Ontology engineering for robotics

Frank Schröder

November 20, 2017

Germany

Abstract

Ontologies are a powerfull alternative to reinforcement learning. They store knowledge in a domain-specific language. The best-practice for implementing ontologies is a distributed version control system which is filled manually by programmers.

Keywords: Robotics, ontology engineering

Contents

1 Introduction

to do: ontology + neural network

In today robotics, the most dominant challenge is the optimal control problem. In short the question is how to adjust the servo motor of a robot to fulfill a given task. The naive approach which utilizes the game-tree of all possible actions and search for an optimal solution is failing because of the huge amount of states. Even if the robot has only 3 DOF the computation will explode. A better approach is to using additional heuristics to specify the problem on a semantic level. To formalize the task knowledge, a dedicated ontology has to created first. In most cases the ontology uses a vocabulary and describes the domain in natural language. In the following paper some details are given about this problem solving technique.

2 Ontology based robotics

Ontology is a word which was first used in the context of cognitive robotics. It describes the usage of "semantic networks" for describing the domain knowledge. OK, that was not the best explanation so let me give a short example. Suppose you want to build a walking robot. In normal non-ontology based robotics you would invent some form of algorithm. In most cases an algorithm which acts as a solver. In contrast, with cognitive-robotics not the algorithm is in the center of the focus, instead a network of concepts. In the easiest which is similar to a mindmap.[26] In more complex cases the semantic network is forming RDF triples, UML charts and class-diagrams.

This thesis postulates, that ontologies for robots are the major problem solving technique especially for very complex tasks like biped robots, dexterous grasping and human-like behaviors. The reason why ontologies are important is that other problem solving techniques which are used in traditional robotics like algorithm and planners will fail to solve complex task. A solver which uses minimax for searching into the gametree of chess or four-in-one-row is not suitable for finding the gait patterns of a walking machine. Only ontologies can do this job well.

After this short motivation there is the question to answer in what form a robot ontology has to be formulated. Sometimes, the programming language Prolog is used for building an ontology.[30] But Prolog is not the language of choice, because it lacks in game-based programming, physics-engine and – very important – speed. An ontology itself is not bound onto a dedicated language. A normal object-oriented programming language is sufficient. The best of visualizing a robot ontology is a UML chart. There are some classes which has methods and attributes. That are the major components of an ontology. For reason of simplification it is right to postulate the ontologies and class-diagrams are using the same principle.

The difference between the two is that a class diagram is normally static. The classes are written in sourcecode and can't modify itself during runtime. An ontology, especially an rdf-triple ontology can be changed during program-runtime. It consists of data like an inmemory-sql-database. But, for practical purpose this feature is not very important, a static ontology is sufficient in most cases. Not the attribute and methods as a formal pattern have to be changed, instead only the values of the attribute are changeable like normal variables.

The reason why some authors in the context of RDF-based ontologies are using memory-based ontology which can change completely during runtime is the hope that the ontology can be learned on a magical way.[27] The aim is to start the program with an empty ontology and to fill the gaps during runtime.[19] That is true in cases if neural networks and reinforcement learning are used for building such ontologies. But, the truth is, that this kind of automatic programming will not work. There is not practical example that a solver can build up an ontology without help from outside. The most – or better, nearly all – ontologies are handmade and change not their structure during runtime. Ontology modeling has much in common with traditional object oriented design and is done manually by programmer in edit-compile-run cycles.

2.1 Computer based training

A well formulated ontology is capable for controlling the robot in the given domain. The ontology acts as a supervisor, a planner, a method-library and is capable to react even in uncommon situations. But, that is only the goal not the way how to create such ontologies. The process of creating have to be done from scratch and in most cases it is unclear what the structure of the ontology is. So there is the need for using a software-engineering-process which leads in developing a well formulated ontology. The best practice is to start slowly with a semi-automatic surveillance ontology which guides not the robot but the human-operator. This kind of software is called computer-based training.

