all encoded as meta-knowledge in the KB and driven by inference. This is represented in red (B) and purple (A) in the architectural diagram Fig. 2.

2.3.1 Question and Statement Formulation Logic. To be able to generate questions and other statements from a KB using an inference engine, there must first be a logic vocabulary supporting that construction. This means that the KB needs to have a meta-structure (knowledge representation) which allows it to describe its own contents and be able to address assertions and other logical formulas and variables. After this is in place, the approach needs logical rules that are used by the inference engine to produce new logical queries or statements. As part of this research, we extended the existing Cyc vocabulary with supporting meta-structures and knowledge acquisition
rules. Additionally, we tested the same approach on the Umko KB and inference engine.

2.3.2 Answer Consistency and KB Placement. The logical queries and statements generated by the inference engine are converted to natural language (section 2.4.1 and 4.5.1) and presented to the user. The user's answer is converted to logic (section 2.4.2 and 4.5.2). Then the system checks whether the answer is coherent and consistent with the existing KB. This is crucial to avoid corrupting the KB over time as more new knowledge is added through the KA process. This check is performed with the help of the inference engine, which is triggered at the time of assertion of new knowledge and detects inconsistencies when or before an attempt to assert the answer into the KB is made. This process needs to be supported by a suitable knowledge representation (as mentioned in section 2.1.3) providing a suitable logical vocabulary. Our solution addressing this problem is described in detail in section 4.3.

2.3.3 Maximizing Knowledge Gain. As we are generating questions and incorporating the answers into the KB, it is obvious that some questions are more worthy of being asked than others. This details depend on the specific task of the knowledge acquisition, but regardless the task, some questions and answers are more valuable than the other. While this problem was not the main focus of this work, we addressed it partially by introducing vocabulary that allows us to control the order and priority of the questions.

2.3.4 Maximizing the chance of getting an answer – Timing, Location and other Context. In the KA process, we want to maximize knowledge gain. A part of that is maximizing the chance of getting any answer from the user. The KA process thus become a multi-objective optimization, where we need to optimize multiple parameters to obtain as many answers as possible, while maintaining quality of the answers and keeping the user engaged at the right level (not bored and not overwhelmed). This is where the context helps the most as we can ask questions which are related to the user's current situation. In particular, the user location, user activity and time of the day helps the inference engine to decide when and what to ask. This allows the system to be aware that the user is in a restaurant and not for example in a church, which would produce totally different questions. Additionally, it enables asking "how was the food?", for example, at the right moment, when the user has already eaten in a restaurant, and not when he or she has just arrived there.

2.4 Natural Language Processing
To be able to use crowdsourcing and address the users in natural language, the KA system needs to be able to convert the logical representation of questions to their natural language representations. Similarly, when getting the user's answers, these answers need to be converted back to logic. In addition, to be able to drive a coherent conversation with the user, there needs to be some logic which drives the dialog and the order of the questions and responses (mentioned also in sections 2.3.3 and 2.3.4.).

2.4.1 Logic to Natural Language Conversion. Being able to convert logic that the system uses to natural language that the user can understand is crucial part of the presented KA approach. While this is in general a hard problem and still not completely solved, it is often easy to solve simple examples of natural language generation. In our
approach, we have investigated two scenarios. One is using pre-existing Cyc natural language capabilities; the other is a simple scenario that is a scaled down problem using Umko. Cyc has more than 92 concepts which can be applied to create natural language generation rules (Baxter, 2005), and most of the predicates and concepts in the KB are annotated with the rules that instruct the system on how to convert them to natural language. For the missing NL rules and also for the new vocabulary, we extended the Cyc KB.

2.4.2 Natural Language to Logic Conversion. Due to language ambiguity, NL to logic conversion is a harder problem than the logic to NL (Schneider, 2015) (sections 2.4.2 and 4.5.2). Similarly, the problem is not completely solved yet, but to some extent, it is possible to do the simple conversion using textual patterns such as the ChatScript (Wilcox, 2011) or AIML (Wallace R., 2003) approach. For our approach, we tested two approaches. One based on Cyc, which additionally to textual patterns uses inference to check whether the conversions make sense and is to some extent being able to disambiguate ambiguous conversions (Schneider, 2015). The second approach applied simple manually written patterns and Umko (Bradeško, et al., 2015).

2.4.3 Dialog Formulation Logic. Maintaining a coherent dialog while asking and answering user questions is almost as important as the language generation itself. Without it, the user quickly loses focus, is unsure about what the question refers to, etc. While this is mostly the domain of purely conversational agents such as chatbots (Wilcox, 2011) (Wallace R., 2013) (Bradeško & Mladenić, A Survey of Chatbot Systems through a Loebner Prize Competition, 2012), our approach still needs to address this to some extent. In contrast to other approaches which use text patterns to encode the dialog request and responses, our dialogs are triggered by knowledge. This means that the system actually “understands” the request and response and thus provides logical sequences in the conversation. The drawback is that there can be examples which are not covered and where the system will not be able to respond or not being able to parse them. While this is not optimal, it is not very different from the current state of the art solutions.

2.5 Crowdsourcing

While the proposed KA approach works with even a single user, it shows its full potential when used in the crowd-sourced setting. Crowdsourcing allows the system to get better knowledge coverage including specifics that only a subset users can provide. Additionally, it allows to double-check the answers entered by the user against the answers of other users. In order for the system to use crowdsourcing, it needs to be able to address the problems that come with that, which are mostly related to different opinions, deliberately entering wrong information, trolling and handling the temporal and time dependent knowledge.

2.5.1 Knowledge Quality and Truth Control. With crowdsourcing systems, the most important and most obvious problem is quality control. Normally there needs to be a person or evaluation system that checks the quality of data returned by the crowdsourcing mechanism. For our KA purposes, we address quality control on two levels. First, the inference engine uses the existing knowledge to check whether the answers are consistent. Then, after the answers come through this first filter, we have
two mechanisms that can forward the newly acquired knowledge to other users able to assess it. This is described in more detail in section 4.7.

2.5.2 Handling Different Opinions. In addition to mistakes and wrong answers, users can simply have different opinions about something. Since our KA system also behaves as an assistant and needs to be consistent towards the user, it also must handle different opinions. For this reason, we allow the users to have their own beliefs about the world, which do not affect the common knowledge base, if inconsistent with it. In this way if the users lie or have different opinion about something, it will only affect themselves and not the rest of the users. This capability is achieved by arranging knowledge in a tree-like structure of contexts, where the upper context is not aware of the lower ones, and the knowledge in the lower tree levels overrides the knowledge in the upper levels. In Cyc this is a kind of mechanism implemented through Micro Theories (aka microtheories, MTs) (Patent No. US 8401988 B2, 2010) In Umko, we loaded user-specific knowledge over and overriding the general/shared knowledge, each time the KB was used. This was possible due to a small KB, limited scope of the KA and the light implementation of Umko KB and Inference engine.

2.5.3 Handling Temporal Knowledge and Changing Knowledge. On top of different opinions, and deliberate and accidental mistakes in the KA process, an additional problem is that some knowledge that reflects the real world must change through time. This problem needs to be tackled from two perspectives. One is that the KB and inference engine need to support the temporal dimension of knowledge. The second is that the systems need somehow to capture that the change happened and data previously valid may become invalid. For the approach using Cyc we use the fact that Cyc supports time dependent knowledge through special microtheories where the ‘always’ microtheory is the top-most one and the time dependent ones are connected to the top one through a hierarchical microtheory structure (see section 4.7). For our simplified implementation using Umko, the time dependent KB is stored externally and only loaded into Umko on the fly, based on the time interval.

2.6 Workflow and scalability
Besides the scientific challenges, we have described related to the proposed KA approach, in order to be able to work on a broader level, technical issues and scalability had to be addressed as well. The knowledge base and inference engine if used for common sense knowledge, tend to be quite large and resource hungry. Our experiments were concluded using three Cyc instances, running on two machines and a transcript server which is used for KB syncing on a remote EC2 Amazon instance. One main machine was running two instances each having 16GB of memory. The additional machine was used mostly for testing and evaluation. All the three Cyc instances were synced over the EC2 transcript server.

3. RELATED WORK
Knowledge Acquisition has been addressed from different perspectives by many researchers in Artificial Intelligence over decades. Some of the more successful approaches are described in the Common-Sense Knowledge Acquisition survey (Zang, Cao, Cao, Wu, & Cao, 2013), which organizes the approaches in four main groups:

1. Labour Acquisition. This approach uses human minds as the knowledge source. This usually involves expert ontologists manually coding the knowledge.
2. **Interaction Acquisition.** As in Labour Acquisition, the source of the knowledge is human minds, but in this case the KA is wrapped in a facilitated interaction with the system, and is sometimes implicit rather than explicit.

3. **Reasoning Acquisition.** In this approach, new knowledge is automatically inferred from the existing knowledge using logical rules and machine inference.

4. **Mining Acquisition.** In this approach, the knowledge is extracted from some large textual corpus or corpora.

Our approach is a hybrid system, addressing the problem by combining labour and interaction acquisition and, adding unique features of using user context and existing knowledge in combination with reasoning to produce a practically unlimited number of potential interaction acquisition tasks. The existing related work can be divided into systems that exploit existing knowledge (generated anew during acquisition or pre-existing in other sources) (Mitchell, et al., 2015; Witbrock, et al., 2003; Forbus, et al., 2007; Sharma & Forbus, 2010; Singh, et al., 2002; Kuo & Hsu, 2010), crowdsourcing (Singh, et al., 2002; D.S. Pedro & Esteveam R., 2012; Pedro, Appel, & Hruschka, 2013; Speer, et al., 2009; Kuo & Hsu, 2010), acquisition through interaction (Hruschka & Pedro, 2012; Pedro, Appel, & Hruschka, 2013; Speer, et al., 2009), reasoning (Spee R., Open mind commons: An inquisitive approach to learning common sense, 2007; Witbrock, et al., 2003; Speer, Lieberman, & Havasi, AnalogySpace : Reducing the Dimensionality of Common Sense Knowledge, 2008; Kuo & Hsu, 2010), and natural language conversation (Hruschka & Pedro, 2012; Speer R., Open mind commons: An inquisitive approach to learning common sense, 2007; Speer, et al., 2009; Witbrock, et al., 2003; Kuo & Hsu, 2010).

One of the most similar systems (by functionality and approach) is Goal Oriented Knowledge Collection (GOKC) (Kuo & Hsu, 2010). Like Curious Cat, GOKC starts with an initial seed KB which is used to infer new questions that are presented to the user. The GOKC authors first demonstrated that without generating new types of questions, the knowledge in a domain gets saturated. Then they introduced a rule which can infer new questions through concepts which are connected by the predicates. For example, if we know `atLocation` is linked with `hasSubevent` and we get an answer “class-room” to the question “You are likely to find in a school”, we can ask players a new question “One thing you will do when you in classroom is __,” (Zang, Cao, Cao, Wu, & Cao, 2013). Unlike Curious Cat, which checks answers for validity and has a variety of question generation rules, GOKC has only one rule, is fixed to a specific domain and accepts all user answers, which are then filtered by voting.

Another related system, which is a predecessor of Curious Cat in some aspects, was the Cyc Kraken Systems’ User Interaction Agenda (UIA) (Witbrock, et al., 2003). Like Curious Cat, the UIA was able to check answers for consistency, but it didn’t use rules to explicitly drive question-asking. This latter feature was introduced in a new approach within Cyc: CURE (Content Understanding, Review and Entry, which is available within Research Cyc), and which Curious Cat extends. In the UIA, the system relies on domain experts to select a concept, which then triggers a sequence of NL forms, which allow users to add new knowledge that may later be modified using “knowledge engineering tools”. Both CURE and UIA are missing crowd-sourcing functionalities and pro-activity based on user context.

After an initial KB, extracted from internet textual content had been gathered, the CMU text-mining knowledge acquisition system NELL (Mitchell, et al., 2015) (Never Ending Language Learner), started to apply a crowdsourcing approach (Pedro, Appel,
& Hruschka, 2013), using natural language questions (Hruschka & Pedro, 2012) to validate its KB. In the same fashion, as Curious Cat, NELL can use newly acquired knowledge, to formulate new representations and learning tasks. There are, however, distinct differences between the approaches of NELL and Curious Cat. NELL uses information extraction to populate its KB from the web, then sends the acquired knowledge to Yahoo Answers, or some other Q/A site, where the knowledge can be confirmed or rejected. By contrast, Curious Cat formulates its questions directly to users (and these questions can have many forms, not just facts to validate), and only then sends the new knowledge to other users for validation. Additionally, Curious Cat is able to use context to target specific users who have a very high chance of being able to answer a question.

On the other end of the spectrum lie conversational agents (chatbots) with a vast number of hand-scripted NL patterns and responses. These patterns help the bot to appear intelligent, but in a limited way: the knowledge is only implicitly encoded in the patterns and cannot be used anywhere else. Recently, some successful chatbots have started to employ an internal knowledge base as well. It is used to remember facts from the conversation for later use. In this way, the chat system is actually doing targeted knowledge acquisition. Two of the more successful approaches to Chatbot authoring are AIML (Wallace R., 2003) and ChatScript (Wilcox, 2011); there has also been an attempt to extend AIML scripts with CYC knowledge (Coursey, 2004), by using NL patterns to match to particular logical queries, enabling a bot to answer questions from Cyc KB. Unlike responses generated due to NL pattern matches, Curious Cat generates NL responses based on knowledge and context driven inference. This allows our system to proactively engage users in sensible conversations and is thus one of the first approaches that attempts to bridge the gap between maintaining the NL conversation of chatbots and actually understanding its content.

4. APPROACH

We propose and implement novel approach to automated knowledge acquisition using the user context obtained from a mobile device and knowledge based conversational crowdsourcing. The resulting system named Curious Cat has a multi objective goal, where KA is the primary goal, while having an intelligent assistant and a conversational agent as secondary goals. The aim is to perform KA effortlessly and accurately while having a conversation about concepts which have some connection to the user, allowing the system (or the user) to follow the links in the conversation to other connected topics. We also allow to lead the conversation off topic and to other domains for a while and possibly gather additional, unexpected knowledge. For illustration see the example conversation sketch in Table I, where topic changes from a specific restaurant to a type of dish. In this example case, the conversation is started by the system when user stays at the same location for 5 minutes.

<table>
<thead>
<tr>
<th>Step num.</th>
<th>Interaction</th>
<th>User1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Curious Cat Where are we? Are we at Joe’s Pizza restaurant?</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Curious Cat I’ve never been here before. What kind of place this is?</td>
<td>restaurant</td>
</tr>
<tr>
<td>3</td>
<td>Curious Cat Does Joe’s Pizza have Wi-Fi</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>Curious Cat Is it fast enough to make Skype calls?</td>
<td>I don’t know</td>
</tr>
</tbody>
</table>
While the system is conversing with the user, it uses all the knowledge gathered in the conversation and is able to use it to further generate new comments and related questions. This is also evident in the interaction number 4 in Table I. At the same time, or later, when some other user is in a similar context, the knowledge can be double checked with another user, as shown in Table II. Based on the votes (confirmations or rejections) from the crowd (other users), the system can decide whether to believe the new knowledge in general, or only leave it in the user's own world theory. This is explained in more detail in section 4.7.

As we hinted in the above examples, Curious Cat can be described as a knowledge driven conversational Knowledge Acquisition agent. It uses existing knowledge and additional sensory context to be able to drive the conversation and ask improve in asking questions. This is also its main differentiation from the other KA systems and conversational bots. It is not driven by some predefined scripts and textual patterns, but uses structured knowledge which can evolve and change through the interaction and also independently through external context and sensors. Moreover, the KA process and the system are controllable¹ (and to some extent self-regulated) and actionable², which is not the case with statistical conversational client approaches such as Cleverbot (Saenz, 2010).

---

¹ It is based on structured knowledge and it is thus possible for humans to review/understand and influence its behavioral rules.
² Because of the use of structured knowledge, it’s possible to create special predicates representing actions in the external world, such us turning on the lights, or reserving a restaurant; not actionable in the legal sense.
4.1 Architecture

The presented approach and its corresponding implementation results in a complex system, with many interconnected modules solving challenging AI, machine learning and natural language processing problems. To make the system as understandable and maintainable as possible, we designed a modular architecture (Fig. 2), where the individual modules correspond to different problems described in section 2. In Fig. 2, the modules are shown in different colors (representing functionality groups), each having one or more sub-components.

We can see that the system and its user interaction loop are built around the knowledge base in the center (marked in purple and letter A in Fig. 2). Internally KB has three components. The main part, which should in real world also be the biggest, is the common-sense knowledge with its upper ontology over which we operate. This part of the KB contributes the most to the ability to check the answers for consistency. The more knowledge already exists, the easier becomes to assess the answers. The second part is the user Context KB, which stores the contextual knowledge about the user. This covers the knowledge that the user has provided about himself (section 4.4.2) and the knowledge obtained by mining raw mobile sensors (section 4.4.1). This is represented as the orange arrow, pointing into the context part of the KB. The sensor based context allows the system to proactively target the right users at the right time and thus improve the efficiency and accuracy and also stickiness of the KA process.

The third KB part, is the meta-knowledge and KA rules that drive the dialog and knowledge acquisition process (section 4.3.3). Although in our implementation we used Cyc KB and tested Umko KB, the approach is not fixed to any particular knowledge base. But it needs to be expressive enough to be able to cover the intended knowledge acquisition tasks and meta-knowledge needed for the system’s internal workings.

After the KB, the second most important part of the architecture is an inference engine (in Fig. 2 marked in red and letter B), which is tightly connected to the knowledge base. The inference engine needs to be able to operate with the concepts, assertions and rules from the KB and should also be capable of meta-reasoning about the knowledge base’s internal knowledge structures. As the individual components (indicated with red color in Fig. 2) suggest, the inference engine is used for:

- Checking the consistency of the users’ answers (e.g., can you order a car in a restaurant if it’s not food?).
- Placement of new knowledge inside the KB.
- Querying the KB to answer possible questions.
- Using knowledge and meta-rules to produce responses based on the user and her/his context input (similar in function to the scripts in script-based conversational agents).
At both ends of the stacked chain in Fig. 2, there are natural language processing components (marked in blue and with letter E), which are responsible for logic-to-language and language-to-logic conversion (sections 2.4 and 4.5). These are crucial if we want to interact with users in a natural way and thus avoid the need for users to be experts in first order logic. This module and its components are described in more detail in section 4.5.

Besides the main interaction loop, which implicitly uses crowdsourcing while it interacts with the users, there is an additional component (marked in green and with letter F). This “crowdsourcing and voting” component handles and decides, which elements of knowledge (logical assertions) can be safely asserted and made “visible” to all the users and which are questionable and should stay visible only to the authors of the knowledge. If the piece of knowledge is questionable, the system marks it as such and then the question formulation process will check with other users whether it’s true or not. This is described in more detail in section 4.7.

In addition to logic-based components presented above, there is a functional driver system (marked in orange), which glues everything together, forwards the results of inference to the NL converters, accepts and asserts the context into the KB, handles the synchronization between the instances of the systems, etc.
4.2 Interaction Loop

Fig. 2 also indicates a simple system/user interaction loop represented by arrows. The system can trigger a conversation based on the new context, such as, change of the location, or some other knowledge that appears in the KB (from other users or the same user). This is also how the system is initialized and how the main pro-activity driver is implemented. The functional support component (orange) constantly monitors user context and asserts it into the KB. Additionally, it is streaming user GPS coordinates through the clustering pipeline (section 4.4.1), and asserts the results. These contextual assertions then usually trigger a change in the set of questions for the user, which is a sign for the system to show the newest question to the user, as happened with step 1 from Table I.

Additionally, the user can trigger a conversation at any point in time either by continuing the previous conversation or simply starting a new one. From this point of view, we have two options for the interaction. Human to machine (HMI), and machine to human (MHI). The specifics of both, which cannot fit in Fig. 2 are explained in the following sub-sections 4.2.1 and 4.2.2. Besides the standard HMI and MHI, the proposed system also allows a novel Machine Mediated Human to Human interaction (MMHHI) which benefits both the users and the KA system.

4.2.1 Machine to Human Interaction (MHI). To communicate to the user, Curious Cat needs to convert the question or statement from its logical form to natural language and present it to the user on the screen, or read it aloud through a text to speech interface. While there are different ways to do the conversion, the original logical representation is not ambiguous which makes the task easier. For Curious Cat this is done by using special language assertions which describe the language generation for concepts from the KB in case of using Cyc. A simpler alternative would be to use specific textual patterns that are mapped to logical concepts, which was done for our Umko implementation. This interaction is triggered by the appearance of new knowledge in the KB which triggers the inference engine to produce assertions like ccWantsToAsk(CCUser1,"userLocation(CCUser1,CCLoc1)") – see assertion 14 in section 4.3.3. These assertions are then converted from logic to NL (section 4.5.1) and then presented to the user as visible from the second column in example conversation from Table I and Table II.

4.2.2 Human to Machine Interaction (HMI). Complexity of the HMI (natural language to logic) is a lot higher than in the opposite direction. For this reason, Curious Cat tries to make the conversion easier by guiding the human through the writing process. The guidance can consist of the variations of these:

- A fixed set of options that the user can pick from, which are generated by the inference engine using the KB at the question generation time. This is possible when the existing KB is big enough to provide enough evidence on the possible answers to particular question.
- A set of options that can be ignored by the user by entering a custom text, which is then treated as the knowledge guided input, or free text input.
- Knowledge guided input, which allow the user to enter free text with autocomplete, but it only accepts the text if it matches the structure or content of the original question and the system’s idea of what is allowed to be the answer.
- Completely free text where the user can write anything. This is the hardest to parse. Curious Cat is parsing these responses by using SCG (Schneider, 2015)
patterns which are linked to logical structures. This would be used for example, when user, instead of simple ‘Pizza Deluxe’ (step 5 in Table I), would say ‘They sell pizzas’ or something even more complex (see section 4.5.2).

The example of mediated answer guidance can be seen in Fig. 3, where the system presented a set of possible answers while still allowing a free text which will be autocompleted with the food types that the system knows about. If the user enters something new, the system will accept that (as was shown in the step 6 in Table I).