I want to give some details about this kind of tools. Normally robots were programmed with the aim of fully-autonomous systems. The robot should do his task alone without any help from outside. The creation of such systems is very difficult so in most cases the software-engineering process will fail. That's the reason why until now the number of working robots is very low. A better way is to reduce the requirement for a robotics system. Instead of implementing a fully-autonomous system only semi-autonomy is needed. That means, the robot act more than a decision support system which observes the human user and give advice.[23] In the literature this interaction mode is described as educational software. The idea is to teach tasks to a human operator, and he (not the robot) learns to drive a car, grasp an object or stand up.

Instead of programming a sophisticated robot-control-system the task for the engineers is reduced to programming a simple teaching software. A concrete method is to use an ontology based authoring tool like SmartTrainer.[17] These systems are not sufficient for controlling real robots, instead they have much in common with edutainment-software and serious games.[6] The main idea is to program some kind of computergame and integrate task-ontologies for describing situations from the real live like flying an airplane, building a city or operate minimally invasive.

2.2 Object-oriented animation

A robot-ontology is nothing more than a computerprogram written in an object-oriented language. Since the 1980s some research projects are done in the context of object-oriented computer animation, e.g. ASAS, HIRES and JACK. ASAS (Actor/Scriptor Animation System) was used in the Disney movie TRON and was build on the LISP programming language.[20] The aim of these research projects was to build an animation system which is controlled by natural language. A command like "move" triggers actions in a 3D world.[11] Instead of sketching lines, the animation process is task-oriented. There is a vocabulary in the-loop which holds the domain-knowledge and with that vocabulary the user activates the animation routines.[10] A concrete example of how such systems work is given in [24, page 11-12] There are some nice screenshots given of the animation software. On the one hand a man is standing on the ground, and as overlay the textual commands are visible which are driving the virtual actor.

From the point of view of how to build such systems the most remarkable aspect of the projects which are cited above is that domain-ontologies are created in every case manually by hand. No sophisticated artificial intelligence techniques like reinforcement learning, general problem solving or self-modifying code was used. The base of object-oriented animation is a handcoded computerprogram which consists of parameters and methods which are oriented on the domain. The knowledge about inverse kinematics, positions in the scene, dynamic walking, walk-cycles and the difference between a smiling and a sad actor are hardcoded in a programming language. The older ASAS system was build upon LISP, modern animation engines are using modula-2 or C++.

Computeranimation is a good example for narrow AI. The aim is to use the computer to do a specific task. Computers can be used for raytracing, calculate the color or to animate a movie. With computerscience or artificial general intelligence this has that nothing in common. Instead, the computer is only the instrument to do something better. Computer in that meaning are used by artists to create something. If they use not a digital computer, they would use a pencil for painting cartoons. Even Robotics is not computerscience in its core meaning. Robotics is like computeranimation only a domain. The aim is to program a mechatronic system. That system deals with kinematics, servo-motors and electric current. Robotics can also be described as narrow AI.

So the golden way of understanding robotics, computer animation and any other field outside the computer science is given by the domain itself. Computernnimation is primarily based on animation, not on computers, and robotics is based on mechanics not computers. But if these domains are located outside of core computerscience what is the fundamental inner circle of computing? That is often called theoretical computing or Artificial General Intelligence. Possible subjects are turing-machine, Kolmogorov complexity, self-modifying code, recursion and most important a strong similarity to mathematics. The bad news is, that with these core computerscience it's impossible to solve one of the major problems like robotics or computer animation. And yes, core computer science is often useless. That is the main reason why Strong AI was replaced by narrow AI in the 1980s.[1]

For the practical working for roboticists, game-programmer and experts for artificial intelligence the effects can't be greater. They are no longer computer scientists, but they are expert for their domain outside of computing. Experts for music, animated movies, walking machines or whatever. If major robotics projects fail, in most cases the team has too much knowledge about computing, and too less knowledge about the concrete domain.

2.3 Ontologies vs optimal control

Ontologies are not the only possibility for solving robotics problem. Some literature uses an idea named "optimal control". Optimal control is more common and it is the natural way of describing robotics problem in mathematical terms. A good example is biped walking which can be expressed with a formula which calculates the ZMP (zero-movement-point). So, the robotics problem is converted to an abstract mathematical issue. The movement of the robots are given by the planner who solves it.[22]

The trouble with optimal control in general and ZMP calculation in detail is, that biped walking is more complex than solving an equation. Using an ontology instead of calculating a formula means, that the semantics of the question has to be understood. Ye problem "biped walking" is no longer an equation which can be iterated by a solver, instead the situation is seen as a mindmap with special vocabulary. The programming technique of implementing an ontology has much in common with object-oriented programming. The problem has to be divided into subproblems (aka classes) and every class has methods (aka the vocabulary). Solving the biped walking task is no longer done by solving one equation, but it is to paint an UML chart and know the different methods from the classes like "move a leg up", "search for footplace" and so on.