![Example of mediated answer guidance](image)

Fig. 3. Example of human to computer interaction options

4.2.3 Machine Mediated Human to Human Interaction (MHHI). Both of the interactions described in sections 4.2.1 and 4.2.2 are explained in Table III. The innermost arrows represent the MHI interaction, where machine (blue arrow) asks the user a question, which the user answers (red arrow). Similarly, the HMI is represented by the outer arrow circle, where a human starts interaction by asking a question (1b red) and the machine answers (2b blue).
Table III. Possible interaction types between the user and Curious Cat KA System

<table>
<thead>
<tr>
<th>Questions</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMI, MHI, MMHHI</td>
<td>HMI, MHI, MMHHI</td>
</tr>
<tr>
<td>Human</td>
<td>Machine</td>
</tr>
</tbody>
</table>

Legend
- Red color represents human questions/answers by:
  - Predefined options (menus)
  - Knowledge guided input
  - Free text input
- Blue color represents machine questions/answers by:
  - NL Generation rules
  - NL Patterns

Source: Luka Bradeško, Marko Grobelnik - IJS, Curious Cat Inc.

Now consider the example, when the user asks the system a question (1b again), but Curious Cat AI doesn’t have the answer (The NL question gets converted into the logical query which doesn’t retrieve any results from the current KB – see section 4.5.2 and 4.6 for conversion and query examples). At this moment, we can start the MMHHI process, so the system forwards the question to other users which are in one way or another connected to the question (3b - the bottom blue arrow). As soon as one of the users provides an answer, the answer is forwarded back to the original user who asked the question (4b - bottom red arrow). Because Curious Cat is primarily the KA system, and because it already parsed the question into its logical representation, it will know which users to ask. Additionally, it will learn a new type of question that is worth asking about something and also learn the answer on that question simultaneously. Additionally, to that, even if the system cannot parse (understand) the question and is unable to convert it to logic, it can still forward it to other users as it is, and then forward the answer. At the same time, it can remember the textual representation of the question/answer pair and process it by some other algorithm or store it for later usage as a data-set for machine learning approaches.

4.3 Knowledge and KA Meta-knowledge

The complete Curious Cat system including the KB is too big and complex to be fully explained here. The KA Meta Knowledge, which drives the KA process and NL dialog alone consists of 12,353 assertions and rules. Also, the detailed formal logical definitions of the system were already presented in the earlier Curious Cat paper (Bradeško, Witbrock, Starc, Mladenic, & Grobelnik, 2016). While the previous paper focuses on the formal definition of the KA logic that is used for our KA system and has very short validation, here we focus on the fundamentals of the approach and define the simplest possible logic to explain the system functionality through examples given in Table I and Table II. Comparing to the previous work, in this paper we have 6 months of additional experiment data and better logs which resulted in more thorough evaluation and
insight in the results, which were not provided in the previous work. Additionally, this paper explains in more detail the mechanisms behind the context retrieval.

The examples are given in higher order predicate logic, where some of the predicates represent more complicated structures as presented in earlier paper (Bradeško, Witbrock, Starc, Mladenic, & Grobelnik, 2016). For better readability, instead of $x,y,z$ we mark variables with a name that represents the expected type of the concepts the variable represents followed by a question mark (?). For example, when we see a variable in a logical formula, such as $\text{CCUser}(\text{PERSON})$, we immediately know that $\text{PERSON}$ can be replaced with an instance or a subclass of the concept $\text{Person}$.

Predicate names are lower case (predicate) and the rest of the concepts start with a capital letter (Concept). At this point it is worth noting that while our logical definitions and formalization are strongly influenced by Cyc (Lenat, 1995), and while we use the Cyc upper ontology, the approach is general and not bound to any particular implementation. Our notation reflects this but is not tightly bound to that of OpenCyc.

4.3.1 Upper Ontology. First, we introduce the concepts that will allow us to present the upper ontology which is a part of the knowledge base:

\begin{equation}
\text{Something, Class, Predicate, subclass, is, arity, argIs, argClass}
\end{equation}

Then we define the relevant concepts. In predicate logic, we can state things like:

\begin{equation}
\text{Person}(x)
\end{equation}

Which means that $x$ is a Person, or more precisely, an instance of a class. Person. We introduce a special predicate ‘is’ stating that something is an instance of some class, with the class being given as an argument:

\begin{equation}
\forall x \forall P (P(x) \Leftrightarrow \text{is}(x, P))
\end{equation}

Now, instead of $\text{Person}(x)$ we can use the $\text{is}(x, \text{Person})$ notation, which will allow us to construct logical statements ranging over classes in a more transparent way. Detailed derivation of the logic is provided in (Bradeško, Witbrock, Starc, Mladenic, & Grobelnik, 2016):

\text{subclass}(\text{CHILD\_CLASS?, PARENT\_CLASS?}) - A subclass relation between two classes. Its arity is 2 (2 arguments) and both of the arguments must be instances of $\text{Class}$ concept.

\text{arity}(\text{PREDICATE?, NUMBER?}) - An arity relation, used to assign arity (number of arguments) to predicates. First argument must be instance of $\text{Predicate}$ and the second must be a number.

\text{argIs}(\text{PREDICATE?, NUMBER?, CLASS?}) - An argument instance relation, used to limit the arguments of the predicates to be of instance of a specific class. First argument must be an instance of $\text{Predicate}$, second a number and third must be an instance of $\text{Class}$.

\footnote{The type here is merely notational; the system does not use type information from variable names. Types are enforced by the predicates, however.}

\footnote{The notation here follows closely practices used in Cyc. For more details, readers can refer to (Lenat 1995) and (Matuszek, et al. n.d.) and the references it contains.}
argClass (PREDICATE?, NUMBER?, PARENT_CLASS?) - An argument subclass relation, used to limit the arguments of the predicates to be subclasses of a specific class. First argument must be an instance of Predicate, second a number and third must be an instance of Class.

\[
\forall x\forall y\forall z (\text{is}(x, y) \land \text{subclass}(y, z) \Rightarrow \text{is}(x, z)) \quad (5)
\]

\[
\forall x(\text{is}(x, \text{Class}) \Rightarrow \text{subclass}(x, x), \forall x\forall y\forall z (\text{subclass}(x, y) \land \text{subclass}(y, z) \Rightarrow \text{subclass}(x, z)) \quad (6)
\]

Where rule (5) tells the inference engine that whenever something is instance of a class and this class is a subclass of its parent class, then it is an instance of the parent class as well. The two rules under (6) are defining the transitivity of subclass predicate. An instance of a class is at the same time a subclass of itself. Additionally, when we have a subclass of a class that is a subclass of another class, the first class is a sub-class of the other class as well.

4.3.2 Existing Knowledge. In our Curious Cat implementation, we use an extended full Cyc ontology and KB, similar to that released as ResearchCyc, as a common-sense knowledge base. This is far too big (millions of assertions), to be explained in any detail here. Thus, we define only the concepts and predicates that are necessary for explaining the proposed approach. These concepts are as follows:

\[
\text{User, Place, PublicPlace, Restaurant, FoodOrDrink, Food, Drink, WirelessService, Service, Vehicle, Car, Visit, Coffee, Restaurant1, User1, User2} \quad (7)
\]

And predicates:

\[
\text{probableUserLocation(USER?, PLACE?)} - \text{A predicate used to represent the probable current user location into the KB. It is probable because it is automatically inferred by the ML algorithm (section 4.4.1) and we still want the user to confirm it.}
\]

\[
\text{userLocation(USER?, PLACE?)} - \text{A predicate representing the confirmed current user location.}
\]

\[
\text{menuItem(RESTAURANT?, FOODORDRINK?)} - \text{A predicate used to enter the knowledge about menu items in restaurants.}
\]

\[
\text{providesServiceType(PLACE?, SERVICE?)} - \text{A predicate used to enter the knowledge about places providing services.}
\]

And the predicate constraints, which will allow us to serve as an example to explain the consistency checks and NL to logic conversion. Basically, with these assertions we tell the inference engine that menuItem can be used to only write assertions that connects instances of a class Restaurant and sub-classes of the class FoodOrDrink:
Notice that places and foods are connected in a hierarchical subclass structure, represented in Fig. 4, where blue squares are representing classes and green instances of classes. The lines represent relations.

Pre-existing knowledge structure can be used by the KA rules and meta-knowledge (sub-section 4.3.3).

4.3.3 KA Knowledge. In the previous two sections, we defined the upper ontology and then using its vocabulary the pre-existing knowledge. This will suffice to support the explanation of the proposed KA approach. As listing full set of Curious Cat KA rules would not fit in the paper, we define an example set of the KB sufficient for describing the approach and keeping the explanation as simple as possible. For this, we need to define a main KA meta-class Formula, which’s instances are all of the logical assertions, statements and queries we operate with. For simplicity, we will be referring to the using string (" operator. For example:

\[ \text{is}("(\text{is(Restaurant1, Restaurant}), \text{Formula}) \]  

And meta-predicates:

\text{unknown} (\text{FORMULA}? ) - A predicate which is true only when the formula it is referring to cannot be proven by the inference engine – this means it is not in the KB, or the knowledge that would make it true is not in the KB.

\text{known} (\text{FORMULA}? ) - A predicate which is true only when the formula it is referring to is asserted in the KB, or can be proven by the inference using other knowledge. It’s the opposite of the \text{unknown} predicate above.

\text{list} (\text{SOMETHING?}, \text{SOMETHING1}? ) – A predicate representing a logical function that can represent lists of concepts through list of lists structures. For example:

\[ \text{list}\left(\text{concept1}, \text{list(\text{concept2}, \text{concept3})}\right)\]  

\text{CCWantsToAsk} (\text{USER?}, \text{FORMULA}? ) – A predicate that is used by the KA rules to show an
intent of the system to ask a question defined in FORMULA?

\[ CC\text{WantsToAskWithSugg}(\text{USER?}, \text{FORMULA?}, \text{LIST?}) \] – The same as the \text{CCWantsToAsk} predicate, except that this one allows a list of pre-defined answers that user can pick (see section 4.2.2).

\[ CC\text{WantsToComment}(\text{USER?}, \text{FORMULA?}) \] - Similar as the above predicates, but used when the system wants to explicitly show that the formula is not the question, but the comment. This makes it easier for the NL engine to figure out the sentence type (declarative, interrogative …).

(11)

Now, after we have supporting predicates, we can define an example KA rule. KA rules can be written specifically to enable the production of questions for a narrow context. An example of a general KA rule is the following:

\[ \forall c \forall P \forall i_1 \exists s \forall i_2 \forall u \left(\text{is}(i_1, c) \land P(i_1, s) \land \text{is}(i_2, c) \land \text{unknown}("\exists s_2 P(i_2, s_2)")\right) \Rightarrow CC\text{WantsToAsk}(u,"P(i_2, $x$)) \] (12)

This rather complicated material implication causes generation of question intents whenever there is an instance of a class that was used in an arity 2 predicate, and there is another instance of the same class which doesn't have any assertion using this predicate. If we take our example KB (Fig. 4) and imagine we add \text{Restaurant2} (Joe's Pizza, as we did in the example conversation Table I, step 1), the premises of the rule can be satisfied like this:

\[ \text{is}(\text{Restaurant1}, \text{Restaurant}) \land \text{menuItem}(\text{Restaurant1}, \text{Coffee}) \land \text{is}(\text{JoesPizza}, \text{Restaurant}) \land \text{unknown}("\exists s_2 \text{menuItem}(\text{JoesPizza}, s_2)") \] (13)

Which produces the consequent \text{ccWantsToAsk}(\text{User1},"\text{menuItem}(\text{JoesPizza},$x$)"), which can be sent to the NL converter to ask the question, “What is on the menu in Joe's Pizza”, as we see in Table I, 5. The rule (12) effectively detects when there is an instance in the KB that doesn't have some kind of information that other instances have, and then it causes the system to intend to ask about it, if and when it has a suitable opportunity (e.g. a suitable interaction context).

The rule described above is an example of the general rule that can produce the apparent curiosity of the system using nothing but the existing background or newly acquired knowledge, whatever that may be.

In very large knowledge bases, general rules like this can produce many questions, including, in some cases, many irrelevant ones. To mitigate this in our implementation we have additional rules that can suppress questions on some predicates or for whole parts of the KB. While we defined rule (12) in some detail here to show the possibilities of the approach, we will explain the rest of the system through simpler examples for easier understanding. For example, the narrower rule which produces the step 1 of example shown in Table I is defined as follows:

\[ \forall u \forall p (\text{probableUserLocation}(u, p) \Rightarrow CC\text{WantsToAsk}(u,"\text{userLocation}($u, $p$))) \] (14)

We can see now that the KA or “curiosity” rules can span from very general, to very specific. General rules can automatically trigger on almost any newly asserted knowledge, while specific ones can be added to fine-control the responses and
knowledge we want to acquire. How specific or general the rule will be, is simply controlled by the number and content of the rule premises. Adding these KA rules (such as 12 and 14) is at the moment manual work (there is a possibility to generate them from user answers and suggested questions – future work), but one such rule can generate a lot of questions/comments, depending on the rule generality. The KA rules and also other vocabulary can be entered into the system in a similar fashion as AIML or ChatScript patterns in the existing chat-bot systems. The example of knowledge entry is provided on Fig. 5, where we enter 6 assertions and 1 KA rule, which generates a question about the location type for all the users and their last visited locations (This KA entry is Cyc specific – using CycL, for Umko or other implementation the language syntax is replaced with the system logic representation).

Fig. 5: Example of knowledge entry (using Cyc). This example is defining a concept PersonTypeByEatingHabits and a rule generating questions about typeOfPlace for last known user venue visit.

4.4 Context

As mentioned in section 2.2, in order to be able to ask relevant questions which users can actually answer, and at the same time maintain their interest, the context of the user is of crucial importance. For this reason, a considerable part of our KB content is user context, which can be used with the KA rules as the rest of the knowledge. One example of contextual knowledge is the userLocation(USER?, LOCATION?) predicate, which holds the information combined from mobile sensors mining (probableUserLocation(USER?, LOCATION?) and the KA process. Besides userLocation, there is a lot of additional information on the user that the proposed system uses to coming up with personalized questions.

4.4.1 Mobile Sensor Stream Analytics (Mined Context). The central and most important piece of contextual knowledge in our approach is the user’s location and the duration of stay at this location. To facilitate this function, we use an improved implementation of the stay-point detection (Quannan, et al., 2008) algorithm, which is able to cluster raw GPS coordinates and detect when user is moving or staying at a particular location as shown in Fig. 6.

This algorithm in the default implementation clusters the staypoints based on two thresholds: the time \( T_t \) the user needs to stay inside the perimeter \( T_p \), which is the
second threshold. The staypoint is defined with the coordinates (of the user’s mobile device) that all lay inside a given perimeter for at least minimum time: \( r(\text{lat, lon}) < T_p \land t(\text{lat, lon}) > T_t \). This simple algorithm proves to be robust to the usual GPS signal, which is not always returning the same coordinates for the same location and is also lost during the time the user is indoors. Our extension improved the accuracy and robustness of the algorithm by a large factor\(^5\). The improvements to the default staypoint detection can be quickly summarized:

- Extending the algorithm to be an online algorithm by adding temporal dynamic thresholds which are smaller than the default ones. This allows the algorithm to report in real time, quickly responding to possible location changes and reporting whether the user is currently traveling or at a fixed location, and also on which location/path and for how long.

- Applying a dynamic threshold in the first iteration of the algorithms, which is based on the GPS accuracy (the higher the accuracy, the smaller the perimeter threshold) and allows to detect more fine grained changes of the stay-points.

- Fine tuning in a second iteration through the promising staypoints detected in the first iteration. This additionally cleans the potential errors, when one or a few raw GPS locations jumped outside the perimeter but then the sensor stabilizes.

Additional improvement (which is not part of the SPD algorithm) comes from predicting the next user locations\(^6\).

\(^5\) A paper with the full explanation and evaluation of extended SPD algorithm that can be used in any location-based application is in preparation.

\(^6\) These predictions are gathered, but currently being ignored by the Curious Cat – it is the improvement for future work.
Once the raw GPS coordinates are identified as belonging to a staypoint, the algorithm checks the online API's (Foursquare, Factual places, Google Places) and uses the inference engine to identify the most probable location and assigns it to the staypoint (Bradeško, in drugi, 2015). This additional information about place (type and name) together with possible locations can be presented to the user to validate during a conversation with the system.

After the algorithm returns all the information about the location \( P_1' \), the functional component (orange in Fig. 2) asserts the location data into the KB and re-asserts the \( \text{probableUserLocation}(\text{USER}? , P_1') \) statement. Besides the \( \text{probableUserLocation} \), the sensor stream results in more detailed knowledge, for example assertions such as:

\[
\text{visitDuration}(\text{VISIT}? , \text{DURATION}? ), \text{userTimezone}(\text{USER}? , \text{TIMEZONE}? ), \\
\text{userLocalTime}(\text{USER}? , \text{TIME}? ), \text{placesInVicinity}(\text{USER}? , \text{LIST}? ) \\
\text{userPartOfDay}(\text{USER}? , \text{DAYPART}? \{\text{DUSK}, \text{DAWN}, \text{MORNING}, \text{NOON}, \text{EVENING}, ... \}) \\
\text{userActivity}(\text{USER}? , \text{ACTIVITY}? \{\text{WALKING}, \text{DRIVING}, ... \}) \\
\ldots, \ldots
\] (15)

Once the data retrieved from sensors is asserted in the KB through the predicate assertions showed in the example above (definition 15), the KA rules can use that to trigger new and more accurate questions. This, for example, allows the system to ask something at arrival at the location, and something else, which only makes sense then, later. Questions about the quality of food that someone ordered in a restaurant is an example of that latter case.

4.4.2 Knowledge Acquired from the User. Besides the ‘external’ context of the user, we also use ‘internal’ context, which is a specific set of KA rules and KB knowledge directly relevant to the user. By considering the knowledge acquired from the user, we improve relevance of the questions presented to the user. This knowledge is obtained by asking the user questions, such as, the languages spoken, profession, interests, preferred food, etc. The knowledge gathered in this way, is additional to the external context and can be used by the rules to better identify the users who will actually be able to answer particular questions.

4.5 NL to Logic and Logic to NL Conversion

In order to interact with the user, it is not sufficient to form the knowledge acquisition questions in logical form using an inference engine and KA rules. These formulas are understandable by knowledge engineering and math experts, but are not at all appropriate for a direct use in general KA using crowdsourcing from the general population. For this reason, the logical formulas of the sentences and questions need to be translated to natural language to be presented to the user. Similarly, as the user is providing some of the answers in natural language these at least to some extent have to be transformed from natural language to logical form.

This means that in addition to the knowledge itself the KB should include natural language description of the knowledge units, or the natural language generation capabilities must be provided by an external service (the letter is the case in our example with Umko KB (Bradeško, in drugi, 2015). Because natural language generation and conversion is not the main focus of this paper, we present here only a simplified version which explains the basic concepts involved. The actual Curious Cat implementation is based on Cyc NL (Baxter, 2005) and SCG (Schneider, 2015) and consists of more than 90 additional assertions and rules beyond those in the baseline Cyc system to handle language generation.
4.5.1 Logic to NL. When the system already employs an extensible KB, each of the concepts in the KB, can be named using a standard textual string:

\[
\begin{align*}
&\text{nameString}(\text{USER?}, \text{STRING?}), \\
&\text{nameString}(\text{User1, "Luka Bradeško"}), \\
&\text{nameString}(\text{Restaurant1, "Joe's Pizza"})
\end{align*}
\]

(16)

For more complicated language structures, we can encode words as concepts inside the knowledge base as shown in the following example:

\[
\begin{align*}
is(\text{PizzaTheWord, Word}), &\ is(\text{haveTheWord, Word}), \\
&\text{thirdPersonSgNotation(} \text{haveTheWord, "has"}), \\
&\text{singularNotation(} \text{PizzaTheWord, "pizza"}), \\
&\text{pluralNotation(} \text{PizzaTheWord, "pizzas"}), \\
&\text{denotation(} \text{Pizza, PizzaTheWord}). \\
&\text{denotation(} \text{menuItem, list($arg1, list(} \text{haveTheWord, list("$arg2 on the menu")}))}), \\
&\text{denotation(} \text{menuItem, "$arg1 serves $arg2"})
\end{align*}
\]

(17)

Based on that assertions, the system can already convert the logical formula \(\text{menuItem(restaurant1, Pizza)}\) to the appropriate English form: “Joe’s Pizza has pizzas on the menu”, or “Joe’s Pizza serves pizzas”.

When considering generation of questions, we can adopt more or less sophisticated way to form questions out of logical forms. The following question: \(\text{menuItem(Restaurant1, MENUITEM?)}\) there are two ways to do it. The first one, declarative, works in the same fashion as the logic without variables and can be produced with our KB defined in (16 and 17): "Joe’s Pizza has ______ on the menu.”

With additional knowledge, it is also possible to generate a question in the interrogative mode: “What has Joe’s Pizza on the menu?” For this, we need a much more detailed linguistic KB than defined above.

In our Cyc-based implementation, \(\text{haveTheWord} \) alone consists of some 79 language generation assertions (not including the actual uses of the word and assertions inferred from general language rules)

Fig. 7. Actual NL generation assertion example for the corresponding predicate for 'menuItem' from our implementation (Cyc)

There is an additional, perhaps unobvious, benefit from explicit NL knowledge. When we are missing some knowledge on how to convert some predicate or concept to

---

3 An astute reader will have noticed that the sketched KB content does not specify how to produce the third person present for 'haveTheWord' used in the example.
English, the system can simply ask users, how to say something in the language that
particular user knows, and thus the system can make its own NL generation
knowledge part of the KA process as well.

4.5.2 **NL to Logic.** While simple processing based on the same NL knowledge suffices
to interpret isolated terms and denotational phrases from the user responses,
converting general NL expressions back to logic is much trickier than the other way
around, because natural language is much more ambiguous. This process exceeds the
scope of the paper and is described in more detail in (Schneider, 2015). Notice that a
full language-to-logic component was only partly deployed on a trial basis within the
tested system.

For illustration consider inverting the example defined in (17). The system is
presented with the statement “Joe’s Pizza has pizzas on the menu.” Because all the
textual representations of the concepts are indexed, it is able to find the concepts:

- *Restaurant1*
- *menuItem*
- *Pizza*

Because of the *is* and *subclass* knowledge of *Restaurant1* and *Pizza* classes, which can
be observed in Fig. 4, and the *argIs* and *argClass* domain limits of *menuItem* (definition
9), the inference engine is actually able to construct the logical equivalent of the above
sentence: *menuItem*(Restaurant1, Pizza).