It is correct, that in most cases, some mathematics is needed, for example for calculating the inverse kinematics. And sometimes a dedicated solver and even brute-force-search is part of the solution. But, that is not the way how to implement the robot-control-system. The control system as whole works more like an ontology. A network of classes which consists of special vocabulary which is the same like experts use if they talk about the issue.

2.3.1 Qualitative Reinforcement Learning

Reinforcement Learning is a well known problem solving technique for robotics. Most famous examples as the "mountain car problem" and the "cartpole problem" have demonstrated it's potential. Both are examples for finding the policy for optimal control. Even if pure reinforcement learning is able to solve these tasks a more advanced approach is needed. This is called qualitative reinforcement learning[9] and can be classified as a hybrid system: prior knowledge plus reinforcement learning is used together. But if hybrid systems works, why we need something else than an ontology? The answer is that ontology-only problem-solving is better than hybrid systems. Because the black box optimizing techniques which are used inside reinforcement learning are only capable for solving toy problems. Ontology only optimal control is not well established in the literature, but it is capable of solving more advanced problems and is the here-to-stay technique for future robotics.

[1]Strong AI can be seen as modern form of cybernetics. Computers are build as a replacement for man.

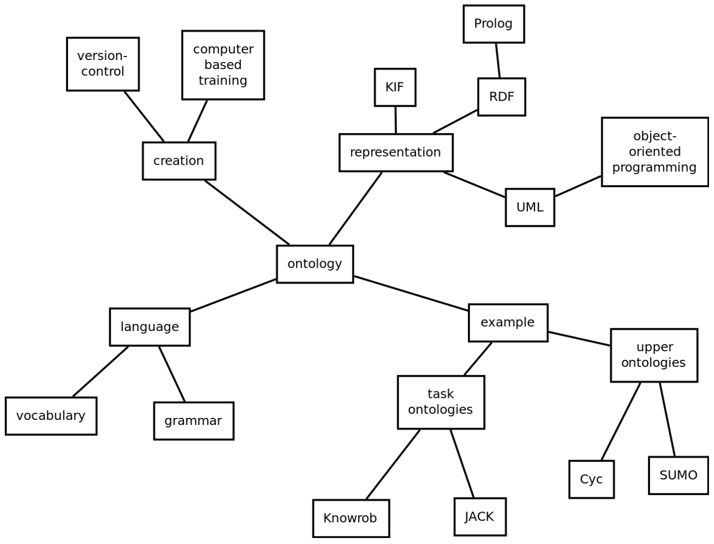

Figure 3.1: Mindmap

2.4 User Interface

An ontology can be seen as a user interface between man and machine. It is comparable to other interactive controllers like GUI Systems or mouse-movements. It has no direct functionality for the computer but it's more a tool for better communicating between human-operator and software. Ontologies consists often of a lexicon, perhaps with words like "move", "jump left" and "pickup". From a technical point of view, this vocabulary makes no sense, because the computer itself needs no natural English description. He is very fine with method names likes "action1", "action2" or even better with function names in hexadecimal syntax like "action#A0", "action#0B". But, if the user interface has clear action names with a common sense meaning, it is easier for novice users to interact with the computer. He understands intuitive on which button he has to click and can program scripts which consists of many subactions with less effort.

3 Example ontologies

- Route graphs in Haskell [18]
- Robocup soccer [8]
- Virtual humans (mostly JACK) [3]
- Upper ontologies: SUMO (largest), Cyc, DOLCE [21]
- Robotics ontologies: KnowRob, CORA [4]

- Upper ontologies for computeranimation: AIM@SHAPE [13]

3.1 Dance notation

In the Startrek episode TNG 4x11 “Data’s day” the android data learns on the holodeck in the famous scene with Dr. Crusher how to dance. The exact working of data’s positronic brain was not given in the episode and perhaps Gene Roddenberry didn’t even know it. The way of how an android learns how to dance is not located in the the brain of the android, but in his environment. The theoretical foundations of “how to dance” is called Labanotation and has a long history.[15] It is only known by dance-experts not by computer scientists. But, if the task is to program a robot how to walk, dance and move, the art of Labanotation is the correct notation. The above cited book is not a book about computer programming or robotics. It is a classical book about dancing. In most cases it is read by art students in context of ballet dance, and is not very common in mainstream. The task which computer programmer has to be done is to translate this and similar books into C++ language. That is the best-practice method for programming a robot to move like a human-dancer. And that’s the way how to teach data in the introduction sentences of how to dance with Dr. Beverly Crusher.

3.2 Critics on SUMO

SUMO is called an “upper ontology”, which is equal to a meta-ontology. An ontology for building other ontologies. Sumo is open source and can be downloaded from the webserver. But in reality the project is useless for robotics. On the one hand, Sumo is nearly perfect. It is a well elaborated ontology, has many facts and rules and even consists of some predefined applications like medicine, transportation and business. But the problem is, that SUMO not really have knowledge inside his database. It is more like a SQL table which has to be filled with information. Sumo alone lacks of semantics.

Perhaps, my critics is a bit harsh but what is the alternative to Sumo? A natural form of storing knowledge is a Wikihow tutorial. These “step by step” guides have information which can be interpreted by humans for doing a concrete task. Another useful information source for humans is Wikipedia itself. In contrast, Sumo can not compete which this rich information sources. On the other hand there is the computer which also deals with information. Computers are programmed in languages like C++. For executing a task this type of software has to be programmed. Here also Sumo lacks of capabilities. Sumo can’t be executed or converted to C++ code. It is more like an XML table. So even for interpreting by machines Sumo makes no sense.

As a conclusion, Sumo has no meaning for humans and no meaning for computers. Sumo is more like a trial to formalize knowledge, This trial was not successful. An “upper ontology” is an anti-pattern of how not transfer knowledge to machines. But why it is so difficult to invent an intermediate language between C++ on the one hand and normal English on the other side? I don’t know. Even highly sophisticated dedicated robotics ontologies like Knowrob, which have demonstrated in practice that they work are not really a here-to-stay in robotics. Knowrob is difficult to understand and perhaps it would be easier to program the sourcecode in normal way.

[13, page 49] differentiates between different types of ontologies. On the low end, there are domain- and task-specific ontologies. On the next level are so called “upper ontologies” which have common knowledge and on the first level are Generic ontologies which are even more “meta” than the upper ontologies. Usefull in the sense that humans or machines can do something concrete with an ontologies are only the domain- and task-specific ontologies.

3.3 Robotics domain

In [25] the development of a robotics ontology is described. The authors idea is to use a semantically grounded language and formalize this language into an ontolgy. On page 2 the paper lists existing robotics ontologies from domains like Sensor, pathplanning, motion planning and event-processing. On page 3 is the own sub-ontology given from the domain Action. It looks like a chart which consists of:

- drivefaster
- stopdriving
- changelocation
- changedirection
- turnleft

So, the domain is subdivided into linguistic categories and probably these categories are mapped into sourcecode. The more abstract robotics ontologies are similar to a classic robotics Firmware like Lejos where a operating system has access to the hardware of the robot (brick). The mid- and lowlevel ontologies are more domain-specific and consists of detailed descriptions like followline or turn-left. Working code is not given in the above paper, the author states modest:

> “We have developed a draft version of ontology for the robot programming domain. ” [25, page 5]

4 Storing knowledge

The natural way to store knowledge are languages like English. In Wikihow-articles, glossaries, lexicons and books are scientific relevant information saved. Unfortunately, computers are unable to parse this kind of information. Some call this a knowledge gap between human and machines. The information about how to walk, doing tasks or driving car are on the one hand given in a explicit way (as books), but computers fail to interpret this knowledge.[2]

The only language computers understand are programming languages. In the history of computing many languages were invented, and some of them with the aim to store knowledge. The most dominant language is SQL for storing knowledge in rows and columns, but for semi structured data also languages exists like ABL, Prolog and KIF. The problem with ABL and the other examples of artificial intelligence languages is, that they not really capable to overcome the knowledge gap. There is no converter from normal English text to ABL programming language. And it comes even worse: so called languages of the fifth computer generation like Prolog and Lisp are not more suitable than normal general purpose languages like C++. So in every case a LISP program can be replaced by well written C++ code but not the other way around.