Consider, that we have implicitly provided natural language representation of the
knowledge for predicate *restaurantServesCuisine* which is the same as one denotation
for *menuItem* (“serves”). Additionally, we have the knowledge that limits that
predicate to only link to subclasses of class *Cuisine*:

\[
\text{argClass(restaurantServesCuisine, Cuisine)}
\]  

(18)

Then, for the same statement, our system will find the following possibilities:

- *Restaurant1*
- *restaurantServesCuisine*, *menuItem*
- *Pizza*

At this moment, it needs to check all the constraints again, and filter out
*restaurantServesCuisine* because it needs a subclass of Cuisine, which is not in the list
of found concepts (only *FoodOrDrink*, which is *Pizza*). This logical statement would be
invalid due to argument constraints in our KB: *restaurantServesCuisine*(Restaurant1,
*Pizza*). Only the predicate *menuItem* can actually take its arguments with our available
concepts and still be valid in our KB.

These two examples illustrate a simplest kind of sentential NL conversion. When there
are many more possible concepts which need to be combined, the complexity of the
problem quickly explodes, or is ambiguous even given the constraints. For this reason,
our actual Cyc SCG implementation contains much more complex KB structures and
patterns which help with the conversion and a high-speed parser, in Java, that
performs the needed search.

For the most complicated NL conversions, similar approach to a standard ‘ChatScript’
or AIML patterns is taken, just that the patterns are not matched to responses, but to
the logical statements instead. Consider the example mentioned in section 4.2.2,
when user, instead of simple ‘pizzas’ (step 5 in Table I), would say ‘They sell pizzas’ or “They have pizzas on the menu”, or something even more complex. In this case the simple approach described at the beginning of this chapter would not find anything and thus the system falls back to the patterns, like:

\[
\text{[Restaurant] (sell|sells|has) [FoodOrDrink] (|on the menu) \rightarrow menuItem(RESTAURANT?, FOODORDRINK?)}
\] (19)

When finding this pattern, the system would be able to propose the restaurant name based on the pronoun ‘they’ and current location, and the name of the food or drink by the string followed by “on the menu”. Let’s say this happens (an answer is matched to the pattern) immediately after the example question from section 4.3.3 is presented to the user: (ccWantsToAsk(User1, “menuItem(JoesPizza, $x”)”)), then it is not hard to assume that both the pattern and the question refer to the same predicate (menuItem), and thus match them. Please note that these patterns are to be used only when the assisted KA doesn’t present valid options and user types free text and also, that the patterns don’t specify what should be the system’s response. Instead the pattern just converts the text to the logic and then leaves up to the inference engine what to do next. This is completely different approach from standard chatbot systems where patterns dictate the responses as well.

4.5.3 Dialog formulation

As mentioned before, Curios Cat conversation is not pre-defined by the patterns, but is completely knowledge driven. The knowledge is being asserted into the system by users (section 4.6) and automated context (section 4.4), which is then picked up by the inference engine that generates question and commenting intends with the help of KA rules (section 4.3). The resulting assertions, which propose the intents like CCWantsToComment(USER?, FORMULA?) and CCWantsToAsk(USER?, FORMULA?) , are then converted to the natural language and shown to the users, which then respond. This system question/comment and users’ responses form a conversation, which depends on the order of FORMULA? values the system uses to show the question/comment for the user.

For the experiments produced in this paper Curious Cat system tracks additional context of user’s current concept (topic) of interest and also the newly created concepts by that user. For example, JoesPizza and PizzaDeluxe from our previous example. It prefers the formulas which contain these concepts in the given order. Then it simply exploits the feature of the system to produce new questions/comments on the fly as all users (including the one having the conversation) and contextual data are constantly producing new knowledge and consequently new questions/comments. It presents these questions/comments to the user as they come in. For example, when user answers that he is at Joe’s Pizza restaurant, the system produces ccWantsToAsk(User1, “menuItem(JoesPizza, $x”)”), which is showed to the user in NL, as it is the newest formula that appeared and contains the concept of JoesPizza. This simple approach works quite well, since the latest formula is always a consequence of the last user action. Additionally, this can be used for pro-activity, when the question/comment for the user appears as a consequence of some other user’s action, or a context change. If the user didn’t use the system for a while, we can simply present him with the latest formula when it appears from the inference as a proactive question/comment.
For the cases when the system needs more explicit control over the order of the questions, the inference rules can be added using special intent predicates which allow ordering of the responses: \texttt{CCWantsToCommentW(USER?, FORMULA?, IMPORTANCE?),} and \texttt{CCWantsToAskW(USER?, FORMULA?, IMPORTANCE?).} In this case the system will pick the formulas with higher IMPORTANCE number for NL conversion. Additionally, the system tracks the window of user and its own responses with logical assertions like \texttt{CCRresponse(USER?, FORMULA?, NUM?)} and \texttt{UserResponse(USER?, FORMULA?, NUM?)}, which stores the history of the conversation and allows rules like:

\begin{equation}
\begin{aligned}
equals(FORMULA1?, FORMULA2?) \land \text{UserResponse}(USER?, FORMULA1?, 0) \land \\
\text{UserResponse}(USER?, FORMULA2?, 1) \\
\Rightarrow \\
\text{CCWantsToComment}(USER?, "User Repeating Itself ($USER?$")}
\end{aligned}
\end{equation}

This (purely) example rule will produce a comment that user is repeating itself, when the user will respond with the same thing two times in a row.

At last, in the occasions when there are no new questions appearing on the list, the system will pick from the old ones in the order, until something will trigger a new one, or until depleting all of them. If the depletion happens, then the topic will traverse all the concepts from the history and thus accidentally trigger additional inference with new topic of interest for the user, which will produce new questions, which will additionally (if answered) produce new follow-up questions/comments. As the final fallback, the system will randomly pick a concept from its KB and present it to the inference engine as a current interest for the user.

### 4.6 Consistency Check and KB Placement

As already described, we can employ the inference engine to deduce various facts using forward-chaining inference including intents to ask a question. The answers can then be automatically insert into the KB. The knowledge stored in the KB can be then retrieved using logical queries, which return the matching knowledge, or it can infer additional entailed knowledge at query time using backward-chaining inference. The queries are simple logical formulas, which the inference tries to prove or satisfy. For example:

\begin{equation}
\text{class}(X?, \text{Food})
\end{equation}

Will return the following if queried over our example KB (Fig. 4):

\begin{equation}
\text{Food, Bread, Baguette, Pizza, Margherita}
\end{equation}

Or the (somewhat odd but valid) query:

\begin{equation}
\text{is}(\text{Restaurant1, Place}) \land \text{class}(\text{Restaurant, PublicPlace})
\end{equation}

Will return \textit{TRUE}.

For illustration, let us assume that the system finds the following question to ask: “What has Joe’s Pizza on the menu?” - \texttt{menuItem(JoesPizza, FOODORDRINK?)}, with the argument constraints on the predicates as defined in (9). Assuming that the user answers “Pizza Margherita” (hypothetical example, different than in Table I, step 8)
which exists in our KB, then we can ask the inference engine the following type of general query:

\[
\text{denotation}(\text{TERM?}, \text{$answer$}) \land \\
\text{argIs}(\text{pred}, \text{argPos}, \text{ARGIS?}) \land \\
\text{is}(\text{TERM?}, \text{ARGIS?}) \land \\
\left(\text{argClass}(\text{pred}, \text{argPos}, \text{ARGCLASS?}) \land \text{subclass}(\text{TERM?}, \text{ARGCLASS?})\right) \\
\lor \\
\text{unknown}(\exists \text{ARGCLASS?} (\text{argClass}(\text{pred}, \text{argPos}, \text{ARGCLASS?})))
\]

(24)

Where we replace the meta-variables $answer$ (“Pizza Margherita”), $pred$ (menuItem), $argPos$ (the position of variable in the query - 2), with our values from the question and user’s answer, so we get the following actual query:

\[
\text{denotation}(\text{TERM?}, \"Pizza Margherita\") \\
\text{argIs}(\text{menuItem}, 2, \text{ARGIS?}) \land \\
\text{is}(\text{TERM?}, \text{ARGIS?}) \land \\
\left(\text{argClass}(\text{menuItem}, 2, \text{ARGCLASS?}) \land \text{subclass}(\text{TERM?}, \text{ARGCLASS?})\right) \\
\lor \\
\text{unknown}(\exists \text{ARGCLASS?} (\text{argClass}(\text{menuItem}, 2, \text{ARGCLASS?})))
\]

(25)

If this query is asked in our small KB, we get the following results:

<table>
<thead>
<tr>
<th>TERM?</th>
<th>ARGIS?</th>
<th>ARGCLASS?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Margherita</td>
<td>/</td>
<td>FoodOrDrink</td>
</tr>
</tbody>
</table>

Table IV. Results of query (25)

Because we obtained the results from the query we can immediately know that the answer is consistent with the KB and it is safe to assert it into the knowledge base as: menuItem(JoesPizza,Margherita). Here it is worth noting that we allow the variable ARGIS? to be unbound because of the logical disjunction in the query. In the real implementation, this is not allowed, so we add additional disjunction to bind it to the concept Nothing.

At this point we have seen an example of how to add new knowledge when the concept that the user has provided already exists in the KB and the answer is structurally valid. But what would happen if user should say “Pizza Deluxe” (as in our example in Table I, step 8), which we do not have in the KB. In this case the query above would return nothing. This would happen as well, if the user should say “car”. When the validation query does not return results (as would happen for “car”), we need to separately check whether the concept exists:

\[
\text{denotation}(\text{TERM?}, \"car\")
\]

(26)

If it does (i.e. the query 26 above returns the resulting concepts), then the answer is actually invalid on structural grounds (the term it includes ca not be used a viable answer). But, if the concept does not exist yet (nothing returned, as would be the case for “Pizza Deluxe”), then we can simply create it together with its NL denotation and assert it using our question predicate.

4.6.1 Detailed placement in the KB. In addition to deciding to add some newly acquired knowledge to the KB, the exact location in the “KB graph” is determined by the argIs
and \textit{argClass} assertions on the question predicate. Consider our example question: \texttt{menuItem(joesPizza,x?)}. Because of the \textit{argClass}(\texttt{menuItem}, 2, \texttt{FoodOrDrink}), the answer must be a subclass of \texttt{FoodOrDrink} concept. Let us say that the answer is: “Pizza Deluxe.”

\textbf{Fig. 8. FoodOrDrink part of the class hierarchy from our example KB}

To insert the knowledge, the system goes through the steps described in the section 4.6 (Consistency Check and KB Placement). Because of \textit{argClass} assertions \texttt{PizzaDeluxe} concept is added as a subclass of \texttt{FoodOrDrink} as illustrated in Fig. 8. While this is logically valid, it is not detailed enough to satisfy our KA requirements. For this reason, when the concept does not already exist, or is not detailed enough (too high in the class hierarchy), we can issue additional query:

\begin{equation}
\texttt{subclass(TERM?, FoodOrDrink)}
\end{equation}

Which, for our example knowledge base returns:

\begin{equation}
\texttt{TERM?: FoodOrDrink, Drink, Coffee, Food, Bread, Baguette, Pizza, Margherita}
\end{equation}

This gives various options. A) Ask the user, which one of these PizzaDeluxe is (excluding the main class \texttt{FoodOrDrink}). For example: “\texttt{What describes it in most detail?}”, or: “\texttt{Is Pizza Deluxe a drink, coffee, bread, baguette, pizza or margherita}”. B) Ask for the first level subclasses first, then for the next level, etc.: 1. “\texttt{Is Pizza Deluxe a type of drink or food}?” 2. “\texttt{Is Pizza Deluxe a type of pizza}?”.

Or C) (which is usually the best option) we can scan all of the resulting concepts (definition 28) for their NL \textit{denotations} and then match the strings to “Pizza Deluxe”, to see which one fits the best. In this case, only the \texttt{Pizza} concept provides a partial match. So, we can immediately ask: “\texttt{Am I right that Pizza Deluxe is a type of Pizza}?” If the user agrees, we can, in addition to \texttt{subclass(PizzaDeluxe, FoodOrDrink)}, add: \texttt{subclass(PizzaDeluxe, Pizza)}. This results in our new concept being located in the KB at the most descriptive place as illustrated in Fig. 9.
4.7 Crowdsourcing Mechanisms

Up to this point, we have discussed the mechanisms of KA, showing how it is possible to get valid knowledge from a single user. In order to lower the cost of and/or increase the speed of KA we can involve a crowd. Crowdsourcing however bring several challenges (as described in Section 2.5) including the following:

- User privacy
- Users making deliberately false claims or having mistaken ideas about the world, and
- The ever-changing state of the real world

We tackle this by organizing our KB into smaller, hierarchical knowledge base structures. Each of these structures is then our virtual knowledge base in which we operate, and which has its own independent knowledge, added on top of all the sub-KBs up in the hierarchy (Fig. 10). In the Cyc system, these contextual KB structures are called Microtheories (Cycorp, 2012).

![Hierarchical structure of Knowledge Bases](image)

Fig. 10 Hierarchical structure of Knowledge Bases

The idea is that the sub-KBs which are higher up in the hierarchy cannot see the knowledge that is at the lower levels. But the KBs which are at the lower levels, contain all the knowledge of their ancestors from the higher levels. For example, let us assume that *WholeWorldKB* in Fig. 10 has the knowledge defined in section 4.3.1 and 4.3.2, *CuriousCatBelievesKB* has the knowledge defined in section 4.3.3 and then acquired from multiple users. *User1KB* has knowledge that was acquired from User1.
and user2KB has knowledge acquired from User2. Following our KB structure given in Fig. 4 and formal logical definitions, we can see that:

- **WholeWorldKB**: contains the upper ontology and pre-existing knowledge from Fig. 4.
- **CuriousCatBelievesKB**: contains KA assertions and union of assertions acquired from all the users.
- **User1KB**: contains assertions acquired from user 1 and only relevant to her/him.
- **User2KB**: contains assertions acquired from user 2 and only relevant to her/him.

Following this structure, it becomes obvious that each user in our system has its own sub-KB, connected to the main knowledge only through CuriousCatBelievesKB. Also, User1KB and User2KB cannot see each other’s knowledge, but only perceive the world through the “eyes” of CuriousCatBelievesKB and their own local sub-KB. This means, that if User1 lies about something, the wrong knowledge will be only available to User1, while the rest of the users will not be affected. In this way, the user’s privacy is also protected.

To benefit from crowdsourcing we want to share some of the knowledge to all the users. Since each sub-KB can “see” only the assertions stored in itself and the assertions higher up in the hierarchy, we can control what only one user knows, versus all the users, by moving the specific assertions up or down through the hierarchy. We are proposing two approaches: 1) Crowdsourcing through repetition (section 4.7.1), and 2) Crowdsourcing through voting (section 4.7.2).

Crowdsourcing 1 is more advanced, since it includes type 2 as well, once the repeated assertions from multiple users got promoted to CuriousCatBelievesKB. Because it only promotes the knowledge which is asserted by multiple users independently, there is reduced chance for temporal wrong knowledge staying in our system before the voting from approach 2 removes it. In this sense, Crowdsourcing through repetition gives better results, but it has a drawback, especially if there is not enough of users in the system, the knowledge takes longer to be promoted to the main KB where other users could benefit from it. Additionally, users might have a feeling that they are the only one in the system if they don’t see some immediate feedback and activity. Initially we started with Crowdsourcing through repetition, but then decided (due to reasons above) to do our initial experiments (described in section 5) by using the option 2 (section 4.7.2).

4.7.1 Crowdsourcing through repetition is based on the number of identical assertions in all the sub-KB’s on the same level (i.e., different users providing the same knowledge). Once the specific assertion count is above a threshold, we assume that there is enough evidence for promoting it to general knowledge, which is performed by moving the knowledge to higher level of the hierarchy (via “lifting rules” in the CuriousCatBelievesKB), and thus making it visible for all the users. After the knowledge is in the public KB, crowd users can start voting on it (see section 4.7.2 - Crowdsourcing through voting).

Consider the case of User1 answering the question from the previous examples: “What did you order?”, with a lie: “spicy unicorn wings”. The system will go through steps described in section 4.6 - Consistency Check and KB Placement), and if the user confirms this new concept, Curious Cat will believe (in the world for User1), that he ate spicy unicorn wings and that the JoesPizza restaurant has them on the menu. The contextual KB User1KB would then get the assertion menuItem(JoesPizza,SpicyUnicornWings). There is a very low chance that any other
user would provide the same answer, so the wrong assertion will never get promoted to higher levels of KB.

On the other hand, if User1 answers `menuItem(JoesPizza, PizzaDeluxe)`, then User2 answers the same, then User3, the assertion already has a count of more than 2, which is the threshold in our system, and the assertion will get promoted to be visible for all the users (world). For newcomers to JoesPizza, the system will already know that they have “Pizza Deluxe” on the menu.

![Screenshot of Curious Cat displaying an already public information about Joe's Pizza to a visiting user.](image)

**Fig. 11.** Screenshot of Curious Cat displaying an already public information about Joe's Pizza to a visiting user

4.7.2 Crowdsourcing through voting. Contrary to the previous example, we can also promote all the answers immediately. In this case, all the assertions except the private ones (e.g. ones that contain the concept for the `User#`), will get asserted into both the user’s KB and the `CuriousCatBelievesKB`.

Once the assertions are in `CuriousCatBelievesKB`, users will not get the standard questions produced by the KA rules, but will still get occasionally the “crowdsourcing” questions (produced by different rules), which will simply be checking the truth of some existing knowledge: “Is it true that Joes Pizza has Pizza Deluxe on the menu?”. These simple yes/no questions allow us to assess the truthfulness of the logical statement and remove it, if it gets more negative answers than positive.

This voting mechanism serves for promoting knowledge (as described in 4.7.1 - Crowdsourcing through repetition) and for detecting when the world changes and something that was true in the past is not true anymore.

5. VALIDATION AND RESULTS

Validation of the proposed approach was performed using our Curious Cat system running alive over the course of 4 years engaging a total of 728 registered users. During that time, the users checked-into 5,551 locations and responded to 57,978 questions, of which 8,611 were voting questions (7,560 positive and 1,051 negative votes), 18,907 answered with “I don’t know”, and 30,460 real answers inserted into the KB as new knowledge, including 31,140 concepts (3,171 on users, 22,563 check-ins and other places and 5,406 other concepts). These triggered additional 386,980 assertions to be
added through forward-chained inference. These can be separated into facts (374,300 assertions) and additional derived question generating rules and questions (12,680). Altogether we gathered **444,958 (374,300 + 30,460)** pieces of completely new knowledge.

For easier understanding, we show the structure of the collected knowledge graphically in Fig. 12, and give real examples of the collected and inferred knowledge in Table V.

**Table V. Examples of answered/asserted and inferred knowledge taken from Curious Cat KB**

<table>
<thead>
<tr>
<th>Type of the collected knowledge</th>
<th>Curious Cat Question</th>
<th>User Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>User concepts</td>
<td>[automatic assert from registration]</td>
<td>Person (CCUser5), Name (CCUser5, “Luka”)</td>
</tr>
<tr>
<td>Places + check-ins</td>
<td>[automatic assert from check-in and Foursquare/Factual locations]</td>
<td>Restaurant (CCPlace7), Name (CCPlace7, “Pig n Whistle”)</td>
</tr>
<tr>
<td>Other concepts</td>
<td>“What kind did you order?”</td>
<td>Duck meat [food(DuckMeat)]</td>
</tr>
<tr>
<td>Real knowledge answer 1</td>
<td>Who is CEO of BMW? / CEO of BMW is ___</td>
<td>Harald Kruger</td>
</tr>
<tr>
<td>Real knowledge answer 2</td>
<td>What is the ticker symbol of BMW?</td>
<td>ABC</td>
</tr>
<tr>
<td>Voting (yes)</td>
<td>Is it true that Harald Kruger is the CEO of BMW company?</td>
<td>Yes</td>
</tr>
<tr>
<td>Voting (no)</td>
<td>Is it true that ABC is the ticker symbol for BMW?</td>
<td>No</td>
</tr>
</tbody>
</table>
While the resulting number and quality (Table XI, Table XII) of assertions shows that the presented approach can be successfully used for high quality KA, we explored in more detail the contributions of specific characteristics of our system and compare them to the baselines, to evaluate the ideas and claims (hypotheses) of the paper:

A. Using context to proactively drive the KA increases engagement and the chance of getting an answer (section 5.1)
B. Pre-existing knowledge and automated inference can be used to filter-out inconsistent answers and thus increase the quality of the acquired knowledge
C. Using newly acquired knowledge to further drive the KA process increases the amount of acquired answers and reach of the system
D. Crowdsourcing additionally improves the results by filtering wrong but logically correct answers

### 5.1 Context and pro-activity

Curious Cat employs two contextual knowledge provision mechanisms. One is location based knowledge, such as, the type of current location, the time of visit and the duration of stay. The second is internal knowledge that the system already knows about the user, such us, languages that the user speaks, food she likes, demographics data, interests, etc. In evaluation, we focused on exploring benefits of using the location based context, since it is practically impossible to disable internal knowledge without bigger changes in the system. Inferring without internal knowledge e.g., total removal of the personal knowledge would mean removal of almost all questions regarding people, animals, human interests, etc., since these are general for human beings and to some extent to animals as well.

For the purpose of this experiment, we deliberately removed GPS location and connected mechanisms for derived knowledge for a duration of three months. We normalized the measures (assertions per user per day) for all experiments to level out the differences in the durations of experiments against a longer duration of fully operative system. The results are presented in Table VI. The columns in the table are organized as follows:

- first column contains measure name,
- second column full data-set when using context (marked as C),
- third is 100-day data-set when not using context (NC100)
- and last is the normalized column with context (C100) which makes it possible to compare the context versus no context knowledge collection behavior (C100 vs NC100).