The question is: how can knowledge which is written in English transferred to C++ source code? The answer is simple: the transition is called programming, or better known as object-oriented domain-specific programming. It can be done by a single programmer, a team or even by a company. The task of transferring human-knowledge to computer-readable knowledge is not solvable by algorithms, but it is a challenge for humans. The conversation is a manual task which has to be done inside projects. It is comparable with creating the Wikipedia-encyclopedia which is also be done by real humans and not by algorithms.

4.1 Task ontologies

Task ontologies are the lowlevel end of ontology engineering. There are describing the process flow on a detailed level. But, concrete examples for this kind of ontologies are rare in literature. It seems, that tasks which are formulated in an abstract description are until now only a research topic and not ready for practical usage. The problem is the following: normally, ontologies are written in an ontology language, mostly based on XML. A concrete task has in contrast be formulated in a computer language like C++ or Java. But it's not possible to store C++ sourcecode in XML syntax, at least it makes no sense.

In gaming-AI context task ontologies are often implemented as behavior trees. This kind of workflow can be stored as abstract XML tree. But, here is also the problem that a behavior tree is not a real computer program. In most cases a behavior tree is more like a diagram which is used in the software-engineering process for defining the requirements for the application. In a real life-scenario the behavior tree has to be converted into executable sourcecode, mostly C++. So, task ontologies seems to me more like software-engineering tools like a UML diagram but not as a way to get working code.

Obviously the reason why in literature task ontologies are discussed is to formalize knowledge. The aim is not to write in a computerlanguage, but to describing problems at a general level which has nothing to do with programming. Formulating ontologies is that, what computer researcher do. Or should I say better: wants to do. The truth is that nobody wants to program, especially not at the C++ level with pointers and llvm-compiler in mind. But why? Mostly because programming is difficult, takes long time and is error-prone. But, using task ontologies is no help in the process, because at the end the ontology alone will not work. The ontology is only a concept drawing on the computerscreen, not working software. And without concrete software, the robot will also not work. Perhaps, task ontologies are a bit too abstract to be useful. The better way around is to focus more into the programming level. I would suggest using a version control system like git, for iterative programming in C++ and in the sourcecode the task-ontology is "populated". Not in the sense of ontology learning or automatic programming, but in the sense of writing lines of code in a real programming project.

4.2 Domain ontology

The idea of using an ontology for describing robotics tasks is a confession of weakness for the computer science. I mean, if the computerscience itself has high-quality algorithms and tools for program a robot to walk, think and act like real humans, these tools would be used on the first hand. The truth is that computerscience lacks of such tools instead ontologies are used for pointing outside of computerscience. Knowledge-engineering means, that domain knowledge from subjects like sports, games, clothing and animation is transferred into computerscience. So the process model for creating such ontologies from scratch is not given in computerscience but it is determined by the concrete subjects. If the aim is to build a dancing robot, than a dance artist gives the vocabulary and procedures which are part of the ontology. It the aim is to create a driving-ontology then the science of traffic will guide the creation of the suitable ontology.

Even if domain-ontology languages like OWL and KIF [16] exists a good starting point for creating a domain-specific ontology is the UML notation. [12] starts on page 5 with a normal UML diagram for describing a businessprocess which consists of customer, order and order line:

> "Hence, to assign business meaning to object-oriented languages, a mapping was created between the ontological concepts and objectoriented constructs." [12, page 13]

Ontology engineering extends methods of object-oriented software engineering with concrete applications like natural language processing and business process modeling:

> "Putting ontological engineering in the context of other disciplines enables both ontological engineers and other specialists to view their fields from different perspectives." [7]

Reverse engineering an ontology from given database-model [1] or existing sourcecode is also possible. [31] gives an example for translating legacy Cobol-program into an ontology.