Since number of days and users is strongly correlated with the number of answers the system gets, and we only have the “No Context” data for 100 days (41 new users during that time and 45 active users), we normalized our “Context” data in such a way, that it matches the duration, new users and active users of NC100. For this, we had to scan all the data-set day-by day with a range of 100-day window, where we looked at the number of new and active users in each of the 1264 sub-windows (number of all different 100 day windows one can fit into 1264 days). In whole data-set there was 18 such 100-day windows, which can be directly compared to the NC100. While the number of new users is fixed to 41, number of new
assertions and active users still varies (min. 1,429, max. 9,279 new assertions and min. 47, max 61 active users). For this reason, we calculated the mean values of all 18 windows which had the same number of new users as NC100. Matching new users (instead of active users) was selected because new users are the ones that contribute most of the assertions, since their interest winds down gradually. This can be seen on Fig. 13. This is also the primary reason why (unintuitively) number of new assertions/day/active user (row 7 in Table VI) is higher without context (NC100) than for non-normalized data from experiment using context (C).

Table VI. Results of KA without context versus results while using location based context

<table>
<thead>
<tr>
<th>Measure</th>
<th>With Location Context (C)</th>
<th>Without Location Context (NC100)</th>
<th>With Location Context 100 days (C100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Experiment duration</td>
<td>1,364 days</td>
<td>100 days</td>
<td>100 days</td>
</tr>
<tr>
<td>2. Number of new assertions</td>
<td>56,586</td>
<td>709</td>
<td>2,244 (+216.5%)</td>
</tr>
<tr>
<td>3. Number of new assertions where user didn’t know the answer</td>
<td>18,380 (32%)</td>
<td>267 (38%)</td>
<td>667 (28.3% = -9.7%)</td>
</tr>
<tr>
<td>4. Number of new assertions where user knew the answer</td>
<td>38,206 (68%)</td>
<td>442 (62%)</td>
<td>1,577 (71.7% = +9.7%)</td>
</tr>
<tr>
<td>5. Number of new assertions/day (all users)</td>
<td>42.1</td>
<td>7.1</td>
<td>22.4 (+215.5%)</td>
</tr>
<tr>
<td>6. Number of new assertions/active user</td>
<td>90.3</td>
<td>15.7</td>
<td>44.2 (+181.5%)</td>
</tr>
<tr>
<td>7. Number of new assertions/day/active user</td>
<td>0.07</td>
<td>0.16</td>
<td>0.44 (+175%)</td>
</tr>
<tr>
<td>8. Number of new concepts/day</td>
<td>3.7</td>
<td>0.4</td>
<td>2.4 (+85.2%)</td>
</tr>
<tr>
<td>9. Number of new concepts (excluding users and locations)</td>
<td>4,925</td>
<td>36</td>
<td>243 (+85.2%)</td>
</tr>
<tr>
<td>10. Number of active users</td>
<td>625</td>
<td>45</td>
<td>49</td>
</tr>
<tr>
<td>11. Number of new users</td>
<td>625</td>
<td>41</td>
<td>41</td>
</tr>
</tbody>
</table>

Source: Counts from the Curious Cat KB and logs database

The results of not using context show the decrease of “raw” new knowledge assertions from 56,568 to 709, which of course can be attributed to the fact that duration of experiment C was much longer, or that it had more users than NC100. As described above, this is properly normalized in the column C100, where the raw number of assertions raises to 2,244 compared to only 709 when not using context (context brings 217% increase).

While the majority of the increase is due to pro-active questions from Curious Cat, which are linked to GPS clustering, some of the increase is also due to better targeting of the questions due to contextual data, since without the context only users are selecting the topics and the system doesn’t have enough knowledge to successfully pick the most relevant next question. This is reflected in a slight increase in the proportion of the questions for which the user knows the answer (+9.7% - row 4 in Table VI). This is independently of pro-active component which is mostly reflected in the raw number of assertions, while the knowing/not knowing ratio improvement is due to better targeting whatever question there was.

5.2 Consistency Checking

Before the system accepts the answer from the user (as described in section 4.6), it converts it into logic and checks whether it is consistent with its current knowledge.
there is an inconsistency, it will reject the answer and thus prevent the wrong or contradicting knowledge to corrupt the KB and consequently the future KA process. The user then either has to fix the answer, or convince the system that the answer is actually only a different name for the consistent knowledge. For example, if the user states that he ate a car in the restaurant, this is either wrong, or ‘car’ is the name for some unknown food and not a vehicle.

Due to development process and timeline of the system, we only have logs and consequently insights on when/how the system rejected the answers due to inconsistency for 143 out of 1,472 days of the experiment duration. During this time, 148 users provided 23,543 answers and the system rejected 563 of them, so 22,980 of assertions went into the KB. Among the 563 rejections, some of them were repeating (user re-tries, or other users did the same mistake), so the system actually prevented 384 unique inconsistent answers.

As presented in Table VII, for every 100 valid answers users provided during the experiment, there were 2.45 answers which were rejected by the system due to being completely wrong or inconsistent. Because we only have logs for 143 days we had to extrapolate the number for the overall experiment to get some estimation of how many bad assertions were prevented by the system throughout the whole experiment. The estimated number of rejected assertion system prevented overall is 1,420.

Table VII. Results of consistency checking

<table>
<thead>
<tr>
<th>Measure</th>
<th>Real data from the logs</th>
<th>Extrapolation for the full experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of new assertions</td>
<td>22,980</td>
<td>57,978</td>
</tr>
<tr>
<td>Number of rejected asserts due to inconsistency</td>
<td>563</td>
<td>1420</td>
</tr>
<tr>
<td>Percentage of rejected asserts</td>
<td>2.45%</td>
<td>2.45%</td>
</tr>
<tr>
<td>Number of unique rejected asserts</td>
<td>384</td>
<td>968</td>
</tr>
<tr>
<td>Percentage of unique rejected asserts</td>
<td>1.67%</td>
<td>1.67%</td>
</tr>
<tr>
<td>Number of new users</td>
<td>128</td>
<td>728</td>
</tr>
<tr>
<td>Number of active users</td>
<td>148</td>
<td>728</td>
</tr>
<tr>
<td>Experiment duration</td>
<td>143 days</td>
<td>1472 days</td>
</tr>
</tbody>
</table>

Source: Counts from the Curious Cat KB and logs database

As presented in Table VIII, for every 100 valid answers users provided during the experiment, there were 2.45 answers which were rejected by the system due to being completely wrong or inconsistent. Because we only have logs for 143 days we had to extrapolate the number for the overall experiment to get some estimation of how many bad assertions were prevented by the system throughout the whole experiment. The estimated number of rejected assertion system prevented overall is 1,420.

Table VIII. Conversation prior to rejected assertions

<table>
<thead>
<tr>
<th>CC</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1</td>
<td></td>
</tr>
<tr>
<td>Which social being has</td>
<td>hotel</td>
</tr>
<tr>
<td>Lagos as assets?</td>
<td></td>
</tr>
<tr>
<td>By hotel do you mean</td>
<td>yes</td>
</tr>
<tr>
<td>hotel (building)?</td>
<td></td>
</tr>
<tr>
<td>Rejected – building is not a social being and cannot be owner</td>
<td></td>
</tr>
</tbody>
</table>

Example 2

| Let’s continue our previous conversation about going on a date. | ok |

ACM Transactions on xxxxxxxx, Vol. xx, No. x, Article xx, Publication date: Month YYYY
All going on a date has ______ as a characteristic sub-event.

Rejected – hello is a greeting and not an event

Source: Counts from the Curious Cat KB and logs database

5.3 Additional Knowledge as a follow up to newly acquired knowledge

As one of the more important features of our proposed system is its ability to use newly acquired knowledge to ask new things, we explored this functionality, comparing it to the baseline KA and consequently measuring the knowledge that was acquired as a consequence of prior acquisition. An example of such a ‘follow up’ question or assertion is given in Table IX.

Table IX. Example of follow-up question and follow-up answer(knowledge)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CC Asks: “What did you order”</td>
</tr>
<tr>
<td>2</td>
<td>User answers: “Meatloaf”</td>
</tr>
<tr>
<td>3</td>
<td>CC asks a follow-up question: “One of the Meatloaf ingredients is ______.”?</td>
</tr>
<tr>
<td>4</td>
<td>User answers: “egg”.</td>
</tr>
</tbody>
</table>

Source: Counts from the Curious Cat KB and logs database

The question number 3 in Table IX is a “follow-up” question since it is only asked when someone enters the previously unknown food concept. Answer under number 4 is a follow-up knowledge, since it was answered and given to the system only because someone created the concept of ‘Meatloaf’.

During the whole experiment the system collected and accepted 22,838 answers, which were a follow-up answer. This means that 39.4% of all the knowledge we obtained was a direct consequence of the system’s ability to generate new questions based on the already acquired knowledge.

The follow-up questions and answers are always a consequence of some new concept that enters the system. The 22,838 of answers are consequence of 5406 new concepts (not counting concepts of users themselves and locations they check-in), where the distribution of new knowledge/concept follows a long-tail distribution - a very small number of concepts triggered a big number of follow-up knowledge (Fig. 13 – logarithmic scale).
5.4 Additional knowledge because of allowing cross-user cooperation

Similarly, as done for follow-up answers in section 5.3, we can check how much of the new knowledge came into the system due to its ability to share knowledge and questions cross-users. For this, we check the answers, where the initial concept was given to the system from a different user then the follow up answer. From the example in Table IX this would mean that the answer number 2 was given by User1 and then CC asks the question 3 and receives answer 4 from User2.

While for follow-up results, we have a complete log, due to changing the system, not all records have information about the original user who created the concept. We have his data only for 45 concepts out of 5406, excluding all user concepts and locations they checked-in. For these 45 concepts, we got additional 123 answers from other users.

5.5 Voting

The voting part of the system is, besides directly providing the answers, one of the main crowdsourcing components. As described in section 2.5, after the knowledge is acquired, the system can re-check with other users whether the assertion is true or not. This is done by presenting a question in the form: “Is it true that [assertion].”, which can be answered with “yes” or “no”. For example: “Is it true that Bornholmsk is a national language of Denmark?”.

While the consistency check is performed automatically by the system, the answers that are manifestly inconsistent with the other content in the KB are stored only in the user specific part of the KB. Checking for truth of the knowledge is motivated by the fact that claims can be still be untrue even if they are consistent with the KB. From the Yes/No ratio, we can see that there are many more “Yes” votes than “No” votes, which hints that the answers the other users provide are mostly recognized as true by the crowd. This could be taken as a hint towards the precision of the truthfulness of
the acquired knowledge. If we take into consideration that more users can vote for the same assertion, the effective precision of the knowledge measured this way can be estimated more accurately. The voting mechanism hides from the public knowledge base 97 of the assertions where the users were unable to agree on truth and 636 assertions which were voted as untrue by majority of the users.