Scripting AI UML based domain-modeling is not the first time when normal computerprograms for controlling robots was used. In early times this concept was called Finite State Machine or Scripting AI and his meaning was to use normal sourcecode for realizing complex robot systems. UML based ontologies extends this basic idea to something more advanced. Instead of using simple macros and behavior-scripts, a complex ontology which is spreading over many C++ classes is used. The new idea is, not to using sophisticated AI programming techniques instead the artificial intelligence is realized in a common software-engineering process. Not algorithms are in the center but it is the knowledge acquisition process. This kind of software-development-cycles are driven by the domain. Like in former chapters described the animation artist, linguist or car-experts defines how the final program will look like.

Ontology engineering starts with the naive statement that computerscience itself has very little to offer for the overall development cycle. The only working technique is a programming language like c++, some object-oriented programming and perhaps a physics-engine. That's all. The rest, especially the question of how the robot looks like, and how he will be animated is decided not by software-engineers but by artists, designers and animation experts. They have the important knowledge and tell the programmer how he must implements this in C++ sourcecode. If the robot doesn't walk, not the guys from the computerdepartment have made a mistake, but it was the animation expert. He was not able to tell exactly what the position of the legs should be.

4.3 Why ontology sharing will not work

If a formal ontology was created it would be nice if the knowledge can shared among other software so that quasi-automatic programming will be possible. The lingua franca in ontology engineering is KIF (Knowledge Interchange Format). Inside KIF new facts and rules are defined by first-order-logic. That is a declarative programming language which has much in common with Prolog. The idea is simple: the knowledge base would be created in First-order-logic and can be translated from that formal specification in other programming languages which runs on a computer. Nice idea, but not very innovative. In the 1960'er the Multics operating system has something similar in mind. That is the reason why Multics was written in PL-1, a programming language which can be – in theory – generated from formal specifications.[5]

Until today first-order logic was not very successful in replacing normal procedural languages. Not Multics has conquered the mainframe, but UNIX and Linux which are both written in a non-declarative C programming dialect. It's obvious that this lesson can be transferred to knowledge engineering. I want to say, that first-order-logic for knowledge storing and sharing will not work.

A good comparison is the relationship between regular expressions and a C program. In regular expressions (which are used by awk and grep tools under Linux) simple rules can be formulated in surprising less programming effort. But, for advanced algorithm a real language like C is the only way to go. It's not possible to write a complete operating system in regular expressions. And that's the same restriction as in Prolog. Prolog is a good macro interpreter for writing small code pieces but it lacks in support for developing large software projects. Here is the object oriented paradigm in general and particularly C++ programming language the more comfortable approach.

Versioncontrol The better alternative to KIF and first-order logic is "versioncontrol based ontology development". That development cycle uses traditional distributed software-engineering techniques likes mailing-lists, wikis and git versioncontrol. Not a programming language like KIF, Prolog or PL-1 is in the center but a manual driven developer collective which is sometimes described as "the crowd". The crowd is the opposite to automatic programming, the crowd works completely human-driven but uses advanced techniques for communicating over Internet. A good example is the Hozo ontology editor [28] which has additional to the Protege Editor a feature for collaborative working on ontologies.

Ontology reusing Ontology sharing is sometimes called reusing. The aim is to create a formal abstract high level ontology and apply it to many different domains. Sometimes this wish is connected with topdown systems, were the core ontology consists of vocabulary for English and subontologies are derived from this concepts to more specific run-able applications. The bad news is, that until now no working example was demonstrated. It is only the hope of computer scientists that this engineering technique will work in future and some projects like Cyc and OWL have this vision in mind.

A possible explanation why top down ontology engineering will lacks is because automatic programming in general also fail. Automatic programming was the aim to develop operating systems and user-applications without programming and only with formal requirement specification. The idea was to use program generators which can build a Linux kernel and similar software from scratch. Let's make a short thought experiment: The task is to automate generation of a prime-number checker. Not by programming it in Java but by using automatic programming. In many papers techniques like genetic programming, Petri nets [14] and even Bayes classification is described as a hopeful technique to solve these problems, but if one of these techniques should be applied to the given domain it will fail. And if even a simple prime number generator can't be created via automatic toolchains, how should more complex domains like pathplanning or motion planning work? Yes, the pessimistic point of view is the right one, automatic programming and general ontology are not ready for solving real problems.