Table X. Crowd voting results

<table>
<thead>
<tr>
<th>Measure</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>All votes</td>
<td>8,611</td>
</tr>
<tr>
<td>Votes over unique asserts</td>
<td>5,436</td>
</tr>
<tr>
<td>Yes votes</td>
<td>7,560</td>
</tr>
<tr>
<td>No votes</td>
<td>1,051</td>
</tr>
<tr>
<td>Rejected answers</td>
<td>636</td>
</tr>
<tr>
<td>Accepted answers</td>
<td>4,703</td>
</tr>
<tr>
<td>Undecided answers</td>
<td>97</td>
</tr>
</tbody>
</table>

*Source: Counts from the Curious Cat KB and logs database*

If we look at some of the most rejected assertions, we can see assertions such as (more no votes than yes):
- `nationalLanguage (Denmark Bornholmsk)` (3 yes, 9 no)
- `typicalColorOfType (Automobile BlackColor)` (3 yes, 9 no)
- `isa (CityOfCopenhagenDenmark Town)` (0 yes, 4 no)
- `subclass (Coffee-Beverage ArtificialMaterial)` (3 yes, 9 no)
- `soleMakerOfProductType (TelevisionSet PhilipsPetroleumCompany)` (3 yes, 7 no)

And the most accepted assertions (more yes votes than no):
- `isa (Denmark CountryWithOnlyOneTimeZone)` (25 yes, 0 no)
- `isa (Food CollectionWithManySpecializations)` (23 yes, 0 no)
- `subclass (Food OrganicMaterial)` (19 yes, 0 no)
- `isa (Food ProductTypeWithoutSoleMaker)` (20 yes, 1 no)
- `countryPhoneCode (Slovenia 386)` (18 yes, 0 no)

And some of the undecided assertions (same number of yes and no votes):
- `subclass (IceCream Mixture)` (2 yes, 2 no)
- `characteristicProperSubeventTypes (CookingFood EatingEvent)` (1 yes, 1 no)
- `electronicDeviceMountingStyle (RearVideoPort DesktopComputer)` (2 yes, 2 no)

5.6 User Stickiness factor

To get an idea of how the users interacted with the system, we checked each of them by the first and the last piece of knowledge they provided and measured the duration of days between these dates. This is a simple indicator of a stickiness – showing how many of the users only tried an application and then forgot about it and how many of them answered questions for longer. On Fig. 14 we can see the long-tail distribution of how long users stick with the application. About 25% of the users stick with the system for more than just one-day testing. While most of the users (548) only tested the system for a day and did not use it further, 180 users stayed for a longer period of time. Among these, 117 used it for more than a week, 98 for more than two weeks, 69 for a month or more, 51 for more than two months and 15 for more than a year.
While this distribution is not too impressive for the commercial product, it is somehow expected for a research prototype without any updates and improvements for more than a year after it was built. It shows that the approach has potential and triggered interest in a decent number of the users, which indicates it could gain a lot of interest with a few improvements and obvious bug fixes.

5.7 Proof of concept usage of gained knowledge

The end goal of the KA is not the KA itself, but to be able to collect good enough knowledge to be able to use by the inference to solve some higher-level tasks previously not possible. While this is not the goal of the paper, we added a few inference rules which try to combine the gained knowledge about the user, his likes and whereabouts and the knowledge about the restaurants and foods, to produce relevant suggestions when user requests them, or when the system thinks the user is hungry (no eating reported for a while). Additionally, the system is able to infer, based on the events around the user and his likes, when to tell him some information regarding that.
On the left screenshot of Fig. 15 we see that Curious Cat ‘smartly’ told the user that there is a (jazz) music event nearby, after it learnt that the user likes jazz music. The event was actually entered by another user and the system picked it up from the KB for the purpose of helping new user. On the right screenshot, there is a similar ad-hoc comment, but this time the trigger was cheaper coffee in the bar next-door. The suggestion is immediately followed by the comparison question, to be able to better rank the coffees next time.

A more complex inference suggestion when user asked for something fast to eat is presented on the Fig. 16. There the inference engine combined multiple pieces of knowledge:

1. “Hot horse” serves fast food cuisines, therefore it must be a fast food restaurant
2. User likes fried food
3. French fries are a type of fried food
4. “Hot Horse” has French fries on the menu
5. Since “Hot Horse” is a fast food restaurant which is near the user, and has something on the menu that the user likes, it is among the good suggestions.

This inference process was then structured into the set of logical statements (special inference rule for suggestions), sent through the NL conversion (section 4.5.1) and presented to the user as the suggestion text.

While this suggestion is a simple basic example, still shows the power of combining separate piece of logic by inference and produce some useful results. The step from liking the fried food to French fries on the menu would be quite hard to tackle with
standard approaches, especially if we consider it just an example of a general mechanism.

Fig. 16: Suggestion as a complex inference result combining knowledge about cuisines, user likes and place specifics.

5.8 Results Summary

As a wrap-up of the result sections, Table XI shows the knowledge or quality contributions of particular system features and their appropriate baselines.

<table>
<thead>
<tr>
<th>System feature</th>
<th>Measure</th>
<th>Baseline</th>
<th>Proposed approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge Gathering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context and proactivity</td>
<td>Number of gathered assertions/day</td>
<td>7.1</td>
<td>42.1</td>
</tr>
<tr>
<td>Follow-up assertions (new knowledge due to already</td>
<td>Number of gathered assertions</td>
<td>35,140</td>
<td>57,978</td>
</tr>
<tr>
<td>collected knowledge)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Number of gathered assertions/concept (avg.)</td>
<td>0</td>
<td>4.22</td>
</tr>
<tr>
<td>Cross-user assertions</td>
<td>Number of gathered assertions (extrapolated)</td>
<td>43,920</td>
<td>57,978</td>
</tr>
</tbody>
</table>
As the overall sanity check of the KB, we randomly picked 100 newly acquired assertions, and assessed them manually to see whether they were:

- Valid (in the sense of consistent with the KB – here we expect 100%)
- True (in the sense of true in an interpretation based on our human world)
- Useful (useful for a potential user, or for the inference engine in producing suggestions, validations, new questions…)

The counts are presented in Table XII. As expected all the assertions are valid, 96% of them are true and 95% are useful. This can be used as an estimate of the knowledge that we acquired through the proposed approach.

The results of this counting are not surprising, since the system doesn’t allow manifestly inconsistent assertions, and the crowdsourcing mechanism already weeded out most of the untrue examples. There were four untrue assertions in the sample, which expose three potential problems:

1. One error was because the price of the coffee changed since the last check with the users – the stored knowledge was obsolete
2. There were two concepts with exactly the same name and were both subclasses of Organization, so the users and also the system didn’t manage to distinguish them and picked the wrong one. For users, everything seemed perfectly correct even though a logical error resulted
3. Twice the user entered a complex sentence instead of a name of the concept and that forced the system to create a concept with that name

Looking more in details of the usefulness of the retrieved knowledge, there were two answers, which related to the internal KB mechanism (how to handle events) and were thus not really useful for the end user. The remaining there were mostly too specific and not really useful in general, such as the name of the spider the user has in the corner of a room.

6. CONCLUSIONS

The proposed novel approach to mobile, context aware conversational crowdsourcing knowledge acquisition is able to gather new knowledge of a very high quality in a never-ending fashion analogue to never ending language learning proposed in NELL (Mitchell, et al., 2015). This is achieved by using an existing knowledge base, user’s
current and past context and, employing a targeted crowdsourcing methodology. We propose to use highly focused context to support targeting the right users at the right time. This had not to our knowledge been tried before. We have implemented the proposed approach in Curious Cat system running online as an experiment for 4 years. During this time, it collected a substantial amount (57,978 user answers resulting in 386,980 assertions) of consistent and highly accurate knowledge and thus proved the proposed approach to be feasible and worth exploring. The evaluation also shows that all of the main features of the approach are contributing to the quality and quantity of the collected data, while making users interested enough (25% using the app for more than a day, 2% for more than a year), besides being in a research prototype state. The context and proactivity increase engagement from 7.1 assertions/day to 42.1/day.

Using newly acquired knowledge to produce more and better questions acquired 39.4% of all the knowledge. On the other side consistency checking improved the knowledge base by preventing bad assertions (2.41% of all user answers) and crowd voting prevented consistent but otherwise wrong or untrue assertions (1.26% of all user answers).

The implementation of the prototype required approximately 2.5 person years, on top of ~930 person years spent on Cyc system. While we implemented our prototype on the top of Cyc (Lenat, 1995), the approach itself is general and can be applied to any knowledge base or inference engine by implementing the necessary technical adjustments.

Similarly, as the approach is inference engine and knowledge base agnostic, it is also not fixed to any particular domain or type of the concepts or assertions. Of course, some adjustments would be needed to ensure relevance of the pre-existing knowledge for the targeted domain. For the purpose of the experiment, most of the KA rules were added for the subclasses of public places, restaurants and bars, which made the system more responsive and produce more questions/comments for these domains. But this didn’t prevent the users, or the system to have a KA conversation about any other domain or concept, including (taken from our logs) concepts of Love, ImaginaryFrient, Tennis, Winter, Whining, Electronics, Balkan, TopGear, Swimming, etc. Additionally, by preferring general KA rules, for example what to ask for each location instead only for locations of type of Restaurant, this approach could quickly gain enough coverage for all types of the locations for users to not get bored, or the system to not run out of questions. The limit is basically not on the approach, but on the existing knowledge that the approach is hooked to. With using Cyc as the main KB the coverage of the domains is automatically quite broad (~500,000 concepts trying to address as much aspects of human knowledge as possible), without the ones newly acquired from the users.

From the validation in Section 5, we can see that the resulting knowledge has high quality and is easily and inexpensively gathered from non-expert users, while they are having a conversation with the system in order to satisfy their information needs. Validation of the system also hinted at potential issues, which are to be addressed in the future work. The first of them is related to knowledge potentially becoming obsolete. This can be addressed by using time constraints and assert meta-knowledge to ensure that if some knowledge has not been confirmed for a while, it should be checked with the users for its validity. The second is related to having several possible answers to the user’s information need due to several concepts having the same NL presentation. This can be tackled to some extent by showing to the user all the answers using alternative NL presentations when available. Additionally, the system could provide more information on each of the answers, for instance, describe each with some
unique predicate that holds for it. The third issue of the user entering a complicated sentence can be tackled simultaneously while improving the conversational client rules.

One of the promising directions for the future work is, for Curious Cat to become a full conversational engine (or to be merged with some existing one), which would make it easy to construct knowledgeable chatbots. This would allow improvement of the system to accept and understand more complex answers and questions from the users, and to not limit itself only to short replies and guided responses. This is to some extent similar to the current AIML and ChatScript systems, except that Curious Cat is completely knowledge driven and designed to be able to extend itself indefinitely while talking with the users. Patterns are utilized purely to convert from text to logic and not to dictate the responses. To still ensure quality of the knowledge base, the free text answers of the users that are providing new knowledge will be first checked to determine whether they can be parsed into a more complex logical sentence, as already described to some extent in section 4.5.2 through SCG technology. For this extension, the amount of work to increase the coverage should be much smaller compared to the current approaches due to knowledge and KA support specifics of our system.

ACKNOWLEDGMENTS
This work was partially supported by the Slovenian Research Agency and the ICT program of the EC under project OPTIMUM (H2020-MG-636160). We thank to all of our 728 users who contributed to the experiment and the KA system, and Dr. Dave Schneider in particular and Cycorp management and members of the Cycorp technical staff (including Chris Deaton and Dr. David Baxter) more generally for licensing, and for their kind assistance with APIs, KR advice and other assistance as we constructed and tested Curious Cat. We also thank Abe Hsuan and Dane Sulić for advices and design assistance, respectively.

Note: The overwhelming majority of this work, except writing, was done while Michael Witbrock was acting as a principal at Curious Cat Company and concurrent with his time at Cycorp Inc.

REFERENCES