It is a mistake to think that programming without human in the loop will be successful. In every case, an ontology is some form of a computerprogram. And computerprograms are normally written by programmers. To improve their productivity some tools are available like high-level programming languages, Stackoverflow question&answer website and versioncontrol. But a tool for automatic generate a runable ontology from natural language without any intervention of human-programmer is purely science-fiction. I would suggest that even in 20 years such tools are not available. So the only possibility for creating robotics ontologies is to use normal conservative techniques. If one person alone can not create complex software, that more programmer are be needed. The question is: how many? Are 100 programmer enough for bulding a walking ontology for a human-robot?

5 Object oriented programming

5.1 OOP = ontology?

On the first impression there is no difference between object-oriented programming and ontology engineering.[29] In both cases the problem will be subdivided into a domain-specific vocabulary. But there is a difference. Object-oriented programming is ready for practice, ontology engineering not. Ontology engineering has it's roots into Strong AI. The so called general problem solvers based on first-order logic are ontology based systems. This tradition was merged into current semantic web and OWL definitions. Virtually unchanged is the aim of describing knowledge in an abstract way. The main difference between OOP and ontologies is, that OOP works, while ontologies not.

Nevertheless in current literature about robotics ontology not object-oriented programming is discussed but ontology engineering. An ontology is often used as a replacement for an API (application programming interface). The ontology is the front-end for an object-oriented program. It consists only of the names of the methods and no sourcecode is given. The advantage is two-folded: on the one hand, ontologies haven't to be complete programs. They are more concepts and requirements for the future. And the second advantage is that while using ontologies and not real C++ code, the author can describes commercial patent restricted software which is not public available under GNU public license. So the usage of ontologies and not of object-oriented programming has often legal reasons.

References

[1] Aftab Ahmed Abbasi and Narayanan Kulathuramaiyer. A systematic mapping study of database resources to ontology via reverse engineering. *Asian Journal of Information Technology*, 15(4):730–737, 2016.

[2] Andrea Addis and Daniel Borrajo. From unstructured web knowledge to plan descriptions. In *Information Retrieval and Mining in Distributed Environments*, pages 41–59. Springer, 2010.

[3] Norman I Badler, Cary B Phillips, and Bonnie Lynn Webber. *Simulating humans: computer graphics animation and control.* Oxford University Press, 1993.

[4] Stephen Balakirsky, Craig Schlenoff, Sandro Rama Fiorini, Signe Redfield, Marcos Barreto, Hirenkumar Nakawala, Joel Luís Carbonera, Larisa Soldatova, Julita Bermejo-Alonso, Fatima Maikore, et al. Towards a robot task ontology standard. In *ASME 2017 12th International Manufacturing Science and Engineering Conference collocated with the JSME/ASME 2017 6th International Conference on Materials and Processing*, pages V003T04A049–V003T04A049. American Society of Mechanical Engineers, 2017.

[5] Boumediene Belkhouche. Generation of ada and pl/1 prototypes from abstract data type specifications. *Journal of Systems and Software*, 16(3):255–264, 1991.

[6] Paolo Busetta, Chiara Ghidini, Matteo Pedrotti, Antonella De Angeli, and Zeno Menestrina. Briefing virtual actors: a first report on the presto project. In *Proceedings of the AI and Games Symposium at AISB*, 2014.

[7] Vladan Devedzić. Understanding ontological engineering. *Communications of the ACM*, 45(4):136–144, 2002.

[8] Frank Dylla, Alexander Ferrein, Gerhard Lakemeyer, Jan Murray, Oliver Obst, Thomas Röfer, Stefan Schiffer, Frieder Stolzenburg, Ubbo Visser, and Th Wagner. Approaching a formal soccer theory from behaviour specifi cations in robotic soccer. *WIT Transactions on State-of-the-art in Science and Engineering*, 32, 2008.

[9] Arkady Epshteyn and Gerald DeJong. Qualitative reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 305–312. ACM, 2006.

[10] Jeffrey Esakov and Norman I Badler. An investigation of language input and performance timing for task animation. 1988.

[11] Jeffrey Esakov, Norman I Badler, and Moon Jung. Human task animation from performance models and natural language input. 1989.

[12] Joerg Evermann and Yair Wand. Ontology based object-oriented domain modelling: fundamental concepts. *Requirements engineering*, 10(2):146–160, 2005.

[13] Alejandra Garcia Rojas Martinez. Semantics for virtual humans. 2009.

[14] Dragan Gaševiü and Vladan Devedžiü. Reusing petri nets through the semantic web. In *The Semantic Web: Research and Applications: First European Semantic Web Symposium, ESWS 2004, Heraklion, Crete, Greece, May 10-12, 2004, Proceedings*, volume 1, page 284. Springer, 2004.

[15] Ann Hutchinson Guest. *Labanotation: the system of analyzing and recording movement.* Routledge, 2014.

[16] Patrick Hayes and Christopher Menzel. A semantics for the knowledge interchange format. In *IJCAI 2001 Workshop on the IEEE Standard Upper Ontology*, volume 1, page 145, 2001.

[17] Mitsuru Ikeda, Kazuhisa Seta, and Riichiro Mizoguchi. Task ontology makes it easier to use authoring tools. In *IJCAI (1)*, pages 342–351, 1997.

[18] Bernd Krieg-Brückner, Udo Frese, Klaus Lüttich, Christian Mandel, Till Mossakowski, and Robert J Ross. Specification of an ontology for route graphs. In *International Conference on Spatial Cognition*, pages 390–412. Springer, 2004.

[19] Wei Liu, Albert Weichselbraun, Arno Scharl, and Elizabeth Chang. Semi-automatic ontology extension using spreading activation. *Journal of Universal Knowledge Management*, 1(1):50–58, 2005.

[20] William E Lorensen and Boris Yamrom. Object-oriented computer animation. In *Aerospace and Electronics Conference, 1989. NAECON 1989., Proceedings of the IEEE 1989 National*, pages 588–595. IEEE, 1989.

[21] Viviana Mascardi, Valentina Cordì, and Paolo Rosso. A comparison of upper ontologies. In *WOA*, volume 2007, pages 55–64, 2007.

[22] Katja Mombaur, Anh Truong, and Jean-Paul Laumond. From human to humanoid locomotionon inverse optimal control approach. *Autonomous robots*, 28(3):369–383, 2010.

[23] Edwin Olson, Johannes Strom, Ryan Morton, Andrew Richardson, Pradeep Ranganathan, Robert Goeddel, Mihai Bulic, Jacob Crossman, and Bob Marinier. Progress toward multi-robot reconnaissance and the magic 2010 competition. *Journal of Field Robotics*, 29(5):762–792, 2012.

[24] Cary B Phillips and Norman I Badler. *Interactive behaviors for bipedal articulated figures*, volume 25. ACM, 1991.

[25] Ignas Plauska. Ontology for robot programming domain. In *XV International PhD Workshop OWD*, 2013.

[26] Biplab Kumer Sarker, Peter Wallace, and Will Gill. Some observations on mind map and ontology building tools for knowledge management. *Ubiquity*, 2008(March):2–1, 2008.

[27] Javier Snaider, Ryan McCall, and Stan Franklin. The lida framework as a general tool for agi. *Artificial general intelligence*, pages 133–142, 2011.

[28] Eiichi Sunagawa, Kouji Kozaki, Yoshinobu Kitamura, and Riichiro Mizoguchi. An environment for distributed ontology development based on dependency management. *The Semantic Web-ISWC 2003*, pages 453–468, 2003.

[29] Florian Weber, Andreas Bihlmaier, and Heinz Wörn. Semantic object-oriented programming (soop). *Informatik 2016*, 2016.

[30] Jan Wielemaker, Guus Schreiber, and Bob Wielinga. Prolog-based infrastructure for rdf: Scalability and performance. In *International Semantic Web Conference*, volume 2870, pages 644–658. Springer, 2003.

[31] Hongji Yang, Zhan Cui, and Paul O'Brien. Extracting ontologies from legacy systems for understanding and re-engineering. In *Computer Software and Applications Conference, 1999. COMPSAC'99. Proceedings. The Twenty-Third Annual International*, pages 21–26. IEEE, 1999.